Universität Rostock

Traditio et Innovatio

Investigating the Role of Software in Science by Automatic Knowledge Graph Construction through Natural Language Processing

Dissertation
zur
Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)
der Fakultät für Informatik und Elektrotechnik
der Universität Rostock

vorgelegt von

David Schindler aus Burglengenfeld

Berlin, 2024

https://doi.org/10.18453/rosdok_id00004777

Gutachter:

Prof. Dr. Sascha Spors, Universität Rostock, Signal Theory and Digital Signal Processing

Prof. Dr. Frank Krüger, Hochschule Wismar, Data Science and Machine Learning

Prof. Dr. Stefan Dietze, Heinrich Heine Universität Düsseldorf, Data & Knowledge Engineering

Jahr der Einreichung: 2024

Jahr der Verteidigung: 2024

Abstract

Software has become increasingly important in data-driven research and plays an integral role in today's science, contributing to all steps of scientific investigations. Knowledge of how software is applied in scientific investigations is essential to the scientific community in terms of provenance, because knowledge of utilized software is a prerequisite for reproducibility and software influences the outcome of scientific research. Furthermore, tracking the usage and impact of software provides a central feedback mechanism for the development of research software, a task often performed by researchers themselves. However, knowledge of software application in research is sparse, with a majority of information on software usage being informally described in the textual descriptions of scientific research, leading to a lack of software visibility. This thesis, develops a method to systematically analyze software usage in scientific publications at a large scale. For this purpose, I created the high-quality ground-truth dataset SoMeSci, which is based on manual annotation covering all relevant knowledge of software usage in scientific publications, such as software metadata (e.g., versions or developers), contextual information, and unique identifiers. Findings on this dataset revealed that providers of bibliographic data currently use unsuited representation formats for formal software citation, hampering systematic analyses of software citations. I therefore developed an automatic information extraction pipeline for software mentions in scientific articles, solving the problem with a high recognition rate of 86.6% in terms of FScore. However, my results also demonstrate the complexity of the task, highlighting the challenges of generalization and the complexity of the required Entity Disambiguation. The established extraction pipeline was applied on a large-scale dataset of 3.2 M articles from PubMed Central and the resulting data was formally modeled in the Research Knowledge Graph SoftwareKG, to allow a FAIR publication and reuse by the scientific community. Finally, I performed large-scale analyses to provide insights on software usage over time, between domains, in dependence of article impact, and concerning the state of open source software, as well as specific analyses tracking the publication of software, extendable software architectures, and the relation between software and article retraction. Overall, the resources of SoMeSci and SoftwareKG—made available in the scope of this work—build the bases for further analyses of software in science and can serve as a starting point to implement applications required by the scientific community, such as software impact measures or a software recommendation system.

Zusammenfassung

Software hat in der datengestützten Forschung zunehmend an Bedeutung gewonnen und spielt in der heutigen Wissenschaft eine entscheidende Rolle, da sie in allen Schritten von wissenschaftlichen Studien eingesetzt wird. Das Wissen darüber, wie Software in der Forschung eingesetzt wird, ist für die wissenschaftliche Gemeinschaft in Hinblick auf Provenance unerlässlich, da die Kenntnis der verwendeten Software eine Voraussetzung für die Reproduzierbarkeit ist und Software Auswirkungen auf Studienergebnisse hat. Darüber hinaus dient die Verfolgung der Verwendung und der Reichweite von Software als zentraler Feedback-Mechanismus zur Entwicklung von wissenschaftlicher Software, welche häufig von Forschenden selbst übernommen wird. Jedoch ist das Wissen über den Einsatz von Software in der Forschung begrenzt, wobei die meisten Informationen dazu informell im Text von wissenschaftlichen Veröffentlichungen beschrieben werden, was zu einer mangelnden Sichtbarkeit von Software führt. In dieser Arbeit wird eine Methode zur systematischen großflächigen Analyse der Softwarenutzung in wissenschaftlichen Publikationen entwickelt. Zu diesem Zweck habe ich den hochwertigen Ground-Truth Datensatz SoMeSci entwickelt. Dieser basiert auf manueller Annotation, die alle relevanten Informationen über die Softwareverwendung in wissenschaftlichen Publikationen, wie Metadaten (z. B. Versionen oder Entwickler), Kontextinformationen und eindeutige Identifizierungen enthält. Ergebnisse aus diesem Datensatz haben gezeigt, dass die Herausgeber bibliografischer Daten für formale Softwarezitate derzeit ungeeignete Repräsentationen verwenden. Aufgrund dessen habe ich eine automatische Informationsextraktions-Pipeline für Softwareerwähnungen in wissenschaftlichen Artikeln entwickelt, welche das Problem mit einer hohen Erkennungsrate und einem FScore von 86,6% löst. Meine Ergebnisse zeigen jedoch auch die Vielschichtigkeit des Problems, vor allem die Herausforderungen der Generalisierung und die Komplexität der erforderlichen Entity Disambiguation. Die entwickelte Extraktions-Pipeline wurde auf einem großen Datensatz mit 3,2 Mio. Artikeln aus PubMed Central angewendet und die resultierenden Daten wurden formell im Research Knowledge Graph SoftwareKG modelliert, um eine FAIR-Veröffentlichung und die Weiterverwendung in der Wissenschaft zu ermöglichen. Schließlich habe ich umfangreiche Analysen durchgeführt, um Einblicke in die Softwarenutzung im Lauf der Jahre, zwischen verschiedenen Forschungsbereichen, in Abhängigkeit der Reichweite eines Artikels und in Bezug auf den Status von Open-Source-Software zu gewinnen. Zudem habe ich spezifische Analysen durchgeführt, welche die Veröffentlichung von Software verfolgen, und erweiterbare Softwarearchitekturen, sowie den Zusammenhang von Software und dem Widerruf von Veröffentlichungen untersuchen. Insgesamt bilden die Datensätze SoMeSci und SoftwareKG, die im Rahmen dieser Arbeit öffentlich verfügbar gemacht wurden, die Grundlagen für weitere Softwareanalysen in der Wissenschaft und können als Ausgangspunkt für die Entwicklung von Anwendungen dienen, die von der wissenschaftlichen Gemeinschaft benötigt werden, wie z.B. eine Methode zur Messung der Reichweite von Software oder ein System zur Softwareempfehlung.

Contents

List of Figures

List of Tables

List of Listings

1 | Introduction

Software has become increasingly important in data-driven research and plays an integral role in today's science [104, 93]. It contributes to all steps of scientific investigations, ranging from the recording and analysis of data to the visualization of research results [119]. Surveys across scientists further highlight the impact of software reporting that 91% to 95% of scientists use software in their research [93, 193] and that 63% are convinced they could not perform their research without it [193]. Furthermore, scientists increasingly take up the role of software engineers themselves, with 84% reporting that developing software is essential for their research [93]. Not surprisingly, an analysis of the 100 most cited articles unveiled multiple articles describing software which are central research tools in different research fields, for instance, *BLAST*[1] and *ClustalW* in the area of Bioinformatics or *SHELX*, *HKL*, and *PROCHECK* in Crystallography [198]. Historically, however, the publication of a software has been considered a weaker contribution than the publication of an article [101]. By now, the impact of software on science has been recognized by the scientific community and it is considered as a key research result and resource [267, 81], and one of the main pillars of science—besides articles and data—as it contains the logic of data transformation [68]. Since software facilitates, or even enables, scientific investigations, its impact on research results has to be considered by the scientific community. In practice, software errors have been found to bias study results and have led to article retractions [134]. One of the first examples, is the retraction of five impactful articles published around 2001, three of which published in the journal *Science*, due to one error in the self-developed analysis software used throughout all five investigations, which flipped two data columns [186].

1.1 Research Software and Software in Research

Since the importance of software in science increases, various studies proposed definitions for scientific or research software. A definition is required by different stakeholders, such as researchers or funders of scientific research, since there is an increasing need for funding specifically allocated for the development of research software, as researchers themselves fill the role of software developers [116]. Moreover, this creates an increasing need for proper credit and attribution for the development of software. The work of Goble [93] summarizes existing definitions of scientific and research software. The term *scientific software* has, for instance, been defined as software that answers a scientific question, is developed with a domain expert, and provides output data to support the scientific work [138]. It has also previously been considered as software developed by scientists for scientists [262], while the term *research software* has been defined similarly, referring to software developed in academia and for research purposes [110, 195]. This also coincides with the definition made by other stakeholders, e.g., by the Journal of Open Source Software (JOSS) [133]. Recently,

[1] Note that software is a research object of this work and is not formally cited when it is mentioned in that context. All software that was applied in the scope of this work is formally cited.

the Research Software Encyclopedia (RSEPedia) set out to define the term *research software* in a community driven, open source effort [264] (see also [279]). It concludes that a clear definition on a single page cannot be reached. Instead, a context dependent definition needs to be derived that best reflects the needs of a stakeholder.

In the remainder of this work, the notion of *software* is used to refer to all software used for research, instead of *research software* specifically, as all software used for research contributes to the research results and is part of the research's provenance. Therefore, this work explicitly includes software that was not developed by researchers or in the scope of academia. Particularly, the knowledge of potentially unsuited software in scientific investigations can be crucial to identify errors or biases in investigations, with a prominent example being the use of *Excel* for analyses involving gene names [317].

In the scope of this work, software is distinguished based on its availability, since access to specific software used or created in investigations can be a prerequisite for repeatability and reproducibility of the results [148]. Moreover, incentives for the creation of software differ between commercial and freely available software [75]. In general, research software established based on public funding is free, and should—following good scientific practice—be published open source [105]. Therefore, it depends on proper attribution as a feedback mechanism for both developers and funders, e.g., to secure future funding for extension and support. Commercially distributed software do not depend on attribution as their developers are financially compensated, e.g., by licensing fees. The work of Howison and Bullard [118] found that the attribution of software in scientific publications differs based on its availability. Specifically, it showed that proprietary software, in terms of source code availability, is more commonly cited similar to scientific instruments, while non-commercial and open source software is more likely to be formally cited. In this work, the availability of software is distinguished based on whether it is free to use in academics between *free* and *commercial* software. Furthermore, it is distinguished between *open source* software and *closed source* software, with proprietary source code. In general, both aspects are related with a majority of *open source* software being also *free*, and most *commercial* software being *closed source*, however, exceptions are possible. Note that software which is stated to be available upon request is not considered as *open source* in this work, since prior work has revealed sever issues regarding this form of data sharing [87, 278].

1.2 The Importance of Knowledge of Software Usage

In general, knowledge of software is important for the scientific community in terms of provenance. From a microscopic perspective, the knowledge of utilized software is a prerequisite for reproducibility of study results, as software includes the processing logic of most investigations. From a macroscopic perspective, knowledge about software in scientific investigations enables large-scale analyses of software impact, similar to evaluation metrics for scholarly publications [136]. Therefore, different stakeholders have an interest in knowing about software in science regarding its development, usage, impact, and the interaction between different software [119]. Researchers developing software, for instance, are interested in how their software is used and re-used, because feedback from the user community is important for active maintenance and improvement. For them, attribution is also of high importance, as they often spend significant parts of their academic careers developing and maintaining software [258]. Moreover, an impact measure providing credit for software use and re-use in investigations can serve as an incentive for software publication and maintenance. Knowledge of a software's impact is also crucial to funding agencies supporting the development of scientific software [119]. Similarly, outlets for software publication, such as software journals, could track the re-use of published software not only by the citation of corresponding articles but by tracking the actual usage of the software.

Knowledge of software is further relevant to the scientific community because software contributes to research results and influences study outcomes. Therefore, it is essential that the quality of software used in research is assessed and that software is evaluated [94, 116]. However, researchers often use software without it being thoroughly validated. Since proprietary source code cannot be inspected and prevents an evaluation of the implemented algorithms, this is often the case for commercial software [237]. The potential pitfalls of software errors were already highlighted above, with a single software error leading to the retraction of multiple high impact articles [186]. With the increasing importance of software for research, there have also been more recent reports of research data and results compromised by employed software [80, 325, 317]. For instance, 31% of human genome data handled with Microsoft Excel has been shown to be altered by the software's auto conversion features [1], while a bug was recently reported in the Python Machine Learning (ML) toolbox Scikit-Learn, which leads to a miscalculation of FScores in specific versions of the software [36]. The influence of software on scientific research can be further highlighted by examples from Retraction Watch (RW) [238], where errors in software usage or unintended behavior of software significantly impacted study results. One article, for instance, had to be retracted because Microsoft Excel re-ordered single data columns upon import [144, 183], while another retraction was caused by data rows shifting between observation groups when exporting data from Excel to SPSS [297]. Knowledge of where software is applied, would enable automatic quality checks and allow to alert authors when software bugs are discovered and fixed. It could further enable an automatic tracking of misused software and the recommendation of alternatives, combining quality control and software suggestion, described below.

Knowledge of software in science is further important for researchers looking to identify suitable software for their research. In general, finding suitable software is considered a challenging task [118, 94] and a variety of tools is often available. Software support in Biology has even been described as overwhelming and difficult to select due to the large amount of existing tools [236]. Therefore, researchers use different strategies to identify software for their research including web searches, opinions of colleagues, related literature, but also online forums or project repositories [122]. Here, researchers could benefit from better knowledge of available software and recommendation of alternatives, based on an underlying mapping of existing and used software in their own and related research domains.

The reasons stated above summarize the most important reasons why the scientific community requires knowledge of software usage in scientific investigations. However, there are several other aspects of interest beyond these most pressing issues, which are interesting in the scope of extended bibliographic analyses with respect to software in science. For instance, exploring the co-occurrence of software [165], analyses on software citation quality [118, 75], differences in use of commercial and open source software [118], or comparative analyses in software usage between scientific disciplines.

1.3 Software Mentions in Scientific Literature

As described above, software is widely used in scientific investigations and commonly related to data recording, processing, and analyses. Therefore, its use is part of the research's provenance and should be documented within scientific publications. In general, authors indicate software usage by in-text description in the full-text document of the publication, which can include a citation of the software by a formal bibliographic reference. In the remainder of this work, in-text software mentions will be referred to as *informal* citations and citations to bibliographic references as *formal* citations. An example of a software mention in scientific literature, containing both informal and formal citation, is given in Listing 1. The software is stated informally in-text with further metadata closer identifying it, including a formal citation to a bibliographic reference (details on metadata are introduced in Section 3.1). The corresponding provided bibliographic reference associated with the in-text software mention is given in Listing 2. In this case, it references an article associated with the software that was cited in its place. This is a common practice in scientific publications and is further described in Section 3.1.4.

Recently, multiple studies have investigated how software can be properly cited in scientific literature [263, 73, 57, 9] and software citation guidelines have been published [135]. These guidelines recommend formal citation of the software itself, rather than a corresponding article, to enable a unique identification of the software including its version. For this purpose, the guidelines define several metadata that should be included in a software citation such as the software's name, its creator, and its release date or version. Since the guidelines were recently established it is difficult to assess compliance by the scientific community, but funders and advisory groups are beginning to require software citation as mean to connect publications and its supporting software [267]. However, earlier and more recent work showed that software is more commonly mentioned informally than formally [118, 75]. Moreover, when software was formally cited, citations of software articles were found to be more common than citations of software itself [118]. The guidelines are also not fully implemented by scientific journals, yet. PLoS ONE, a large open source journal, for instance, considers software in its submission guidelines since 2019, instructing authors to: "List the name and version of any software package used, alongside any relevant references" [218]. However, the journal's policies do not consider direct citation of software itself.

1.4 The Lack of Software Visibility in Scientific Research

Currently, the majority of information on scientific software usage is contained in the full-text document of scientific publications, hidden from reference scanning mechanism, hampering its large-scale identification. Gathering this information manually is not feasible due to the large amount of existing scientific publications and the high volume of new publications. Moreover, software names are challenging processing targets due to ambiguities and inconsistent naming across publications [76], which impedes tracking unique software across scientific literature. Overall, this leads to a lack of visibility with respect to scientific software in publications, hindering development of applications as outlined in Section 1.2.

Currently, the scientific community has limited means to systematically investigate software citations compared to the citation of scientific literature, which is systematically analyzed in the field of Scientometrics by measures such as the h-Index. For the analyses of scientific literature, a broad infrastructure has been established that allows complex analyses, e.g., the examination of co-citation networks. The majority of bibliometric analyses depends on knowledge bases containing structured bibliographic information including references and their interrelation, such as Crossref, Semantic Scholar, or Scopus (see Section 3.2.3). While these databases can potentially only reflect

```
R language [30] (version 3.2.2; R Foundation, Vienna, Austria) was used for statistical
analysis and creation of several figures.
```

Listing 1: Example of an informal software mention from the dataset introduced in Section 3.2 including a name (🟦), formal citation (🟪), version (🟩), and developer (🟧). The example was extracted from Bigras et al. [32].

```
Ihaka R, Gentleman R. R: a language for data analysis and graphics. J Comput Graph Stat.
1996;5:299-314.
```

Listing 2: Formal bibliographic reference associated with the software mentioned in Listing 1 by citation #30, referring to an article published about the software *R*.

the fraction of formally cited software in the scientific literature, it is not clear whether their infrastructure can be utilized for the analyses of software citations, because they are differently structured and contain elements not present in regular citations, e.g., version numbers.

Overall, investigations are required to determine which information have to be systematically extracted from scientific articles to satisfy the requirements of the scientific community, and to determine how such an Information Extraction (IE) can be implemented. In general, automatic methods are required that process articles to systematically extract the hidden information on software usage and offer scalability to the large volume of scientific publications. Moreover, it is necessary to consider both, informal and formal software citations, the metadata provided with these mentions, and the unique identification and tracking of software across scientific literature. Specifically, the identification of software can be considered along three dimensions, the identification of the *software* itself required for tracking, the identification of its *developer* required for proper attribution, and the identification of the specific *codebase* by version or release data, necessary for reproducibility as software changes over time. Furthermore, the citation intent of scientific software can be important for analyses to identify the creation and publication of new software in articles, while distinguishing different types of software can also enable targeted analyses. Finally, all extracted knowledge needs to be semantically represented and provided to the scientific community to enable all researchers to access and work with this knowledge.

1.5 Contribution

In this work, I developed a method to systematically analyze software usage in scientific publications at a large scale. For this purpose, I created a high-quality ground-truth dataset based on manual annotation, covering all relevant knowledge of software usage in scientific publications, and developed an automatic IE pipeline for software in scientific articles. Further, I formally modeled the resulting data in a scientific Knowledge Graph (KG), to make it available to the scientific community as a FAIR [308] publication in terms of Findability, Accessibility, Interoperability, and Reusability. Finally, I performed large-scale analyses that give insights on software usage over time, between research domains, in dependence of article impact, and concerning the state of open source software.

Overall, the pipeline for examining software in science that I developed in this work does not only allow to examine software in science and implement features desired by the community, e.g., a software impact measure. The established pipeline also serves as a model on how knowledge from scientific articles can be systematically extracted and modeled, in order to implement the same process for other scientific artifacts, such as data. In this context, this work also serves as a demonstration of major caveats that need to be taken into account when developing similar approaches, e.g., Entity Disambiguation (ED) (see Section 4.4) proved to be a challenging problem, but was not yet taken into account by previous work assessing software in science (see Section 2.1)[2].

[2] Note that Istrate et al. [128] take ED into account, but their work was performed after the work described here.

1.6 Outline

The remainder of this thesis describes the development of a scalable, automatic, large-scale extraction pipeline for software mentions from scientific literature, the semantic modeling of the corresponding information, and analyses performed based on this data. Initially, Chapter 2 discusses related work on the topics of software analyses in scientific literature and the modeling and publication of information in research KGs, and introduces the required preliminaries for the ML models used in the scope of this work. Then, a manual analysis of informal and formal software citations is described in Chapter 3, with the goal of determining the requirements for large-scale software analyses and establishing a suited high quality gold standard corpus that enables the development and evaluation of ML models. This analysis also assesses the current representation practices of formal software citation by scientific literature databases, evaluates their corresponding information representation quality, and their usability for systematic software analyses. Next, the automatic information pipeline and the corresponding ML approaches for large-scale software extraction from scientific articles are introduced in Chapter 4, and their application on 3.2 M articles of the PubMed Central Open Access Subset (PMC OA) is described. The pipeline itself is split into three main parts based on the problems of Named Entity Recognition (NER) for software and its metadata, Relation Extraction (RE) between identified information, and Entity Disambiguation (ED) of identified software. Then, Chapter 5 discusses the developed semantic representation of software mentions in the scientific literature and introduces the corresponding KG scheme. Subsequently, Chapter 6 describes how large-scale analyses on software mentions are performed to gather insights on software usage and citation by the scientific community, taking into account changes over time, differences between research domains, article impact, and the state of open source software. Moreover, a case study regarding the relation between software usage and article retraction is included in the scope of the analyses. Each chapter includes a separate discussion of results and limitations, before Chapter 7 reaches a conclusion of the overall work and outlines future research regarding the analyses of software usage in science and its requirements.

2 | Related Work

The content of this section is based on the following publications:

> Frank Krüger and David Schindler. "A Literature Review on Methods for the Extraction of Usage Statements of Software and Data". In: *Computing in Science & Engineering* 22.1 (2020), pp. 26–38. DOI: 10.1109/MCSE.2019.2943847 [150], covers a review of literature concerned with the analyses of software and data mention in scientific publications.

Related work is split up into three major sections to cover the main aspects of this work. First, prior work analyzing software in science is described with respect to its methodology, and its findings are summarized to highlight the current knowledge of software in science, its reliability, and open questions. Next, research KGs are introduced, highlighting their value for data sharing and reproducible research. Then, related work on ED is introduced, split between the fields of Natural Language Processing (NLP) and large-scale databases. Finally, ML preliminaries are introduced, which build the basis of the automatic IE pipeline developed in this work. This section covers the main principles of applied ML models, but also includes general concepts, e.g., unbiased evaluation.

2.1 Analysis of Software in Science

This section provides a detailed overview of prior work dealing with the analyses of software in scientific investigations, classifying it between manual and automatic approaches, as described in Section 1.4. Note that some of the work described here was actually published after some or all methods described in Chapter 4 were developed and published (particularly concerning the work of Istrate et al. [128]). A summary of the related work is also provided in Table A1.

2.1.1 Manual Analyses and Ground-truth Data

The mention of software in scientific publications has been analyzed manually in different investigations. Manual analyses have the advantage of providing highly reliable data and provide valuable insights on software mention practices, but are expensive in terms of annotation effort and are, therefore, limited to a small number of publications. Manual analysis was performed by Howison and Bullard [118], who examine 90 publications in the field of Biology with an annotation overlap of 83%, and perform a detailed examination regarding metadata associated with the software. Nangia and Katz [192] annotated 40 articles from the journal *Nature* in a call for collaboration on the topic of software mentions without considering associated metadata. Duck et al. [77] established BioNerDs, a set of 85 Life Science articles focused on Bioinformatics annotated for informal software mentions with an FScore of F=80% without annotation of associated metadata. Most recently, Du et al. [74] established Softcite, a dataset of 4971 articles, including metadata on developer, version, and URL, annotated with an overall overlap of 76% followed by further expert curation. The articles were sampled from the research disciplines of Life Sciences (2521) and Economics (2450). Softcite was further updated in a recent release to Softcite v2 [24, 120], adding annotations regarding software types and the mention context of software (see Section 3.1.1), and extending the overall number of annotated tosoftware. Annotation specific details on BioNerDs and both versions of Softcite are summarized in Table 3.3 in the scope of Section 3.3, which compares them against the dataset established in this work.

While the available ground-truth datasets are valuable resources for software mentions in science, they do not capture all relevant aspects regarding software mentions in scientific literature. As described above, BioNerDs only considers software names without any further metadata, while Softcite captures some metadata, with information on the mention context being added in Softcite v2. However, both do not consider the unique identification of software across scientific publications, as they do not provide any means to disambiguate software beside the extracted name. Another relevant aspect is the age of the contained articles, due to recent paradigm shifts towards data driven analyses across all disciplines, making software a first class citizen in science and changing the awareness for software citation. This might lead to a concept drift of how software is indicated in scientific literature. Both, BioNerDs and Softcite, contain articles published before 2011, which do not reflect the most recent changes.

2.1.2 Automatic Analyses

Automatic methods can be used to perform large-scale analyses, e.g., regarding software trends over time or across disciplines, but are commonly less reliable than manual approaches as they require automatic IE models. A system based on manually engineered rules incorporating a dictionary of known resource names, for instance, was developed based on BioNerDs to recognize software and database names in scientific literature [77], achieving a strict overlap FScore of F=53% with a Precision of P=49% and Recall of R=57%[1]. The rule-based method was extended by an additional ML post-processing filter [78], improving the performance to F=67%, P=66%, and R=69%. It was implemented by a Random Forest classifier, which performed best out of a selection of ML methods. In a separate work, a Conditional Random Fields (CRF) classifier was also applied on BioNerDs [76], achieving a performance of F=63%, P=54%, and R=74%. Based on the best performing model 714 k articles from PubMed Central (PMC) were further analyzed in the work of Duck et al. [78]. The work of Pan et al. [205] also employs a rule-based method based on contexts and software names learned by iterative bootstrapping. This method has the advantage of requiring minimal supervision for training, but usually achieves low performance, particularly in terms of Recall. Pan et al. [205] report a performance of F=58%, with P=94% and R=42%, and use their approach to analyze a set of 10 k articles from the journal PLoS ONE. Li and Yan [165] restrict the set to software packages of the software framework R and only consider articles that mention or cite R. They then develop a rule-based approach that contains manually engineered features and a dictionary of known R packages gathered from prominent package repositories. Based on this approach they achieved an FScore of F=86%, with Precision of P=84% and Recall of R=87%. The classifier is then applied to ≈14 k articles with the goal of performing a co-mention analyses. While this method achieves high performance, it is restricted to a limited set of target software and does not generalize beyond R packages.

Following my first results, different studies applied deep learning models to the problem of software mention detection. Lopez et al. [173] developed a classifier for software mentions based on the Softcite dataset and compare a Bi-LSTM-CRF approach with a SciBERT-CRF (for details

[1] Details on the calculation of Precision, Recall, and FScore are provided in Section 2.4.2

regarding CRFs, Long Short Term Memory Networks (LSTMs), and BERT see Section 2.4). The results show that the SciBERT-CRF performs better on the problem and achieves an FScore of F=71%, with a Precision of 69% and Recall of 73%. They report that performance could be further improved to F=74% through additional Entity Linking (EL) with Wikidata. The classifier was later applied in the work of Du et al. [75] on the CORD-19 corpus containing more than 280 k articles related to COVID-19 and research on coronaviruses. Further analyses of software mentions was then performed on a manually selected set of recognized software. Most recently, the approach was updated by training a SciBERT-CRF model on Softcite v2 with P=74%, R=89%, and F=81% [24]. This approach was developed to track the production of research software on a national level in the scope of the French Open Science Monitor (BSO) platform. Istrate et al. [128] established a large-scale dataset containing software mentions in $3,8\,M$ articles from PMC and further $16,8\,M$ articles obtained from publishers and bioRxiv, with significant overlap between sets. The software extraction within the dataset was based on SciBERT and trained on Softcite with a reported performance of F=92%, P=90%, and R=93%. Moreover, Istrate et al. [128] employ DBSCAN for ED of software mentions and evaluate the performance by additional data annotation. They report a final performance of F=70%, P=95%, and R=56% for this step.

Overall, early automatic approaches for large-scale analyses of software in science often employed methods based on low performance values, which led to unreliable results. In more recent work, higher recognition performances have been achieved based on the use of state of the art methods in NLP, while the problem of ED (see Section 2.3) is not considered by most prior work and requires further investigation.

2.1.3 Prior Findings

Descriptive statistics are reported for most manual and automatic approaches introduced above. One major aspect is the average number of mentioned software, with previously reported results illustrated in Figure 2.1. The findings of prior work strongly vary between studies, dependent on the underlying scientific domains.

However, the results may also be influenced by the specific data selection, e.g., regarding the age of the analyzed articles, or the methodology employed for analysis. The reported numbers range between only .2 software mentions per article in the Softcite Economics set [74] to 30.8 within the domain of Bioinformatics in BioNerDS [78]. Further, Howison and Bullard [118] report 3.2 software per article in Biology, while Duck et al. [78] report 12.9. For articles sampled from PMC, Duck et al. [78] report 5.5 while Du et al. [74] report 1.4, and Pan et al. [205] report 2.7 in articles published by Public Library of Science (PLoS), with both representing a broad selection of articles with main focus on Life Sciences. The distribution of software itself was found to be skewed towards commonly used general purpose software, which is often statistical software. However, there is some variation in the actual reported numbers. Pan et al. [205] report that 20% of software names account for 80% of mentions, while Duck et al. [78] report that 5% account for 47%. However, both methods do not take ED of software names into account, making the results hard to interpret.

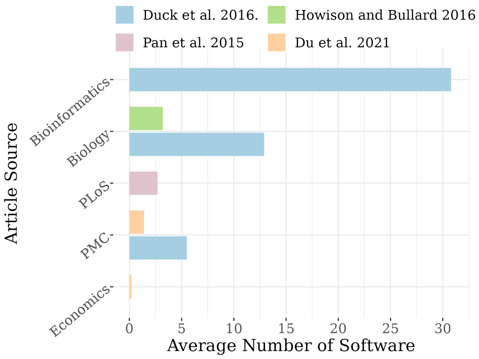


Figure 2.1: Average number of software per article reported across prior work, showing strong disciplinary differences, but also discrepancies between different studies.

Prior work is also concerned with the completeness of software citations, with respect to provided metadata. As described in Section 3.1, complete citation is essential for proper identification and attribution of software. However, this information is often missing in practice. Howison and Bullard [118] found that only 28% of software mentions include a version, 18% the developer, and 5% a URL. They were still able to locate 86% of used software by using web searching, but could only identify the codebase (see Section 1.4) in 5%. Du et al. [74] report that 27% of mentions include version information, 31% developer, and 17% URL. Du et al. [75] found that 46% include versions, 4% URLs, 35% developer, and 18% a formal reference. Further, they report that 96% of the software were identifiable based on the provided information and that the codebase could be identified for 43% of software.

Formal software citation is only investigated by Howison and Bullard [118] and Du et al. [75]. Howison and Bullard [118] found that 44% of software mentions include a formal citation. Further, both investigations distinguish the cited resource behind formal citations and report that respective 84% and 89% of citations refer to articles. Howison and Bullard [118] further identified 5% of citations referring to software manuals and 11% to software directly, while Du et al. [75] report 8% referring to software directly. Moreover, Du et al. [75] analyses the content of formal citations and report that 35% include a version and 78% identify software developers.

Another aspect that received attention in existing literature is the availability of the used software, in terms of free availability and open source publication. Here, Pan et al. [205] found that 64% of the most commonly mentioned software are free for academic use. Howison and Bullard [118] analyzed accessibility, license, and source code availability in combination with citation style and report that commercial software is more likely to be mentioned similar to scientific instruments while open-access software is more likely to receive a formal citation. Du et al. [75] report that 97% of software is accessible online, with 68% being free to use, 47% with source code available, and 43% with open source licenses granting modification permission.

Additionally, there is some work that analyzed specific aspects of software mention and citation, while restricting the scope to specific software or groups of software. Such a restriction can be of interest for software developers and facilitates analyses because keyword search for names, URLs, and associated terms can be applied, potentially followed by manual verification. A semi-automatic approach on a large scale has been established by SwMATH [96, 53], who map software used in Mathematics contained in zbMATH by manual labeling software pre-filtered by a heuristic search. Li et al. [164] analyze mentions of the software *LAMMPS* used in the research discipline of Engineering and found that important information such as the version and the software specific settings are often omitted. An analysis of the software *R* and associated packages by Li et al. [166] revealed inconsistencies resulting from a variety of citation standards that are not well followed by authors. Overall, they could show that *R* receives a high number of formal citations while there is a trend towards more package citations. Pan et al. [204] analyzed the completeness of software usage statements for three specific bibliometric tools and found that a version is provided in 30% of cases, URLs in 24%, and formal citations in a comparatively high number of 76%. They argue that this number, which is higher than average reported numbers, could result from good author citation instructions provided by the tools. Allen et al. [8] analyzed the availability of source code for software mentions in Astrophysics and report that they could locate it for only 58% of all mentions. The work of Tomaszewski [282] analyzed cited references of the software *MATLAB* by using the Web of Science, to investigate in which discipline the software is applied. Lastly, the work of Irrera et al. [127] matched a restricted set of 22 software by name, URL, and Digital Object Identifier (DOI) to analyze the citation position of software in scientific articles.

2.1.4 Different Approaches to Tracking Software in Science

There have also been other approaches with the goal of improving traceability and citability of software in scientific publications. SciCrunch [254] introduces research identifiers for scientific resources including software, which provide a unique identifier for each individual resource. However, the success of this approach depends on the compliance of both software developers and users to allow proper tracking, e.g., by registering new software. Another aspect is the archival of software to ensure reproducibility, which is a cornerstone of science [81]. This is attempted by Software Heritage [69, 3], a nonprofit organization that collects software source code for preservation and long term archival. It can, therefore, also serve as a resource to uniquely identify software that additionally takes versioning into account, which is not considered by SciCrunch. The Software Sustainability Institute [266], on the other hand, sets out with the more general goal of improving research software and making it more sustainable. There are also efforts to facilitate and improve formal software citation in the scientific literature. One initiative is the Citation File Format (CFF)[2], which allows developers of software to easily add human and machine-readable citation information in their repositories, while the biblatex-software package[3] has been released to support authors of scientific publications in citing software, considering the specific citation targets of software, software versions, software modules, and code fragments.

2.1.5 Summary

The results of prior work give some suggestions on how software citation in scientific literature can be systematically analyzed, to enable the applications outlined in Section 1.2. Overall, the previous investigations suggest that software is more commonly mentioned than formally cited, while the mentions themselves often omit essential information. However, reported results often vary, showing strong disciplinary differences in software mentions and rapid changes in citation and mention habits of scientific software, highlighting the difficulty of the problem. Limited gold standard data is available for recent publications reflecting new developments in software usage and citation. Particularly, the most recent trends have only been analyzed based on automatic approaches, which offer less reliability compared to manual methods. Some aspects such as distribution of software within scientific articles are currently investigated without considering software ED and could be biased. Most attention has been payed to software mentions as they are known to be the most common way software is mentioned, with only limited work examining formal citations and no ground-truth annotation available for the problem. There has also not been any previous investigation of how software citations are handled within existing literature databases and of the related data quality. Overall, there is need for further investigation and a high quality ground-truth dataset that covers all intricacies of software mention and citation in the scientific literature, to enable large-scale software analyses as described in Section 1.2.

2.2 Knowledge Graphs

KGs have established themselves as a common way to represent knowledge. The development of large-scale KGs has initially been driven by industry applications such as the Google [259], AirBnB[47], or Amazon [149] KGs. Due to their advantages, they have also been broadly applied for knowledge representation in research, and large-scale KGs were established by the scientific community such as Wikidata [293], DBpedia [161], Freebase [34], or YAGO [114]. The core idea

[2] https://citation-file-format.github.io/, accessed 3 March 2024.

[3] https://www.ctan.org/pkg/biblatex-software, accessed 3 March 2024.

for all KGs is to use graphs for data representation enhanced with a way to explicitly represent knowledge, used for applications that require large-scale integration, managing, and extraction of values from diverse sources of data [200, 115]. Different definitions for the term KG have been used in scientific literature throughout the years. In general, KGs represent knowledge that is amenable to processing in a graph in which entities are related to their attributes and other entities, along with provenance of where that knowledge was obtained [35]. Here, the definition of Hogan et al. [115] is used to provide a definition that is well suited for the context of this work, defining a KG as *"[. . .] a graph of data intended to accumulate and convey knowledge of the real world, whose nodes represent entities of interest and whose edges represent relations between these entities."*.

Graphs provide a concise and intuitive abstraction for data representation, with edges capturing relations between entities, and are applicable to a multitude of different data and domains [12]. Their biggest advantage is that they offer an explicit representation of knowledge, and enable linking between different knowledge bases, allowing to combine existing knowledge. This adds value to the data as it can enable new use cases and allow unexpected re-use of information [31]. This makes KGs particularly well suited for the scientific community, as this format allows a broad integration of different sources, enabling new analyses that would otherwise not be possible, hence, advancing scientific progress. Data analysis itself can also be directly performed on the graph structure by running queries against it, which can be highly abstracted. Moreover, regarding accumulation and linking of knowledge, queries can be federated over graphs to allow broader analyses. The SPARQL query language, standardized by W3C[4], can be used as a tool for this purpose, having the same expressive power as relational algebra [13]. Overall, representing information in KGs allows a semantic representation and enhances findability, accessibility, interoperability, and reproducibility through the underlying graph structure, which contributes to a FAIR publication of established resources [308].

KGs offer advantages, specifically in comparison to relational models, which are commonly used for representation of record type data with known structure and a fixed schema, making their extension difficult and different schemata hard to integrate, especially by automatic approaches. In contrast, KGs allow to easily link different published data and enable broad analyses [115]. The definition of the schema itself could even be postponed allowing data to evolve in a more flexible way, and to capture incomplete information [2]. However, KGs are also well suited to semantically represent information, which is a prerequisite for linking data. Semantic representations for KGs are commonly included by using ontologies, which define the semantics of terms used to label nodes and edges in the graph [115].

The information for KGs can be generated in different ways. The information for general purpose KGs, for instance, is either based on public data sources or added by manual curation as a community effort. Wikidata [293] and DBpedia [161], as prominent examples, have been generated based on automatically processing information obtained from Wikipedia, while Freebase [34] and YAGO [114] are built as community efforts. A common way to obtain knowledge for KGs is by automatic IE. For the research KGs described below, this information is often obtained from publishers of scientific literature or from scientific publications themselves, e.g., by NLP.

2.2.1 Research Knowledge Graphs

In the scope of this work, research KGs are of particular interest because the results of this work will be released in a research KG due to the advantages highlighted above, particularly, in order to link it with existing research KGs. The term research KG is used to refer to KGs that represent information about scholarly literature, the scientific community, and research in general. Different

[4] https://www.w3.org/TR/sparql11-query/, accessed 25 January 2024.

research KGs are available, some covering information about scholarly publications in general, while other focus on specific aspects of scientific publications.

A first effort to represent scholarly data as Linked Open Data was established by the Database Systems and Logic Programming (DBLP) research group at the University of Trier. The corresponding DBLP[5] was first introduced in 2007 and contains information on scientific publications in the domain of Computer Science. According to a recent statement, it covers more than 7 M publications, 7 k conferences and workshops, 2 k journals, 115 k table of contents, 142 k books, 3.4 M authors, and includes links to external resources [4]. The main limitation of the database is that it is limited to a specific scientific domain, while the active quality maintenance of the database is associated with a high curation effort.

The Open Citation Corpus (OCC) [215, 216] was developed to make scholarly citation data openly available to the scientific community. For this purpose, data from PMC OA was selected and processed to create suited representations, based on a newly developed ontology for the description of citations. The main drawback of this approach is that it only considers citations and no other entities such as publication venues, fields of study, or their interrelations. ScholarlyData.org [201] is an effort to represent the academic conferences centered around the Semantic Web community as linked data. It is based on the Semantic Web Dog Food (SWDF) linked dataset of the Semantic Web community that contains information on papers, people, organizations, and events related to conferences. It covers 48 conferences, 235 workshops and 93.5 k individuals. Here, a main limitation is the restriction to a specific community. SPedia [15] is a large, semantically enriched knowledge base of scientific publication data, containing information about over 9 M documents and 24 disciplines in 4 languages. It is based on data gathered from the academic publisher Springer, who has also released a KG representation of its publication data, described below.

The Academic Knowledge Graph (AceKG) [299] is another example of a large research KG, including 62 M papers, 52 M authors, 50 k fields of study, publication venues (22 k journals and 1 k conferences), and 20 k institutes as well as relations between entities. The data itself is based on Acemap[6], with limited information available on data gathering, quality control, and updates. AceKG has the advantage of covering a large set of scholarly publications, but the usability is limited by the unclear data handling. AIDA [11] also represents information of 21 M publications and 8 M patents, with the goal of integrating knowledge on both academia and industry. Its main focus are topics drawn from the Computer Science Ontology. The information is based on information from existing knowledge bases, including the Microsoft Academic Graph (MAG), described below. It enhances the information by performing a topic detection for publications, classifying the type of author affiliation, and industry sectors. Regarding the covered publications, it has not the same size as other large-scale KGs, due to the restriction of the considered scope. Wikidata, a general purpose KG, does also contain bibliometric information, which was integrated based on initiatives such as WikiCite[7]. However, the specialized, large-scale research KGs outlined below, contain a significantly higher number of bibliographic information [84], even so the number of covered publications has increased up to 22.6 M in April 2023 [307].

Groth et al. [97], further, proposed the concept of nanopublication. It is a framework that allows the publication of minimal statements or claims from scientific articles as Resource Description Framework (RDF) triples. While the base concept was proposed, a broad implementation is more challenging because it requires crowd-sourcing or reliable automatic extraction of claims from articles. An application of the concept is performed by Wijkstra et al. [305], to enable the idea of living literature reviews, capturing the results of reviews as nanopublications to allow updates as

[5] https://dblp.org/ [6], accessed 25 January 2024.

[6] https://www.acemap.info/, accessed 26 January 2024.

[7] http://wikicite.org/, accessed 25 January 2024.

regular reviews are static and quickly updated in active research fields. However, this work also presents an initial case study and does not publish a large-scale KG.

The Springer SciGraph aggregates Linked Open Data from the academic publisher Springer Nature and its scholarly partners. It was first published 2017 and is an effort driven by the academic publisher itself. Aside information about publications it covers funders, research projects, coferences, and affiliation, and uses Shapes Constraint Language (SHACL) for data validation [103]. In a 2019 release [265], it contained $8\,M$ articles, $4.5\,M$ chapters, $270\,k$ books, $5\,k$ journals, $7\,M$ persons, $400\,k$ grants $30\,k$ clinical trials, and $300\,k$ patents, while the graph is actively extended with new data. From a linked data perspective this approach is valuable and would allow complete coverage if all scientific publisher would follow this approach. However, overall it covers significantly less publications than large-scale KGs aggregating information over publishers.

Semantic Scholar[8] is a well established scientific literature data and search engine. In the scope of this work, the underlying Semantic Scholar Academic Graph (S2AG) has been established, a KG containing metadata of scientific publications for $205\,M$ articles, $121\,M$ authors, and $2.5\,B$ citation edges [295, 143]. It is based on information from Crossref, PubMed, Unpaywall, Preprint Servers and other sources, but also incorporates information extracted directly from article full-text documents for $60\,M$ articles, based on published PDF files. In that way a field of study classifier is developed, and a disambiguation of papers, authors, and other covered entities is performed. Overall, the covered database is large, and the resource is available through an API, while the data can also be downloaded. However, the semantic structure of the information is not further discussed and it is hard to assess how well it can be linked with other data.

A central resource, that built the basis of different other KGs is the MAG [260]. It represented information on $209\,M$ papers, authors, venues, events, and institutions, and enhanced data with an automatic field of study classification. The development effort was driven by Microsoft, and was, therefore, an industry effort. The gathering of information was largely based on Bing and on data that already existed in-house at Microsoft, with most details of the processing being hidden. The corresponding data was further transformed into an RDF graph in the work of Färber [84]. However, MAG was discontinued in 2022, creating the need for a replacement as it had established itself as a central resource with broad usage.

As a replacement, OpenAlex [220] was developed, with efforts driven by OurResearch, a non-profit organization[9]. In contrast to its predecessor, it is a fully open source project covering scholarly metadata, achieving a clear step towards open science. It is the largest of the outlined KGs, representing five types of scholarly entities and their connections. Works ($219\,M$) describe any kind of scholarly document including datasets; authors ($213\,M$) describe people creating work and are disambiguated with ORCIDs; venues ($124\,k$) are places hosting works such as conferences or journals; institutions ($109\,k$) are represented with ROR IDs; and concepts ($65\,k$) are abstract ideas of what works are about. Information is gathered from different sources, e.g., publication information from Crossref and publishers. The used concepts are based on Wikidata concepts and structured hierarchically. They are automatically generated based on titles and abstracts with the extraction trained on MAG's corpus.

Specific Research KGs

The previously outlined KGs focus mainly on representing scholarly publications themselves, sometimes enhancing information by including aspects such as field of study classification. The research KGs described here focus on representing specific aspects of publications. Semantic Lancet [20], for

[8] https://www.semanticscholar.org/, accessed 25 January 2024.

[9] http://ourresearch.org/, accessed 25 January 2024.

instance, generates semantic abstracts based on deep learning methods, with its data being based on Science Direct and Scopus, for which it creates RDF data to create Linked Open Data.

There is different prior work dealing with the extraction of claims or statements from articles. All are based on existing tool support to extract the corresponding information from articles using ML methods. Buscaldi et al. [42] include $12\,k$ abstracts from the domain of Semantic Web research, based on MAG. They extract and represent the entities of task, method, metric, material, other-scientific-term, and generic and a set of relations between them, resulting in $27\,k$ relations in the corresponding KG. Luan et al. [175] extract the same information to construct a KG including $100\,k$ abstracts from twelve AI conferences based on the Semantic Scholar corpus. AI-KG [66] covers $330\,k$ articles, extracting $820\,k$ research entities, and $1.2\,M$ statements. It covers the entities of method, task, material, metric, and other entities, and relations between them. Compared to the other approaches it covers a broader range of publications, but is also limited to a specific scientific domain. The subsequent work CS-KG [65] further increases the number of represented articles setting the focus on Computer Science articles and the corresponding research concepts they deal with. It includes $6.7\,M$ articles, $10\,M$ entities and 179 different semantic relations between the entities, with article data originating from MAG, where future versions will adapt OpenAlex.

The Open Research Knowledge Graph (ORKG) [16] represents structured and semantic descriptions of research contributions based on article content. It includes $29\,k$ papers, $45\,k$ contributions, 729 research fields, $33\,k$ benchmarks, $6.4\,k$ research problems, and other entities, with metadata identifying articles gathered from Crossref. It allows to perform state of the art comparisons between articles dealing with the same problems. The graph is populated with survey articles presenting an overview of state of the art research, which often provide semi-structured information, e.g., through tables. It is based on manual sourcing and curation, employing a human in the loop approach, where users add papers aided by ML based tool support. Due to the human curation high quality can be achieved, but the scope of the KG is limited. Another drawback is that reviews are static, and new literature on the benchmark is not automatically integrated.

Another highly specific KG represents information on benchmarks used for evaluation of artificial intelligence based on the Intelligence Task Ontology (ITO) [33]. It covers $7.6\,k$ papers with $26\,k$ benchmark results achieved on $3,6\,k$ benchmark datasets, further including aspects such as performance metrics. To establish it, Papers With Code[10] was utilized, a platform where users can add benchmark datasets and report their performance results for the corresponding tasks. The data was then further manually curated.

Summary

Research KGs are an active research field. Large-scale scholarly publication graphs have been established to support the scientific community in bibliographic analyses, often enhanced by automatically extracted information. Specialized KGs further add information to enable deeper analyses or to provide specific services to the scientific community. They are often based on automatic processing of article content, however, there has also been work that is based on manual curation. Currently, these KGs do not include representation of software, while OpenAlex does include datasets. A new KG representing software in scientific publications could be integrated in the existing landscape of research KGs to make use of the advantages KGs offer in terms of linked data, in order to enable broad analyses for the scientific community. Notable, Wikidata does represent both scholarly articles and software, however, it does not establish a link between them.

[10] https://paperswithcode.com/, accessed 25 January 2024.

2.3 Entity Disambiguation (ED)

Another important aspect of this work is ED for scientific software mentions, to identify cases when different publications refer to the same software. The concept of software identities is further described in Section 3.1.3 and the need for ED in large-scale analyses is outlined in Section 3.5. Therefore, this section summarizes existing work dealing with related topics on ED, and their applicability for software mentions. In general, the problem of ED exists in different research fields and is approached with different methods based on their specific requirements. Here, methods from the domain of NLP are discussed as they best match the domain of the given problem considering ED based on mentions extracted from text. Further, methods from database ED are introduced, because the approach of this work is intended for application on large datasets, and the work of the database community focuses on reducing run-time requirements specifically in the scope of big data. Run-time is a central problem in ED because a direct comparison of entity mentions results in a quadratic run-time complexity $\mathcal{O}(n^2)$, where n is the number of extracted mentions. This problem is particularly prevalent when large-scale databases with millions or even billions of entries are considered, motivating the search for methods reducing run-time to a sub-quadratic level. Initially, EL is also briefly discussed, which offers run-time benefits but requires a suited knowledge base to link against.

2.3.1 Disambiguation by Linking

EL describes the process of linking entities against an existing knowledge base. For this purpose, different methods have been proposed in the literature, e.g., manually engineered features and SVMs [72] or more complex methods such as deep learning approaches [294]. If an ED problem can be solved by EL the time complexity can be reduced. Assuming a direct comparison between all mentions m and knowledge base entries e it would lead to a complexity of $\mathcal{O}(n_m n_e)$, which is a run-time improvement assuming that $n_m \gg n_e$. In general, an external source to link against is not available for every problem, or it might lack the required coverage. Specifically for software mentions, EL has been applied in the work of Lopez et al. [173], who report that EL improved entity recognition as a post-processing step, and enriches information, however, the achieved linking coverage is not provided. It has also been applied in the work of Istrate et al. [128], who use it to link software to external sources after performing ED. Based on curator feedback, they report that 54% of extracted links are correct, while correctness was unclear for a large amount of links (40%). In the scope of this work, it is argued that the coverage of software in available data sources is insufficient to perform EL for large-scale analyses and, instead, ED of software in scientific articles is required (see Section 3.5). Therefore, EL is not further considered here and other ED methods are the primary focus. However, EL can still be useful as an additional step after ED to gather additional information about software [173, 128], and could be used in future applications when more comprehensive knowledge bases of software in science are available.

2.3.2 Entity Disambiguation for Natural Language Processing

Different work has investigated ED based on mentions of entities extracted from texts, often referred to as Cross-Document Coreference Resolution (CDCR). Cattan et al. [44] approach the problem of event CDCR based on the ECB+ dataset [59], containing 982 documents of news articles, with the largest category of entities being actions with ≈15 k mentions. Their method works from an initial prediction of mention spans, which are encoded based on document encodings by using Large-Scale Language Models (LLMs) (see Section 2.4.3). Spans are then pairwise scored by an ML based scorer to establish a distance measure between them, which is initially trained on scoring mention pairs to binary predict whether they refer to the same entity. During training the ratio of positive and negative pairs is selected in a 1:20 ratio. Based on the established distances, agglomerative clustering is performed.

Zeng et al. [318] also work on the ECB+ dataset to solve the problem of event CDCR. They perform an initial document clustering to narrow the search scope, which decreases the number of required comparisons and was reported to mitigate errors. Their ED method is also based on pair-wise comparisons by an ML model, with the model being trained on the problem by oversampling positive and downsampling negative sample pairs during training. They combine two different similarity vectors with a deep learning approach. The first vector is based on BERT (see Section 2.4.3), calculating similarity based on the sentence containing the event mention, while the second vector is based on Semantic Role Labeling. Furthermore, the work of Caciularu et al. [43] establishes a pre-trained language model specifically for cross-document contexts. In the scope of their work, they test it on different benchmarks including the ECB+ corpus, on which they report good performance using their model with pair-wise comparisons. In general, prior work on the topic commonly followed the approach of using pair-wise scoring followed by clustering [58, 315, 52, 23].

While the prior work described above focused on news articles, Ravenscroft et al. [224] perform CDCR across scientific articles and newspapers for entities mentioned in both. They create a new dataset containing automatically gathered pairs of scientific publications and corresponding news articles about them. The corpus has approximately the size of ECB+. On this basis they compare the performance of different baseline approaches, such as using BERT to encode mentions and the described approach of Cattan et al. [44]. Cattan et al. [45] fruther develop a CDCR approach for technical concepts in scientific publications, particularly focused on Computer Science. Same as other work, they employ existing baseline models and LLMs on the problem and base their approach on pair-wise scoring, with the final step performed by agglomerative clustering.

In general, the outlined approaches are closely related to the problem of software ED even so they consider different document types and entities. Especially, the approach of Cattan et al. [45] is similar to the problem considered in this work regarding the underlying texts and the task itself. However, the general drawback of all outlined methods is that they only consider small gold standard datasets. Therefore, they do not take run-time complexity into account but optimize the recognition performance. Particularly, all outlined approaches have run-time requirements of $\mathcal{O}(n^2)$, and often employ complex ML methods for each comparison. Only the approach of Zeng et al. [318] includes document clustering to reduce the number of required comparisons, however, since there is general purpose software applied over all scientific domains this approach cannot be direclty applied to software ED. Only the work Istrate et al. [128], published subsequently to the methods introduced in this work, considers the ED of software mentions. For that purpose, it constructs a similarity matrix between software mentions which is clustered by applying DBSCAN [82]. The matrix itself is constructed based on three separate components, a keyword-based synonym generation based on the Python Package Index (PyPi), The Comprehensive R Archive Network (CRAN), and Bioconductor, the use of SciCrunch synonyms (see Section 2.1.4), and Jaro-Winkler string similarity. They report an FScore of F=70.4% with a Precision of P=95.4% and Recall of R=55.8% for ED, with the evaluation based on curator labeling of 5884 synonym pairs in the disambiguated data.

2.3.3 Entity Disambiguation for Large-Scale Databases

ED is actively researched in the context of databases, because several problems regarding big data and data linking are related to it. One problem that is particularly close to the context considered in this work, is referred to as dirty Entity Resolution (ER), and considers one collection of data in which duplicate mentions can exists. In the following, a general overview of methods employed in this context is provided based on the review of Christophides et al. [54]. Note that below the term mention is used to refer to database entries to use consistent terminology with the descriptions above.

ER is commonly approached in four different steps. Initially blocking is performed to reduce the number of candidates to be compared for ER. Blocking creates sub-blocks of mentions, and only within these blocks the actual resolution is performed, while mentions that are not in the same block are not compared in subsequent steps. In general, the goal of blocking is to reduce the number of required comparisons as far as possible while maintaining high Recall. This initial run-time optimization enables the application of subsequent ER methods, and has the general goal of achieving a sub-quadratic run-time. Therefore, it is highly relevant for the entire pipeline, and different work is solely focused on improving blocking itself, outlined below. Often, block cleaning is performed as a second step to prune larger blocks that would require a high number of comparisons. The third step is matching, to identify whether mentions refer to the same entity. It is usually based on the calculation of similarity scores between pairs of mentions within the generated blocks, and commonly has a quadratic run-time complexity, similar as in the domain of NLP. The final step is clustering, where different approaches have been employed in existing literature [54].

Recently ML models have been adapted to both blocking and matching. Here, the main focus are blocking methods, as they differ from the methods employed in NLP ED. DeepER [79], for instance, establishes an end-to-end approach, solving the blocking step by calculating tuple embeddings, based on word embeddings processed with a LSTM (see Section 2.4). Subsequently it uses Locality Sensitive Hashing (LSH) on the generated representations, which is used to approximate high-dimensional Nearest Neighbor Search [125]. In contrast to regular hashing, trying to prevent collisions, LSH strives to generate collisions for similar mentions. For matching, cosine similarities are calculated between pairs of samples and clustering is performed up to an optimized threshold. Another approach is Autoblock [321], which assumes semi-structured features such as attributes based on mention descriptions in relational databases. Token-based embeddings of the attributes are then combined by a deep learning attention mechanism to calculate tuple signatures, which are used for blocking with LSH.

The DeepBlocker framework [280] follows the same general approach as the work outlined above. It uses word embeddings to generate tuple embeddings for specific mentions. However, it further sets up a framework to compare different ML methods for generating tuple representation to compare how well they perform. The work found Autoencoders [111] to be well suited for blocking, as well as a hybrid approach between Autoencoders and cross-tuple training, which aims to optimize tuple embeddings by defining a training objective over pairs of tuples integrating regular Autoencoder training. Same as above, LSH is used to achieve efficient blocking based on generated embeddings, and a combined approach of LSH and similarity based pairing, e.g., by cosine similarity, is proposed. Most recently, the work of Zeakis et al. [316] compares different available word embeddings and language models for blocking and matching. They use an otherwise equal setup to assess which pre-training approach is best suited for the problem, with mentions represented by the concatenation of their attributes. They report that SentenceBERT [229] performs overall best for blocking across the ten considered evaluation settings, and argue that it is due to the sentence based training and the wider training corpora for the model. Particularly, compared to other BERT based language models generating contextualized representations, which were found to perform worse on the problem.

The main difference between ED performed in the scope of large-scale databases and in NLP is the step of blocking, introduced specifically to reduce run-time, while the methods employed for matching and clustering are similar and focus on pair-wise distance calculation for which ML methods have been adapted in both fields. A drawback of the used blocking methods is that it is not clear whether they are applicable to software in scientific publications. Blocking is generally based on embedding methods and requires the mentions of interest to be well represented to achieve high performance. Highly specialized scientific software is rarely mentioned and only appears in few contexts across all existing literature, making its representation in word embeddings and language models challenging. Further, it is also not clear how well embeddings integrating context information would work for software, as there is a broad range of software that is mentioned in similar contexts in scientific literature. In general, extensive investigation is required to determine how blocking methods could be adapted for software mention ED.

Another main difference is that database methods are employed on semi-structured data corresponding to record entries. However, they are often handled by concatenating record information as a string before employing NLP methods, while only some work explicitly considered semi-structured data. In general, ED input information on software mentions is either available as full sentences, or as semi-structured data, assuming that metadata is extracted together with the software mention. This makes both NLP and database methods applicable in the given context.

2.4 Machine Learning Preliminaries

Prior work has shown that large-scale analyses of software in science is a challenging problem and requires the use of ML approaches (see Section 2.1.2). Therefore, this section describes state of the art ML methods for the problems related to software analyses, which are later tested and evaluated in the scope of this work. Extensive background on essential, complex models is provided where necessary, while some base concepts regarding ML and Neural Network (NN) optimization are assumed and pointers to further literature are provided. However, some base concepts are also explicitly introduced, e.g., regarding measures and methods for unbiased evaluation.

Supervised learning is applied in the scope of this work, which has the goal of automatically learning a mapping between observations and corresponding outcome. Formally, the goal is to learn a mapping function f from the observations $\mathbf{X}$ to predictions $\mathbf{y}$ with θ parameterizing the ML model: $\mathbf{y}=f(\mathbf{X}|\theta)$. The statistical method of the model provides the framework for the learning process and has significant influence on how well the problem will be represented, with the goal of learning θ to best approximate the real function between observation and outcome.

2.4.1 Data

ML models are directly dependent on the data that is used to train them, with the quality of the training data influencing the quality of the model. Even within gold standard data established by manual annotation, errors are present. This was demonstrated by Northcutt et al. [199] who estimate an average error of 3.3% across ten commonly used ML benchmarks. It should also be considered that a problem might not be solvable based on given data, because the provided information might not be suited to discriminate the posed target variable, for instance, due to ambiguities in the data. In the scope of natural language data, they can make it difficult to determine what an author refers to in a restricted context without external knowledge or topic background.

This work considers natural language data, which makes it necessary to introduce some base concepts of how language is represented for ML methods and the related terms that will be used throughout this work. *Document* refers to the complete text of one semantic document, in the given context this corresponds to one scientific publication. *Sentence* refers to a semantic sentence in a document, and *Token* refers to a continuous string without spaces within a sentence, including all punctuation. Lastly, *Character* refers to a single character, either letter, number, or special character. Since ML models require numerical inputs, the semantic units described above are transformed into numerical representations. Documents are commonly represented as sequences of sentences, where each sentence is a sequence of tokens, and each token a sequence of characters. Numerical representations are often generated on the basis of tokens, characters, or sub-strings of tokens.

To test ML models the available data is usually split between training, development, and test sets. The training set is intended for learning the mapping function from input features to targets, while the development set is used to assess the performance specifically for model selection and hyper-parameter optimization. The test set is reserved and only used to evaluate the performance of the final selected model to achieve an unbiased evaluation of its generalization capability. The term generalization describes the ability of the model to correctly predict unknown samples. There are two main problems that can occur in this training process, assuming the available data is suited to represent the given problem. *Overfitting* occurs when the model is fitted to close to the training data and has learned to represent noise in the training data, while *underfitting* occurs when the model is not able to represent the relationships between features and targets [137]. The reason for overfitting is usually that the model is too complex, i.e., the model has a high capacity for feature representation and is able to memorize the data instead of learning suited feature representations. Underfitting, on the other hand, is caused when the model is lacking capacity for feature representation. In both cases the model does not generalize well and the errors can be identified examining the models performance between sets. When the model is overfitting the training error is low and the development error is higher, while both the training error and the development error are high when underfitting is present.

2.4.2 Evaluation

Evaluation is a central aspect in ML and concerns both data and models. Evaluation metrics are usually defined based on the differences in model prediction $\hat{y}{=}f(\mathbf{x}|\theta)$ and known target value y. The same concepts can also be applied to evaluate the quality of data annotation by comparing the annotation labels y_1 and y_2 assigned by two human annotators. Here, all given problems are classification tasks, which is why only labeled data on a nominal scale is considered. In the following, the main performance measures used throughout this work are introduced: Cohen's κ, which is primarily used to assess Inter Annotator Agreement (IAA) for data annotation, and FScore, Precision, and Recall, which are used to assess performance for classification problems.

Cohen's κ

Cohen's κ assumes that annotators have an underlying distribution of annotations, as they annotate some classes more often than others [14]. The κ statistic then accounts for chance agreement between the annotation leading to a suited estimation for categorical annotations tasks [55]. It is defined as:

$$\kappa = \frac{p_0 - p_e}{1 - p_e}, \tag{2.1}$$

where p_0 is the measured agreement and p_e the probability of chance agreement calculated based on the knowledge of how often every class was annotated by each annotator [14], defined as:

$$p_e = \frac{1}{N^2} \sum_{c \in C} n_{a_1 c} n_{a_2 c}, \tag{2.2}$$

where N is the cardinality, C the set of annotated classes, and $n_{a_i c}$ is the number of annotations for class c made by annotator a_i.

FScore

The FScore can be applied to both binary and multi-class classification problems. First, the evaluation of a binary classification task is considered, where only two outcomes exist: 1 (*true*) and 0 (*false*). This results in four evaluation cases considering the prediction of a classifier p against the target label t: *true positive* (tp) $t{=}p{=}1$; *false positive* (fp) $p{=}1 \wedge t{=}0$; *false negative* (fn) $p{=}0 \wedge t{=}1$; and *true negative* (tn) $t{=}p{=}0$. Based on these values, the *Recall* describes which amount of overall *true* labels was identified by the classifier and is defined as:

$$R = \frac{tp}{tp + fn}, \tag{2.3}$$

while the *Precision* describes the amount of assigned *true* labels that is actually true, defined as:

$$P = \frac{tp}{tp + fp}. \tag{2.4}$$

The *FScore* is calculated as the harmonic mean between *Recall* and *Precision* to assess the overall performance, defined as:

$$F = \frac{2PR}{P + R} = \frac{2tp}{2tp + fp + fn}. \tag{2.5}$$

Theoretically, a weighting factor different from 1 can be added to the FScore to implement a trade-off between *Recall* and *Precision* if one is more important in a specific application, which is not considered here. With respect to multi-class problems, a separate binary score can be calculated for each class projecting the problem back down to the binary case. These separate scores allow to assess for which classes the classifier performs well, and can be summarized to an overall score.

2.4.3 Models

In ML, there is often a distinction made between classical approaches based on manual feature engineering followed by statistical predictors and deep learning approaches—also known as Neural Networks (NNs)—which aim to learn suited feature representations. Which models are best suited for a problem is dependent on the specific application and its requirements regarding predictive power and explainability, but also on the available data. Deep learning models commonly require a large amount of supervised data due to the high amount of parameters to be trained, while classical models can benefit from prior defined features. This section introduces the ML models that were used in this work, while the choice for these models is motivated in Chapter 4.

All ML models are of statistical nature, and can be adjusted to different problems by changing hyper-parameters, which control the capacity of the model, its structure, and the statistical properties of its learning process. The parameters are unique to each method and their optimization is a central part of finding a suited model and adapting it to a given task. Therefore, this section explains the main hyper-parameters for the considered models, while the considered optimization process is described in Chapter 4. In general, the training and optimization of state of the art NNs is a resource intensive process. Beyond the basic hardware requirements it consumes a considerable amount of energy, and produces both financial and environmental cost [273]. The task is particularly challenging when a high number of hyper-paramters has to be considered, and there are several approaches that try to reduce the run-time requirements of model selection, such as using Bayesian hyper-parameter search [309], population based learning [129], or Hyperband [167].

Random Forest

Random Forest [38] is a classical ML approach that it is broadly applied because it achieves high predictive performance on a broad range of problems, while offering good explainability of the decision process. This allows it to be employed in sensitive applications where it is important to control the behavior of the application. As described in Section 2.1.2 it has also proved successful for software mention detection in comparison to other ML methods. The concept of the Random Forest is based on Decision Trees, a tree based classifier with further background provided in [112, 137, 168]. Hyper-parameters relevant in this context are the purity criterion used to generate the Decision Tree, for which either the *entropy* and the *Gini index* are used [137], and methods to regularize the tree by restrictring its growth, e.g., by setting a maximum depth or setting a minimal number of samples for which leaf nodes are further split.

The Random Forest uses an ensemble of Decision Trees to reduced overfitting [38], with the goal of generating several trees with a low correlation between their classifications. Two methods are usually combined to achieve this goal. The first is bootstrapping (also referred to as bagging) of samples from the training data [37], where a separate sample set is created for training each Decision Tree by sampling with replacement. The second restricts the tree creation process by only considering a randomly selected subset of all available features at each step. With respect to hyper-parameters, both aspects and the number of used Decision Trees need to be considered. The classification is then performed by a majority vote between all generated trees.

Conditional Random Fields

Conditional Random Fields (CRF) [152] are useful when predicting multiple outputs with interdependencies. Here, their specific application on sequential data is considered where the output labels are dependent on the neighboring output labels, which is often the case in NLP applications because tagging restrictions are imposed on output data. A broader introduction beyond the application to sequence data can be found in Sutton, McCallum, et al. [276].

The application of CRFs in a sequence tagging context, particularly in combination with a LSTM (introduced below), was described in multiple prior studies [155, 177, 50]. All apply CRFs for joint output modeling in a similar fashion, which is summarized in the following. The input sequence is defined as $\mathbf{X}{=}(\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n})$, with a corresponding output sequence given by $\mathbf{y}{=}(y_1, y_2, \ldots, y_n)$, with a classification between k distinct output tags at each step y_i. It is assumed that a score matrix $\mathbf{P}$ of size $n{\times}k$ was calculated in a previous step of a larger ML model, where entry $P_{i,j}$ is the score of jth tag of the ith step in the sequence. The score of tag sequence $\mathbf{y}$ for features $\mathbf{X}$ is then defined as:

$$s(\mathbf{X}, \mathbf{y}) = \sum_{i=0}^{n} A_{y_i, y_{i+1}} + \sum_{i=0}^{n} P_{i, y_i}, \tag{2.6}$$

with $\mathbf{A}$ defined as the transition matrix between tags, with $A_{i,j}$ denoting the transition score between tag i and tag j. At this point, the literature differs in implementation details between modeling a start probability [50], or including explicit start and end tags [155], which also influences the size of matrix $\mathbf{A}$, but does not change the overall CRF behavior.

The model is then trained by maximizing the log-likelihood of the correct tag sequence based on a softmax formulation of the problem, defined as:

$$\log(p(\mathbf{y}|\mathbf{X})) = s(\mathbf{X}, \mathbf{y}) - \log\left(\sum_{\widetilde{\mathbf{y}} \in \mathbf{Y_X}} \exp^{s(\mathbf{X}, \widetilde{\mathbf{y}})}\right), \tag{2.7}$$

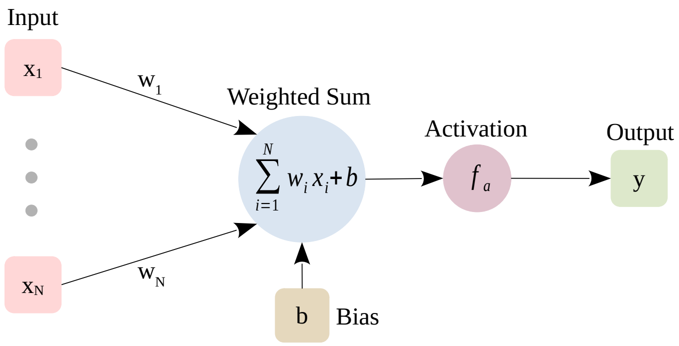


Figure 2.2: Illustration of the processing of a single NN neuron.

with $\mathbf{Y_X}$ defined as the set of possible tag sequences for input sequence $\mathbf{X}$. The training itself, corresponding to weight updates for the CRF, is then implemented by Stochastic Gradient Descent (SGD), together with the entire model. During prediction, the most likely output sequence can be calculated by applying the Viterbi algorithm. When the CRFs are applied as described here and their training is integrated together with a NN for feature generation there are no specific hyper-parameters that need to be tuned aside the ones determining the overall training process, which are further discussed in Section 2.4.3.

Neural Networks

Neural Networks (NNs) receive their name from their original modeling based on human neurons, while most practical implementations do not bear much resemblance. The base idea is built on stacked neurons that perform weighted sums of their inputs with added non-linearity. The concept is illustrated in Figure 2.2 and defined by:

$$z_j = \sum_{i \in P} w_{ji} x_i + b \,, \quad y_j = f(z_j), \tag{2.8}$$

where j indicates a specific neuron in the network, P is the set of all predecessors to neuron j, w_{ji} the weight of neuron j for input i, x_i the input value from predecessor i, b an added bias term, and f a non-linear activation function applied on the neurons output. This non-linearity is essential because otherwise the network would come down to simple linear transformation of a given input. Networks are usually defined in terms of layers, where each neuron in a layer behaves in the same way. Layers are then stacked to form deep networks where information is typically only passed between adjacent layers. The network layers are often specifically designed to represent features of specific data structures, for instance, recurrent networks are usually used to represent sequence data and convolutional networks for image data. This means humans still define the model architecture, but they do not actively engineer features, as in classical ML approaches. Theoretically, it was shown that two stacked layers with a sufficient number of neurons are sufficient to model every function and can be considered as universal approximators [117]. In practice, however, deeper network designs with a higher number of layers have been found to perform better, and achieve state of the art performance in many ML benchmark, since the necessary hardware to train them has become available [158].

Here, some basic principles regarding deep learning are introduced, while a deeper background is available in Goodfellow et al. [95]. Models are commonly trained by SGD based on the principle of backpropagation, with the process begin implemented in mini-batches of samples to reduce variance in weight updates while allowing an efficient implementation [231]. For this purpose, the output of a NN model is compared against the target output based on a defined cost function, e.g., the categorical cross-entropy loss for classification problems, and the resulting loss is propagated back through the network to update the weights. Often the process is also based on more complex methods than plain SGD, which additionally modify the strength of weight updates, such as the widely applied Adam optimizer [142]. Overall, it should be noted that the loss functions become non-convex due to the non-linearity in the NNs, and that there is no global convergence in NN training. Instead, the iterative weight updates will most likely reach a local minimum [95].

In practice, there are several additional aspects influencing the training and performance of a NN, i.e., how well it solves the problem it is trained on according to the defined evaluation metric. For one, the choice of activation function, referring to the implementation of the non-linear functions within the network, does influence performance. It is also necessary, to appropriately initialize the networks weights, typically with small, random values. There is also a number of methods with the goal to reduce overfitting by regularization of networks, with well known approaches being dropouts and batch normalization[126]. Other typical issues are vanishing or exploding gradients in NNs, with a simple approach for exploding gradients being gradient clipping, e.g., by rescaling them when they are beyond a given threshold [206]. Another concept that often finds application in practice are residual networks [107], which introduce short-cut connections that bypass layers, with the benefit of better gradient propagation.

Long Short Term Memory Networks

Long Short Term Memory Networks (LSTMs) [113] are a form of recurrent NNs [235]. This means they share parameters and weights across time steps and can process sequences of arbitrary length. This means sequences of arbitrary length are mapped to the same hidden representation, which will lead to a selective keeping of past information at every step in the sequence [95]. The major issue in recurrent networks is to learn long-term dependencies due to vanishing gradients, because short term dependencies result in higher gradients and mask the learning of long term dependencies. This makes learning them particularly difficult and time consuming even at short sequences of only 10 to 20 elements [30]. This problem is mitigated by the LSTM [113, 206] that introduces a gating mechanism for the memory cell. Intuitively, the gates can create instances in which certain elements from the memory cell are not updated and instead the value in the memory is passed on over multiple steps, which allows long term gradient flow.

The gate units in LSTMs define how an update of the internal network cell is performed at a time step t based on the input $\mathbf{x}_t$ and the previous network cell state $\mathbf{h}_{t-1}$. The principle is illustrated in Figure 2.3 and defined by Ma and Hovy [177] as:

$$\mathbf{i}_t = \sigma\left(\mathbf{W}_i\mathbf{h}_{t-1} + \mathbf{U}_i\mathbf{x}_t + \mathbf{b}_i\right), \tag{2.9}$$

$$\mathbf{f}_t = \sigma\left(\mathbf{W}_f\mathbf{h}_{t-1} + \mathbf{U}_f\mathbf{x}_t + \mathbf{b}_f\right), \tag{2.10}$$

$$\mathbf{o}_t = \sigma\left(\mathbf{W}_o\mathbf{h}_{t-1} + \mathbf{U}_o\mathbf{x}_t + \mathbf{b}_o\right), \tag{2.11}$$

$$\hat{\mathbf{c}}_t = \tanh\left(\mathbf{W}_c\mathbf{h}_{t-1} + \mathbf{U}_c\mathbf{x}_t + \mathbf{b}_c\right), \tag{2.12}$$

$$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \hat{\mathbf{c}}_t, \tag{2.13}$$

$$\mathbf{h}_t = \mathbf{o}_t \circ \tanh\left(\mathbf{c}_t\right). \tag{2.14}$$

The gating mechanism is controlled by the *input* gate $\mathbf{i}_t$ (Equation 2.9), *forget* gate $\mathbf{f}_t$ (Equation 2.10), and *output* gate $\mathbf{o}_t$ (Equation 2.11), which are all based on the same mechanism. Both,

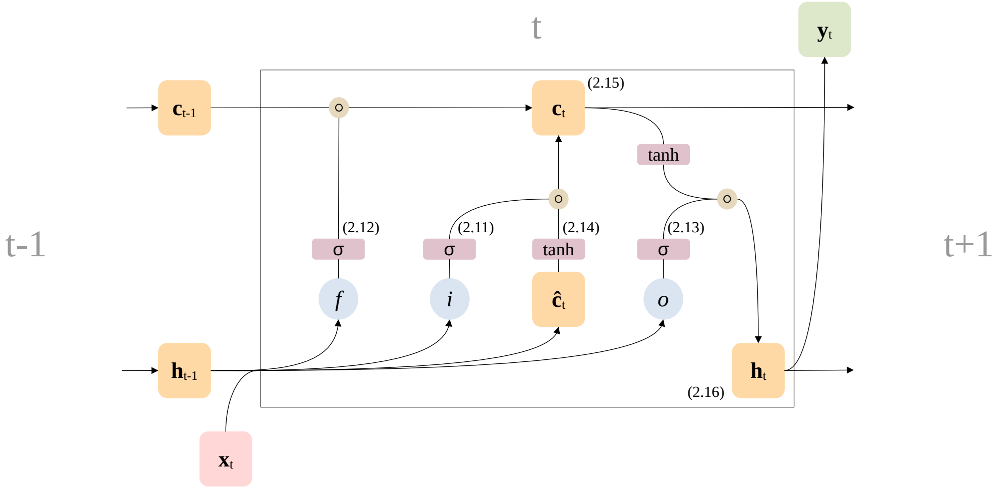


Figure 2.3: Illustration of an LSTM cell at a specific time step t, taking reference to Equations 2.9–2.14.

the previous cell state $\mathbf{h}_{t-1}$ and the input $\mathbf{x}_t$ are multiplied with respective weight matrices $\mathbf{W}_{i/f/o}$ and $\mathbf{U}_{i/f/o}$[11], with additional bias terms $\mathbf{b}_{i/f/o}$. The outputs for all gates are then calculated by applying a sigmoid activation function σ, scaling the outputs between zero and one. This is the main aspect of the gating mechanism, as it defines the strength with which new information is added to the internal memory, the strength with which information from the internal memory is forgotten, and what information is returned at the current time step. The update to the internal memory state is then calculated by first creating a new candidate memory state $\widetilde{\mathbf{c}}_t$ (Equation 2.12), and then combining previous and candidate memory scaled based on forget gate $\mathbf{f}_t$ and input gate $\mathbf{i}_t$ in Equation 2.13, where $\circ$ denotes a point-wise multiplication. Intuitively, a forget gate value of 0 completely removes information from the previous memory cell, while a value of 1 retains it. Similarly, an input gate value of 0 completely ignores new information, while a value of 1 includes it. The new hidden state is then modified by the last gating mechanism $\mathbf{o}_t$ in the same manner, to create the new hidden state $\mathbf{h}_t$ (Equation 2.14).

In terms of hyper-parameters there are several aspects to consider. Aside parameters for modifying the training process of NNs in general, the main aspect of LSTMs is the size of their hidden state, which also determines the size of all weight matrices contained across all gates. Additionally, it can be decided whether to use a uni-directional or bi-directional model [253], where two separate memory cells are used to process a sequence, one forward and one backward, with the outputs combined at each step so information of the entire sequence is available at each time step. Moreover, it can be considered to stack multiple LSTMs on top of each other, for a deeper network structure in terms of layers.

Convolutional Neural Networks

The idea of Convolutional Neural Networks (CNNs) was originally based on the work of LeCun et al. [159], and implements the mathematical concept of a cross-correlation. Details on CNNs can be found in Goodfellow et al. [95] and are not discussed here, because CNNs are not applied in this work but only mentioned as an alternative implementation of a sub-module of a network architecture. Coarsely, a single neuron implements a n-dimensional convolution kernel of a fixed size k that is applied on a n-dimensional input, kernel itself learned as the neurons parameter weights.

[11] Matrix multiplication is used as an effective notation for the application of multiple neurons, i.e., a layer.

This enables the model to extract shift invariant features from its input, which can implement complex filter operations based on parallel applied neurons in one layer, and multiple stacked layers on top of each other.

Transformers and Attention Models

Attention based models have been achieving state of the art performance in a broad range of NLP problems, particularly in combination with pre-training approaches (see Section 2.4.3) generating Large-Scale Language Models (LLMs), such as BERT (Bidirectional Encoder Representations from Transformers) [67]. A major benefit of this model class is that their training can be parallelized more efficiently in comparison to recurrent network models. The base principle of neural attention has first been proposed in combination with other network structures in the context of machine translation [21, 176]. The concept has been further adapted by Vaswani et al. [291], who base a new model structure mainly on a specific attention mechanism called self-attention, a mechanism relating different positions of a single sequence to compute a representation of the sequence. The initially proposed model was also intended for machine translation and, therefore, consisted of an encoder-decoder architecture. Subsequently, it was found that particularly the encoder part of the model can be used for training LLMs, described below. Therefore, only the encoder part of the network is introduced in the following, while the specific training procedures for LLMs are outlined in Section 2.4.3.

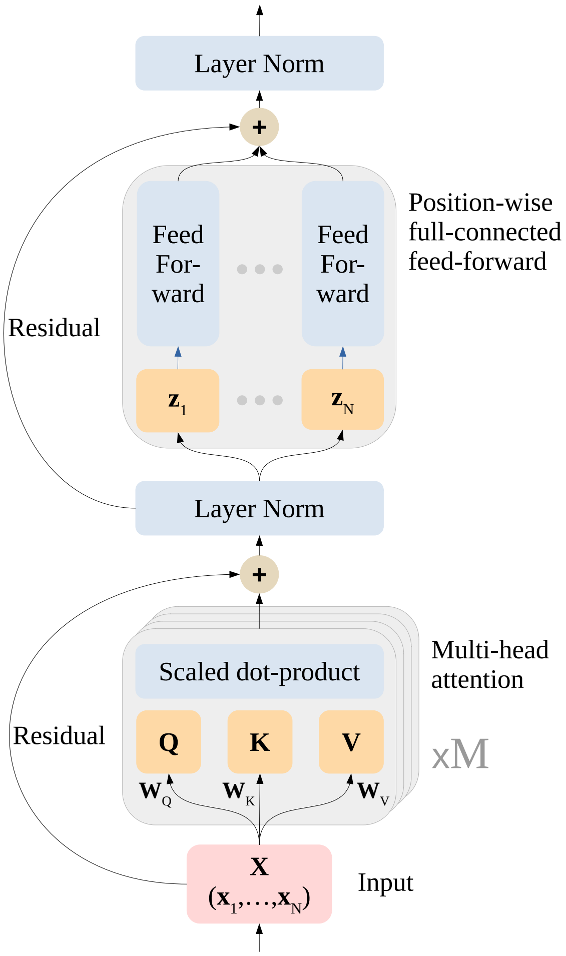


Figure 2.4: Illustration of a transformer encoder block consisting of a multi-head attention mechanism based on scaled dot-product attention and a point-wise feed forward network. Each block is separately covered by a residual short-cut connection and followed by a layer normalization. Query $\mathbf{Q}$, key $\mathbf{K}$, and value $\mathbf{V}$ matrices are generated by multiplication with trainable weight matrices $\mathbf{W}_Q$, $\mathbf{W}_K$, and $\mathbf{W}_V$, respectively.

The transformer encoder as introduced by Vaswani et al. [291] consists of six identical layer modules that are stacked on top of each other, with the structure of one module illustrated in Figure 2.4. Each layer consists of a multi-head attention layer and a point-wise fully-connected feed-forward network. The multi-head attention is based on parallelly applied scaled dot-product attention, defined as follows:

$$f_a(\mathbf{Q},\mathbf{K},\mathbf{V}) = softmax\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}, \qquad (2.15)$$

with the matrices $\mathbf{Q}$ and $\mathbf{K}$ of size $N{\times}d_k$, and $\mathbf{V}$ of size $N{\times}d_v$, defined as query, key, and value information. The query, key, and value are calculated at every step $t{\in}N$ of the input based on a trained weight matrix, which can also be interpreted as a simple fully-connected feed forward network. The main aspect of the attention lies in how a new representation is calculated based on $\mathbf{Q}$ and $\mathbf{K}$, and $\mathbf{V}$ at each time step. For further explanation, the process is illustrated for a specific time step t defining $\mathbf{q}_t$=$\mathbf{Q}_{t,*}$, $\mathbf{k}_t$=$\mathbf{K}_{t,*}$, and $\mathbf{v}_t$=$\mathbf{V}_{t,*}$. Initially, scores $s_1, \ldots, s_N$ are generated by calculating the dot product between q_t and all keys $k_1, \ldots, k_n$, normalized by the key length $\sqrt{d_k}$, with the normalization argued to control the gradient of the subsequent softmax function. The scores are then transformed to a probability distribution based on a softmax formulation defined as:

$$p_i = softmax(s_i) = \frac{\exp(s_i)}{\sum_{j=1}^{n} \exp(s_j)}. \tag{2.16}$$

Note that the softmax operation in Equation 2.15 performs the same operation row-wise over the provided matrix input, yielding another matrix as output. Finally, the generated values are scaled by the calculated probability scores, and are summed to generate the output at point t:

$$o_t = \sum_{i=1}^{n} p_i \mathbf{v_i}. \tag{2.17}$$

In the multi-head attention described by Vaswani et al. [291] this scaled dot-product attention is then applied eight times in parallel, with results being concatenated and transformed by another trained weight matrix.

The attention output is then passed to a point-wise fully-connected feed-forward network, which means that the same network is applied at each time step in the sequence. Moreover, residual connections [107] (see Section 2.4.3) are applied around the attention layer and the fully-connected layer, as indicated in Figure 2.4. Finally, layer normalization [17] is applied after each layer, which in difference to batch normalization (see Section 2.4.3), normalizes an input based on a single training sample based on mean and variance to stabilize the training process. Furthermore, a cyclical positional encoding is added to the input data representation to integrate sequence information in the model. In general, there are several hyper-parameters that could be tuned regarding this model architecture. However, the model is usually applied pre-trained as a LLM, as outlined below. Therefore, the architecture is pre-defined and only the training process can be modified as in regular NN training. BERT [67], for instance, uses an architecture with 12 layers, and 12 attention heads, and increases the feature dimension of query, key, and value vectors of the original transformer.

Transfer Learning

A central problem for deep learning is data sparsity since NNs require large amounts of data to perform well. Therefore, there are approaches aiming to reduce the amount of required task specific training data [95]. Transfer learning can in this context be considered regarding the domain D of the underlying data and the task T to solve. In general, the goal of transfer learning is to generate representations for a target task T_T on the target domain D_T based on a source task T_S and a source domain D_S, where $D_T \neq D_S$ and $T_T \neq T_S$. More specifically, the term transfer learning is often used to describe the application of a model trained on one task to another ($T_T \neq T_S \wedge D_T = D_S$), while domain adaption describes the application of a model for the same task but with different underlying data distribution ($T_T = T_S \wedge D_T \neq D_S$). In the context of the given work, different domains could, for instance, be scientific articles from different fields of research, while examples for different tasks are outlined below, when pre-training objectives for BERT are described. A more extensive formal definition of the problem is given in the work of Zhuang et al. [324] and Ruder [233].

Another central concept, with particular success in NLP is pre-training of models in order to learn general feature representations without task specific training data. In general, the selection of representation determines how difficult a task is to handle [95]. Where classical feature engineering tried to find representations manually, ML models create latent representation during training. When applying pre-training the latent space is already initialized in a meaningful way, shifting the objective from learning a new feature representation for a problem to mapping existing features for a task, reducing the required amount of task specific training data. Empirically, the method works well in the domain of NLP with LLMs achieving state of the art performance in NLP benchmarks since the publication of LLMs such as BERT [67]. The power of language models such as GPT [39] is also demonstrated by their recent application in chatbot systems, e.g., ChatGPT[12].

Pre-training approaches generally work in two steps, where a large-scale model is first pre-trained, and then fine-tuned with task specific data. In the domain of NLP pre-training is commonly performed on unsupervised data, but with an approach that is often referred to as self-supervision [130], where labels are generated based on unsupervised data. This can, for instance, be achieved by masking words from a sentence and predicting the missing word, or by predicting the next word in a sentence. For such tasks a large number of data is available, e.g., NLP applications can be trained on all English websites. A drawback of this approach is that considerable computational resources are required for pre-training and that it is coupled with a high time and energy consumption. Therefore, it is common practice to rely on published pre-trained models, while it should also be considered that these models can learn biases based on the underlying data [196].

BERT [67] combines two different self-supervised training tasks. In the first task, the objective is to predict parts of an input sequence which were randomly masked, leading to a setting in which information from both left and right context of the masked tokens are included in the prediction. The second task is a prediction of the next sentence based on a binary selection between the actual next sentence and a random sentence in a 50:50 ratio. As underlying data for the model the BooksCorpus [323] of 800 M words and English Wikipedia with 2500 M words were used. Different versions of BERT were trained in the literature because it was argued that there is a data distribution shift from general texts to biomedical articles [160]. Specifically SciBERT [27], BioBERT [160], and PubMedBERT [98] were adapted as versions of BERT specifically trained on scientific literature. SciBERT was trained on full-text articles from Semantic Scholar with 18% of articles set in the domain of Computer Science and 82% in Biomedicine, while BioBERT was trained on PubMed abstracts and PMC full-text articles. PubMedBERT was also trained PubMed abstracts and PMC full-text articles, however, in difference to the other two models it was not initialized from a pre-trained BERT model, but newly trained. The authors argue that this leads to the learning of better representations specific to biomedical NLP problems, however, it is not clear how this translates to software related problems.

[12] https://chat.openai.com, accessed 25 January 2024.

3 | Manual Analysis of Software in Scientific Publications

The content of this section is based on the following publications:

> David Schindler et al. “An annotation scheme for references to research artefacts in scientific publications”. In: *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. 2019, pp. 52–57. DOI: 10.1109/PERCOMW.2019.8730730 [250], covers definitions of software related information.

> David Schindler et al. “Investigating Software Usage in the Social Sciences: A Knowledge Graph Approach”. In: *The Semantic Web – ESWC 2020*. 2020, pp. 271–286. DOI: 10.1007/978-3-030-49461-2_16 [251], covers a partial annotation of the established dataset.

> David Schindler et al. “SoMeSci- A 5 Star Open Data Gold Standard Knowledge Graph of Software Mentions in Scientific Articles”. In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. CIKM ’21. Virtual Event, Queensland, Australia, 2021, pp. 4574–4583. DOI: 10.1145/3459637.3482017 [244], covers the full annotation of informal software citations.

> David Schindler et al. “A multi-level analysis of data quality for formal software citation”. In: *Quantitative Science Studies* (2024). Accepted for publication; currently available as Preprint from https://doi.org/10.48550/arXiv.2306.17535 [246], covers the full annotation of formal software citations.

As described in Section 1.2, different stakeholders in the scientific community would profit from systematic large-scale analyses of software mentions in scientific publications. However, existing analyses of software mentions, summarized in Section 2.1, differ strongly in reported results, dependent on underlying data and method of analysis. Furthermore, existing ground-truth annotation for software in science lack important aspects of software citations such as contextual information, identity, and formal software citation (see Sections 3.1.1, 3.1.3, and 3.1.4). In summary, this creates the need for further manual analyses of the problem, and further data annotation to establish a comprehensive ground-truth dataset for the development of automatic approaches. Therefore, this chapter describes the process of a manual high-quality data annotation for formal and informal software citations in scientific publications. Based on the established dataset, initial analyses on the role of software in science are performed and the requirements for following large-scale analyses are systematically assessed.

A central aspect of the analyses performed in this chapter concerns data quality. Measurements of data quality, according to Batini et al. [25], consist of different dimensions including completeness and accuracy. Regarding software citation in general, data quality—specifically completeness—is dependent on the authors of scholarly publications who provide metadata allowing an identification of software and its codebase (see Section 3.1). Formal software citation could further allow an integration of software citations in scientometric analyses, based on existing infrastructure for scientific publications. For this purpose, further analyses of data quality are required, since data quality is known to be essential for bibliographic analyses [106]. While it is the author's responsibility to provide complete information in formal citation, the corresponding reference information needs to be marked up to enable downstream processes on the publisher's side. Compared to other citations, technical updates might be necessary to create suited and machine-readable software representations [267], particularly, regarding special aspects of software such as versioning (see Section 3.1). Here, completeness determines whether all information provided by authors is also reflected in the structured metadata, and accuracy determines whether the information is correct and reflects the same semantics. A semantic representation is important, because metadata that is not correctly structured can become useless for downstream tasks. Moreover, the information from publishers is usually not directly used for applications such as scientometric analyses. Instead these analyses are based on literature databases, and similar to publishers, it is not known whether they adequately reflect software. Therefore, it is also important to investigate how well the information provided by authors and publishers is reflected within these databases. As before, both dimensions of completeness and accuracy can be considered regarding this question.

This remainder of this section first defines and illustrates metadata and information related to software usage in scientific publications, and outlines different practices for formal software citations. Then, it is described how all outlined information was systematically annotated in a set of representative publications to establish a high-quality dataset, which can be utilized for analyses of software mention and citation practices, but can also serve as training and evaluation data for subsequent automatic analyses. Finally, the data is quantitatively analyzed, the results are presented and discussed, and conclusions regarding large-scale analyses of software mentions in science are drawn.

3.1 Software Metadata

There is several essential metadata with respect to scientific software, that is required to identify the software itself, its developer, and the utilized codebase, i.e., the development state in the software life-cycle, see Section 1.4. However, as outlined in Section 2.1 some of this information is often missing in practice. In the following, all metadata considered in this work is defined, illustrated, and its purpose in identifying the software is described. The metadata is based on previous studies and the recently established software citation guidelines [135], which are intended for formal citations but define the overall necessary information to identify software. Initially, metadata relevant for both software mentions and bibliographic references is outlined, then specific information for informal references, and, lastly, specific information for formal references.

The most basic information about software is its **name**, a designated, human-readable identifier which is commonly assigned by the software's developer. It is the main analyses target of most related work on software citation outlined in Section 2.1. Here, software is considered as named entities, which means that software names are required to be proper nouns. However, the word itself is not required to start with an upper case letter because software often receives unconventional names in terms of spelling and casing. Examples for software names are *ImageJ*, *MEGA*, and *R* with further examples given throughout Listings and Figures. Since software names are often acronyms or can be shortened, many authors include more than one name to better identify the software, which is considered as an additional metadata (**alternative name**). For informal mentions the name is always given as the anchor of the mention, while it is possible that the name is missing

```
Reconstructed images were further processed using image-editing software (Adobe® Photoshop®
CS ver. 8.0.1; Adobe Systems Inc., San Jose, CA) when needed to produce printouts.
...
These AOI were automatically counted, and data was exported and saved to a spreadsheet computer
program (Microsoft Office Excel 2003, Microsoft Corporation, Redmond, WA).
```

Listing 3: Examples from the dataset established in the scope of this work, illustrating the use of version, extension and release date. Software name (🟦), developer (🟧), version (🟩), release date (🟥), and extension (🟪) are highlighted. Both sentences originate from Salinas-Navarro et al. [239].

from formal citations associated to software.

The concept of versioning is essential in software development as most software is under constant development and receives regular updates after its initial publication. Therefore, the range of functions and their behavior can vary over time, making knowledge of the applied version a prerequisite for reproducible research. Version indicators are also required by software citation standards [135], while the amount of provided versions in combination with software citations was first manually analysed by Howison and Bullard [118]. In general, versions can be provided by different means: **version numbers** are commonly assigned by developers to uniquely identify the software development state, and use a number format intercepted by delimiters to distinguish major and minor development states; **release dates** define a software development state by a specific date; **extensions** distinguish different meta-versions of a software, e.g., professional and standard distribution. The different types of versioning are illustrated in Listing 3 with version *8.0.1* and extension *CS* for *Photoshop* and release date *2003* for *Office Excel*.

The person, group, or organization developing and maintaining a software is considered as its **developer**. Knowledge of the developer provides a means to closer identify a software and provides attribution for its development. This aspect is also required by citation guidelines [135] and the amount of provide developers was manually analyzed by Howison and Bullard [118]. It has been pointed out that its often difficult to identify the developer of software due to dynamic changes [263, 135]. In large-scale open source projects, for instance, it can be impossible to distinctly identify a developer, while the developer can change in commercial software due to mergers or acquistions. Authors typically only state the current developer of a software at the time they acquired it, in form of a named entity. While this information cannot be considered as complete, it is still valuable since it enables a better identification of a software. Furthermore, information on the full development process of the software might be publicly available and accessible through additional sources once the software was uniquely identified. Examples for developers are given in Listing 3 with *Adobe Systems Inc.* and *Microsoft Corporation*.

A bibliographic reference connected to the software by an in-text citation is considered as a **formal citation**. As described, this practice is advocated by software citation guidelines [135], with its extent having been investigated by Howison and Bullard [118]. This is generally a string identifying a reference of the article's bibliography, as provided in Listing 1. The resolved references do provide further information and are introduced in detail in Section 3.1.4.

URLs are resource identifier connected to the software, which commonly indicate a download location, documentation of the software, or any other website maintained by the publisher. They are valuable resources in identifying a software, and can serve as unique indentifiers in case they are persistent, e.g., pointing to archives such as Zenodo[1]. Their use with software citations has been

[1] `https://zenodo.org/`, accessed 10 March 2024.

```
Diffusion image preprocessing was done using FSL 4.1.4 (www.fmrib.ox.ac.uk/fsl).
```

Listing 4: Examples illustrating the use of a URL. Software name (🟦), version (🟩), and URL (⬜) are highlighted. The example originates from Mulder et al. [190].

```
The pSSAlib source code and pre-packaged installers are freely available from mosaic.mpi-cbg.de
as open source under the GNU LGPL v3 license.
```

Listing 5: Examples illustrating the use of a license. Software name (🟦), URL (⬜), license (🟥), and version (🟩) are highlighted. The example originates from Ostrenko et al. [203]. Note that in this example the version refers to the license, not the software, which has to be annotated and modeled accordingly during data annotation.

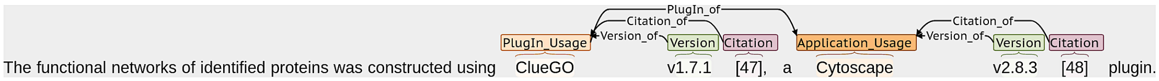
analyzed by Howison and Bullard [118]. However, the URLs provided in scientific publications often point towards software specific websites. These are usually not persistent and might no longer be accessible after only a few years due to link rot [212]. Software citation guidelines [135], therefore, distinguish between **archives**, and URLs, but during data annotation archive URLs were found to be extremely rare. An example for a URL is illustrated in Listing 4. There have also been efforts to introduce unique **identifiers** for research objects such as software, see Section 2.1.4, which are also covered by citation guidelines [135]. Since these identifiers are fairly new compared to DOIs as assigned by Zenodo, further investigation is required to determine which are the most commonly used unique identifiers for software in scientific publications.

The **license** associated with a software declares its permissions and terms of usage, and is mostly mentioned when new software is developed and published as part of the research. Knowledge about the license is important for practitioners working with a software. An example for a license is illustrated in Listing 5 with license *GNU LGPL* associated with software *pSSAlib*.

Additionally, there is metadata that only concerns formal software citation, namely a **description** and **type of citation** can be included in a direct software reference in accordance to software citation guidelines [135]. The description provides additional information about the software, while the type of citation specifies the resource cited in a bibliographic reference and is often stated in square brackets, e.g., “[source code]”. While it is also possible that a description for a software is provided in-text, it is difficult to clearly identify it. In the scope of a formal citation a description clearly provides additional information about the purpose of a software, while in-text the application of a software is described together with its capability. Overall this leads to a situation where a description cannot be concisely defined as in-text metadata.

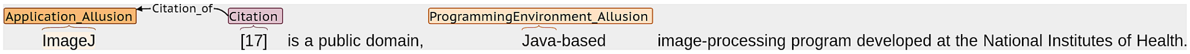
3.1.1 Contextual Information in Software Mentions

All metadata outlined above is available as separate textual information in a statement on software usage or inside a formal reference. Additionally, contextual information associated with software is present in informal mentions. The *software type* distinguishes four different types of software applied in scientific research and allows deeper and more specific analyses of software and their interaction. The distinction is based on the work of Li et al. [166] who distinguish between end user applications and PlugIns:



The functional networks of identified proteins was constructed using ClueGO v1.7.1 [47], a Cytoscape v2.8.3 [48] plugin.

Figure 3.1: Complex software mention example containing the mention of a *PlugIn* in combination with a host *Application* software. The example further illustrates relations between metadata in a setting where relations are not directly clear. The example originates from Gangoda et al. [88], and is illustrated by a screenshot from the annotation tool BRAT (see Section 3.2.2).



ImageJ [17] is a public domain, Java-based image-processing program developed at the National Institutes of Health.

Figure 3.2: Example illustrating the mention type *Allusion* and the software type *PE*. It originates from Bigras et al. [32].

- *Application* is a stand-alone program, including web-services. Further, the definition includes compiled programs that can be readily installed, but also source code that can be obtained and executed by a user.

- *PlugIn* is a software that is specifically built upon an existing software to extend its function range, and cannot be executed on its own. The combination of a *PlugIn* and its corresponding host-software is illustrated in Figure 3.1.

- *Programming Environment (PE)* is an environment used for writing executable code that is based on a programming language but also include the means to execute the program, i.e., a compiler or interpreter. The mention of a *PE* is illustrated in Figure 3.2.

- *Operating System (OS)* is the basic software for any computer system used to manage the hardware of a computer and its software processes.

There are cases in which the distinction between software types is not clear, for instance, software can be available both as a *PlugIn* and *Application*. Furthermore, there are cases where a software has clear programming elements but also aspects of a stand-alone *Application*, e.g., MATLAB. Lastly, it is difficult to distinguish scientific databases from web-services, especially if they allow interactive exploration. Here, databases are not considered software as long as they only implement data storage, filtering, and selection. If they include complex analytic tools, e.g., for gene sequences, they are considered as *Applications*.

Authors of scientific articles have different intents when mentioning software in their publications. Therefore, the *mention type* is considered here to capture the intent to enable more specific analyses. It is based on the work of Howison and Bullard [118], who distinguish between software that was only mentioned and software that was used, which is further extended to the following four *mention types*:

- *Allusion* describes the mere statement of a software name without any further indication of how it relates to the described investigation, e.g., a fact is stated about a software or multiple software are compared. An example is provided in Figure 3.2.

- *Usage* statements indicate that software was applied as part of the investigation, and it can be inferred that the software has made a contribution to the study results, making it a part of the research's provenance. Examples for software *Usage* statements are provided in Listing 1, 3, and 4.

```
PMC3900378:
The VBA toolbox is then used to invert a "dynamical" variant of the Q-learning model (cf.
section "Reinforcement learning models of choice data"), given the agent's sequence of choices.

PMC3492432:
The re-sampling procedures were conducted in Microsoft Excel 2010 using VBA code written by
FAC (Appendix S2).
```

Listing 6: Examples illustrating two mentions of a software called *VBA* referring to two different software entities. Software name (🟦), developer (🟧), and release date (🟥) are highlighted. The first sentence originates from Daunizeau et al. [62] and the second from Wikberg et al. [306].

- *Creation* statements indicate the development of a new software as part of an investigation. Knowledge about established software is valuable to assess the overall available research software and to provide scientific credit for its development. The software can either be the main focus of a software article (see Section 3.1.4) or only a partial aspect of an investigation.
- *Deposition* statements are a special case of *Creation* statements, which do not only indicate that a software was created, but that it was also published and made available to the public or other researchers. In this work, a deposition statement is required to either include a publication URL or license. Knowledge about the deposition of a software enhances the information on created software and enables analyses on its availability and terms of usage. A deposition statement is illustrated in Listing 5.

3.1.2 Relations in Software Mentions

Up until now it was explicitly assumed that the described metadata is associated to software. In practice, it is necessary to explicitly model the relations between metadata. The reason are complex cases as depicted in Listing 5 and Figure 3.5, where a version is connected to a license instead of a software or where multiple software and metadata are mentioned together. In general, all metadata except the name requires a relation to other metadata, where all metadata can relate to a name, representing the software itself as anchor of informal mentions. Additionally, URLs and alternative names can also relate to developers and licenses and versions can relate to licenses. Moreover, software names can be related among each other if one software is a *PlugIn* and the other the corresponding host-software (see Figure 3.5).

3.1.3 Identity

All software mentions and formal references refer to real world software. The software identity is a concept uniquely identifying the real world software and allowing a clear distinction between software. In general, software cannot be distinguished based on software name alone due to spelling variations or errors present in scientific literature [76]. Software names can also vary between scientific disciplines, due to cultural differences, name changes over time, or marketing decisions. It is also possible that different software has the same name. An example is provided in Listing 6 regarding the name *VBA*, that refers to the software *Variational Bayesian Approach* in one case and to *Visual Basic for Applications* in another. Moreover, some software becomes unavailable over time which can make a subsequent identification based on name alone impossible. Therefore, unique identifiers, as advocated in citation standards [135], are of high importance. With respect to

```
Weckx S, Del-Favero J, Rademakers R, Claes L, Cruts M, et al. (2005) NovoSNP, a novel
computational tool for sequence variation discovery. Genome Res 15: 436-442.
```

Listing 7: Example for the bibliographic entry of a software article. The example originates from Zhang et al. [319] and the content of the bibliography entry is provided as in the original PDF publication. All information describing the software is highlighted with developer (🟧) and name (🟦). Note that a publication year is provided, but does not provide information about the used software and codebase but about the article.

analyses, the software identity is the only way to aggregate over all references of the same software, which is a prerequisite to many important analyses, e.g., an impact measure for software or an analysis of the software distribution in science. Therefore, software ED or EL is an essential aspect regarding analyses of software in science, which needs further investigation, as it is not taken into account by most prior work regarding software analyses (see Section 2.1).

Moreover, the concept of identity is not restricted to software, but does also concern its related metadata. Specifically, the software developer, formal citation, and license have clear identities, that can be further disambiguated beyond the name alone. Only with respect to version, no ED should be required, as versions are defined to be unique in the context of a specific software.

3.1.4 Formal Software Citation Types

As discussed in Section 1, software used in scientific investigations should be adequately indicated to ensure reproducibility and provide provenance information about the research results, with current guidelines suggesting a formal citation [135]. However, different types of resources can be referenced by the bibliographic entry associated with a software, because different software citation practices exist in the scientific community. Not all of these practices are suited for formal software citation because they might not provide all necessary information to identify the software. The relevant resource types were defined based on the previous work of Howison and Bullard [118], who distinguished between citations to publications, user manuals, and project names or websites. The considered types were then further extended based on observations made during data annotation.

Software articles are scientific articles describing a research software that are published by developers of the software, and cited in its place [94]. An example is provided in Listing 7. The practice is common as it allows developers to receive scientific attribution for the costly development of research software. Historically, publishing papers about software was considered a stronger contribution than the publication of the software itself [101]. Software articles are among the highest cited scientific papers [288] and specific journals for publishing software articles have been established, e.g., *The Journal of Open Research Software* or *Source Code for Biology and Medicine*.

There are drawbacks to the use of this citation style. In general, software articles are cited the same way as other scientific articles without software specific information. Therefore, information identifying the codebase such as the version or release date is generally missing from this citation type. Furthermore, a software article might not credit the full developer team, particularly, concerning aspects such as long time maintainance.

Software manuals are textual instructions for using a software, and, often the closest textual document associated with a software, especially for commercial software. It is an established practice to cite such manuals instead of the software itself with an example given in Listing 8. Corresponding references are formatted for citing a text source with information typically provided for online sources such as an URL and a date of access. Similar to software articles, this citation type omits crucial information closer describing the corresponding software.

```
Muthen LK, Mutheń BO. Mplus User's Guide. Seventh ed. Los Angeles, CA: Mutheń & Mutheń;
1998-2012.
```

Listing 8: Example for the bibliographic entry of a software manual. The example originates from Maraz et al. [182]. Information identifying the software, its developer, and the used codebase are highlighted: developer (🟧), name (🟦), version (🟩). Note that information on the version or publication date of a manual can in some instances be used to identify the software codebase if the manual is updated with every release of the software and authors cite it appropriately.

```
Accelrys Software (2005) DS Viewer Pro 6.0 [computer program]. Available:
http://www.accelrys.com/dstudio/ds_viewer/index.html. Accessed 3 November 2007.
```

Listing 9: Example for the bibliographic entry of a direct software citation. The example originates from Del Sol and Carbonell [64]. Information closer identifying the software is highlighted: developer (🟧), year (🟥), name (🟦), version (🟩), type of citation (🟫), URL (⬜), date of access (🟪).

Direct software citations (referred to as direct citations) describe the cited software itself and are the recommended way to cite software by recently established software citation guidelines [135]. An example for a direct software citation is given in Listing 9. Properly executed, direct citations capture all metadata of software that is required for a unique identification of the software, its developer, and the exact codebase. In practice, not all required information may be present and the references themselves can be arbitrary structured as there is no commonly agreed upon citation style for direct software citations.

Websites associated with a software are sometimes cited instead of a software. The corresponding references are structured as typical online resources, potentially providing the date of access. An example is provided in Listing 10. Same as for other styles, relevant information specifying the software itself is missing. As direct citations often include URLs, it is important to distinguish them from websites. Here, all cases where additional information about the software is provided (except name, URL, type of citation, and date of access) are considered as direct citations.

Lastly, **Other** citations describe the rare instances in which no verifiable resource is described by a reference. Those cases are present in practice and are likely to result from faulty automatic citation recommendations or author errors.

3.2 Dataset Creation

This section describes the process of establishing a high-quality ground-truth dataset reflecting software mentions and formal citations in scientific literature. For this purpose, a set of representative articles was selected, and manually annotated to capture all information introduced above. As outlined in Section 2.1, existing datasets do not fully assess the state of software citation and are lacking important aspects of software in science to serve as ground-truth data for developing automatic extraction methods. A detailed comparison of these annotated resources and the dataset established in this work with respect to size and contained information is provided in Table 3.3 within Section 3.3.

Data size and annotation quality are two central concerns for creating the dataset. Both aspects influence the reliability of statements made about software in science based on it, and its applicability as ground-truth data for ML models. In general, a direct trade-off exists between data size and annotation effort. ML models require large datasets to perform well, while annotation of textual

```
gplots-Rpackage. Available: http://www.rdocumentation.org/packages/gplots/versions/3.0.1
```

Listing 10: Example for the bibliographic entry of a software website. The example originates from Malvisi et al. [179]. Information identifying the software, its developer, and the used codebase are highlighted: name (▨), type of citation (▨), URL (▨).

documents, specifically scientific texts, is a highly expensive process because it requires trained annotators to perform the complex annotation task. A potential reference point to estimate required data size are existing datasets. Here, the CoNLL dataset [281], a well-solved standard benchmark for NER is selected, with the objective of matching at least the same order of magnitude in sample number. The sample number range between 5062 and 10,645 for the English, and 3968 and 6579 for the German benchmark set. With respect to annotation quality, the annotation process is split up into individual steps to reduce its complexity, and the annotation quality is separately evaluated at every step, with details described below.

The remainder of this section first describes the selection of a representative article set, introduces measures for annotation quality, and then outlines all steps of the annotation process including background on the annotation framework and the involved annotators.

3.2.1 Article Selection

A base requirement for articles selected for the dataset is that they are published under a license that allows republication so the annotated resource can be released as a stand-alone dataset covering software citation. This excludes a large amount of publications that are either pay-walled or published under terms that do not allow a republication. A notable exception is PMC OA containing more than 3 M freely available articles with suited licenses to make the resulting dataset available to the scientific community. Furthermore, articles from PMC OA are available as Journal Article Tag Suite (JATS), a XML-based format for scientific publications using a common mark-up scheme that allows the extraction of both metadata and plain text. This facilitates text extraction and prevents errors compared to using PDF documents, which requires a PDF to text conversion, potentially leading to artifacts due to markup and encoded metadata, e.g., journal specific formatting including headers or footers.

On the other hand, PMC OA has the drawback of being built primarily for biomedical literature and has an overall bias towards Life Sciences. The Public Library of Science (PLoS) is one of the largest open-access publishers indexed in PMC OA, with an interdisciplinary focus covering a broad range of scientific domains. Therefore, all articles included in the dataset were sampled directly from PMC OA or from PLoS, to reduce biases by journals and domains.

In general, annotation of full-text articles is a highly time-consuming task. Therefore, it was considered to only annotate specific segments of an article, to reduce annotation time, while achieving good generalization. Specifically, four different data sets were selected to optimize annotation effort while still fully covering the intricacies of software mention and formal citation in science.

- *PLoS methods sections (PLoS_m)* is the largest subset in terms of contained software mentions. It consists of 480 "Materials and Methods" (methods) sections automatically extracted from 500 randomly sampled articles from PLoS tagged with the term "social science", with 20 articles omitted because they did not contain a methods section. The selection of methods sections was performed to reduce annotation effort based on the assumption that the application of software is most frequently described as part of the methodological description. The

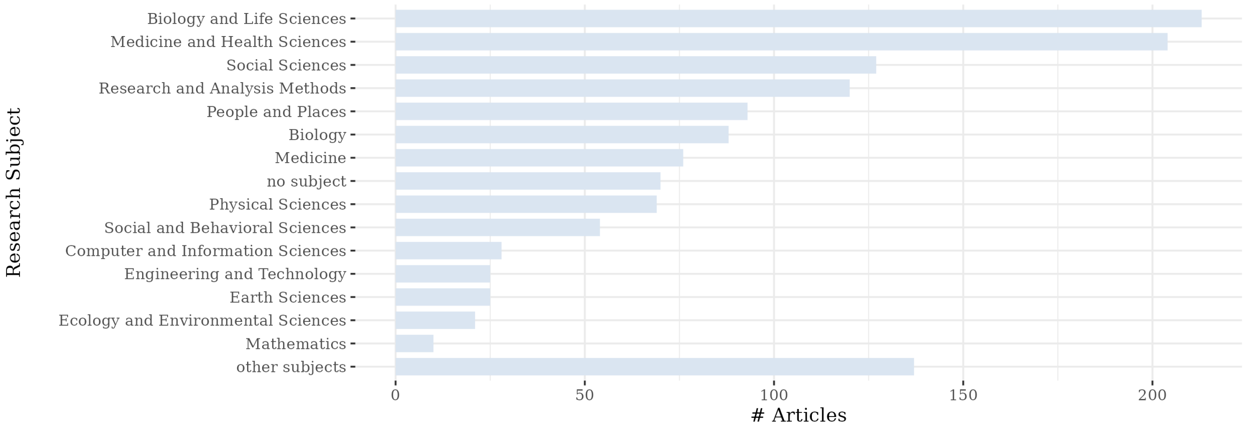


Figure 3.3: Research subjects of articles in the *PLoS_m* subset.

article set was initially selected to represent Social Sciences, but the distribution of scientific disciplines within the selected articles, illustrated in Figure 3.3, shows that a wide variety of domains is present, even so the main focus remains Life Sciences.

- *PubMed full-text documents (PubMed_f)* contains 100 full-text document articles, to allow an assessment of generalization capabilities of ML methods to full-text documents. The specific number of articles was chosen as a trade-off between annotation effort regarding full-text document articles and sufficient sample size for reliable generalization statements, with details on annotation efforts outlined in Section 3.2.4.

- *PLoS sentences (PLoS_s)* consists of individual sentences selected from 677 PLoS articles that contain the names of common software, selected based on prior work in software identification [205]. They were selected to increase the number of mention contexts with minimal annotation effort while also increasing the number of negative samples if the identified word does not actually refer to a software in the given context.

- *Creation sentences (Creation_s)* consists of individual sentences from 110 articles (50 PMC OA and 60 PLoS), that describe software development. They were specifically selected to increase the number of samples for the otherwise rare cases of software *Creation* and *Deposition*, described in Section 3.1.1. The articles were automatically pre-selected by analyzing titles and abstracts for structure and keywords, which indicate articles to be related to software development. Resulting articles were then randomly selected and manually verified, before specific sentences containing software *Creation* and *Deposition* statements were manually extracted. Additionally, a limited number of *Allusion* and *Usage* statements were also extracted and added to the set when they were observed in the context of *Creation* and *Deposition* statements to further increase the overall number of samples.

3.2.2 Annotation Implementation

The annotation is implemented through manual annotation by human annotators. The literature states that for high quality annotations it is important that annotators understand the research question and the purpose of the annotation [86]. This is the case for all three annotators involved in the performed annotation process because they are all involved in research on scientific software mention and citation. Furthermore, all annotators have a background in computer science or engineering and a general understanding of software in science. All concepts to annotate were discussed between the annotators, all discrepancies within the initial overlapping annotations were resolved, and all issues or challenging examples were further examined by multiple annotators throughout the entire annotation process. A different number of annotators was involved in every annotation step, as they took different effort, with the details being provided below.

To further optimize the annotation process it is aided by tools which allow the annotators to work efficiently. BRAT v1.3 [272, 271] is utilized for text annotation, the most complex annotation problem in the given context. It was chosen because it is widely used, well established, and can easily be run on a local installation without specific hardware requirements, and it allows to implement all relevant text annotations. Annotations made with BRAT are illustrated in Figures 3.1 and 3.2. Moreover, a simple custom tool was implemented to perform fast point-and-click annotation of citation types for software references. Further, automatic checks were implemented to ensure annotation consistency, e.g., checks for missing entity relations.

Since human annotators are prone to errors and data quality is crucial for all downstream applications of the dataset (see Section 2.4.1), the annotation quality was assessed at every step throughout the annotation by measuring the IAA on overlapping annotations. For this purpose, two measures were mainly applied. Cohen's κ [55] was utilized for categorical annotations, and FScore for text annotation as suggested by Hripcsak and Rothschild [121] (both measures are introduced in Section 2.4.2). Cohen's κ is suited for categorical annotation because it accounts for chance agreement in the evaluation, however, it was argued that it is unsuited for full-text document annotation because chance agreement can be neglected when random phrases are annotated in a given text. Furthermore, the κ statics is missing a well defined negative class [121], which could be any arbitrary selection of non-annotated phrases in text annotation. Therefore, the FScore was suggested as a better suited alternative. Note that Recall and Precision, in difference to FScore, are ill-defined for IAA as they depend on the notion of prediction and ground-truth, which is not given in manual annotation. The FScore, however, is symmetric and can be applied by simply assuming one annotation as the ground truth and the other as the prediction [121]. In general, all outlined evaluations do not consider partial overlap as matching annotation, but as an error, because annotation boundaries are important for consistency and there is no separate control step for the annotation.

3.2.3 Annotation Pipeline

The annotation is split into multiple steps to reduce the complexity of the annotation and to improve the resulting annotation quality. Most steps are hierarchically dependent, which leads to re-examination of previous annotations at each step and can help uncover errors in earlier annotation stages, particularly if the annotators switch between the annotation stages. Even so the pipeline is defined for software it can serve as a general template for future annotation of information in scientific articles due to the high IAA that was achieved based on the described process. An overview of the annotation pipeline is given in Figure 3.4 and the distinct steps are described in the following:

1. *Entity (Software):* In the first annotation step, software name, software type, and mention type are annotated, as well as relations between software, making it the most important step of the annotation process. The corresponding annotation result is illustrated in Figure 3.5. The IAA was calculated on 10% overlapping samples of each of *PLoS_m*, *PLoS_s*, and *PubMed_f*. *Creation_s* were excluded because the annotation methods varied from the other sets since

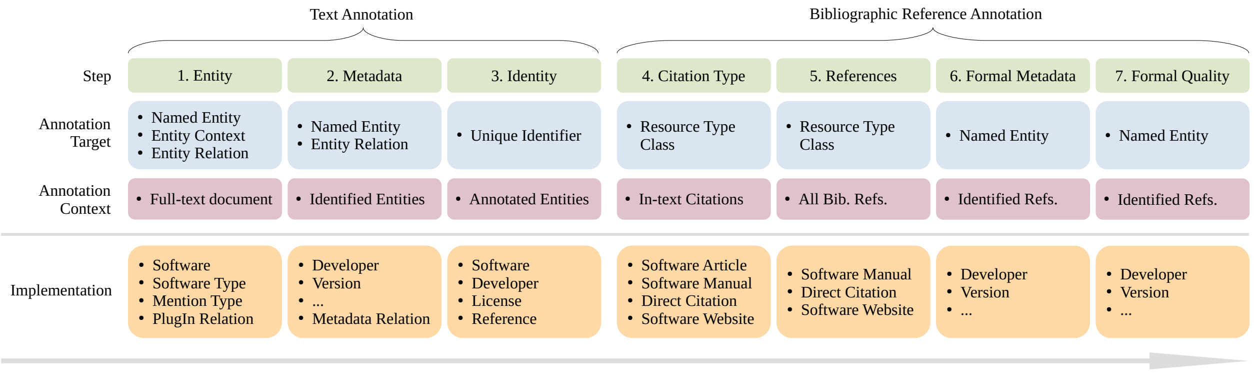

Text Annotation
Bibliographic Reference Annotation
Step
1. Entity
2. Metadata
3. Identity
4. Citation Type
5. References
6. Formal Metadata
7. Formal Quality
Annotation Target
• Named Entity • Entity Context • Entity Relation
• Named Entity • Entity Relation
• Unique Identifier
• Resource Type Class
• Resource Type Class
• Named Entity
• Named Entity
Annotation Context
• Full-text document
• Identified Entities
• Annotated Entities
• In-text Citations
• All Bib. Refs.
• Identified Refs.
• Identified Refs.
Implementation
• Software • Software Type • Mention Type • PlugIn Relation
• Developer • Version • ... • Metadata Relation
• Software • Developer • License • Reference
• Software Article • Software Manual • Direct Citation • Software Website
• Software Manual • Direct Citation • Software Website
• Developer • Version • ...
• Developer • Version • ...

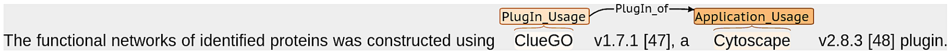
Figure 3.4: Overview of the annotation pipeline for research artifacts in scientific literature split between text annotation and reference annotation. The illustration shows the annotation target at each annotation step, and the context the annotators need to examine at each step.

the individually annotated sentences were selected during annotation. The text annotation of software names was performed with F=95% (n_1=329, n_2=339). The categorical annotations of software type and mention type were annotated with respective κ values of κ=.82 and κ=.79, with error propagation taken into account, i.e., non-overlapping software annotations are included in this calculation as errors. Relations between software were considered as a categorical annotation with a clearly defined negative case, in which no relation exists between two software mentions. Here, the IAA was evaluated at κ=1 based on n=16 related software. Overall, the IAA was considered sufficiently high for the remaining articles to be annotated by a single annotator, while all discrepancies between annotations were re-evaluated and the annotations updated. However, determining the software type and mention type proved to be challenging even so the IAA values can be considered in the ranges of "almost perfect" and "substantial" agreement [156].

2. *Metadata:* In the second step, all metadata associated with software and its relations were annotated. An illustration of the annotation after this step is provided in Figure 3.1. The annotation is performed by re-examining the immediate context of annotated software from the first annotation step. Therefore, the annotation effort in this step is significantly lower, as only excerpts of the text need to be analyzed by the annotators. After this step, all information regarding software contained in the textual description is annotated. As in the first step, the IAA was calculated on 10% of overlapping samples from each of *PLoS_m*, *PLoS_s*, and *PubMed_f*, including the annotation results from the first step. The text annotation of additional information was estimated with an overall FScore of F=96% (n=293) with scores for all individual entities provided in Table 3.1. Furthermore, the categorical relation classification was evaluated at a value of κ=.98. The IAA for metadata was considered sufficiently high for a single annotator to annotate all remaining samples.

Metadata Information	F (%)	n
Version	98	136
Release	89	5
Extension	100	4
Developer	93	85
Citation	92	40
URL	100	16
Alternative name	100	7
Overall	96	293

Table 3.1: IAA for metadata annotation.



The functional networks of identified proteins was constructed using ClueGO v1.7.1 [47], a Cytoscape v2.8.3 [48] plugin.

Figure 3.5: Annotation state after the first annotation step in which software name, software type, mention type, and relations between software are annotated, corresponding to the example given in Figure 3.1.

```
"https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5690316": {
  "sentence_119": {
    "string": "R language [30] (version 3.2.2; R Foundation, Vienna, Austria) was used for
              statistical analysis and creation of several figures.",
    "links": {
      "R":{"beg": 0, "end": 1, "link":"https://www.wikidata.org/wiki/Q206904"},
      "[30]":{"beg": 11, "end": 15, "link":"https://doi.org/10.1080/10618600.1996.10474713"},
      "R Foundation":{"beg": 32, "end": 44, "link":"https://www.r-project.org/foundation/"}
    }
  }
}
```

Listing 11: Information on unique identities for software (*R*), formal citation (*[30]*), and developer (*R Foundation*) in an annotated sentence formatted as JSON. The example corresponds to the example given in Listing 1.

3. *Identity:* In the third annotation step, the identity was annotated for all software, developers, licenses, and formal citations. Wikidata[2], introduced in Section 2.2, was selected as the primary source for unique identifiers because it is a general purpose KG that covers a broad range of entities. When software was not covered in Wikidata, package repositories such as CRAN[3], Bioconductor[4], or PyPi[5] were considered, before Github[6] and general websites were considered as alternative sources in the stated priority. Regarding developers, Wikidata entries were prioritized over general websites, licenses were assigned URLs corresponding to *opensource.org*, and formal citations were assigned DOIs. An example for annotated identities is provided in Listing 11. If possible, all links were adjusted to the suggested canonical form. The IAA for this step was initially assessed on 10% of software. It was determined by overlap because all entries differ in value and chance agreement for entering URL based identifiers can be neglected. However, only 68% agreement was achieved in the initial assessment, which was considered insufficient. Therefore, all entries were annotated twice and discrepancies merged in order to ensure high quality annotation. Further investigation regarding the reason for low annotator agreement showed that the errors were due to: different web-resources available for the same entity; resource locations changing over time; no use of canonical form; old software without valid, persistent identifiers; and information hidden behind pay walls.

4. *Citation Type:* In the fourth step, the type of formal citation introduced in Section 3.1.4 was annotated for all formal references associated to software citations. The reference content was obtained from the article JATS. The annotation decision was based on the reference information including an online search for the referenced resources in cases of uncertainty. An initial overlapping annotation of 10% of references for this categorical problem showed an IAA of κ=.75, which can be considered substantial agreement [156] but was determined

[2] https://www.wikidata.org/, accessed 10 March 2024.
[3] https://cran.r-project.org/, accessed 7 February 2024
[4] https://www.bioconductor.org/, accessed 7 February 2024.
[5] https://pypi.org/, accessed 7 February 2024.
[6] https://github.com/, accessed 7 February 2024.

```
<ref id="pcbi-0030239-b042">
    <element-citation publication-type="other">
    <collab>Accelrys Software</collab>
    <year>2005</year>
    <source>DS Viewer Pro 6.0 [computer program]</source>
    <comment>Available:
        <ext-link ext-link-type="uri" xlink:href="http://www.accelrys.com/dstudio/ds\_viewer/index.html">
        http://www.accelrys.com/dstudio/ds_viewer/index.html
        </ext-link>. Accessed 3 November 2007.
    </comment>
    </element-citation>
</ref>
```

Listing 12: Corresponding JATS entry for the direct software citation in Listing 9, highlighting developer (🟧), year (🟥), name (🟦), version (🟩), type of citation (🟫), URL (🟦), and date of access (🟪). Additionally, metadata that is not structured with a corresponding label is highlighted (⬚).

as insufficient for the annotation. Therefore, all samples were annotated twice by different annotators, and discrepancies were discussed and re-annotated to ensure high annotation quality.

5. *References:* It is possible that software references exist that are not linked to an informal software mention, for instance, when software is mentioned by generic phrases (e.g., "source code") instead of its name. While Howison and Bullard [118] report this case to be uncommon and to only account for 1% of software mentions, further reasons might exist in practice. Therefore, in the fifth annotation step, the extent of this practice is investigated by annotating all bibliographic references in the *PLoS_m* and *PubMed_f* sets for citation type. The remaining articles were not included to keep the number of references and the corresponding annotation cost at a feasible level. Moreover, the annotation was implemented in two steps to improve Recall and annotation efficiency, and performed based on the reference information available from the publisher (as illustrated in Listing 12). In the first step, it was only annotated if a reference is potentially relevant, and in the second step the citation type (see Section 3.1.4) was annotated for the marked references. Here, the citation type of software article was excluded from the task, because it can, in general, not be distinguished from regularly published articles without examining its content, making the task infeasible in the scope of this annotation. The annotation quality of this annotation was measured by assessing the Recall based on the known references connected to informal software mentions that are known to be present in the references from previous annotations. Overall, 74 of 75 (99%) known direct citations, 24 of 25 (96%) manuals, and 5 of 5 (100%) *Websites* were successfully identified, which was considered as satisfactory quality.

6. *Formal Metadata:* In the sixth annotation step, metadata in formal software citations was annotated for all available direct software citations, manual references, software websites, and other references (as defined in Section 3.1.4), identified in the previous two annotation steps. The corresponding reference texts were manually extracted from the published PDF documents in February, 2023. Examples of the annotation are provided in Listing 8, 9, and 10. The annotation was performed by two annotators and the IAA was calculated on 10% overlap at a value of κ=.82, which is considered in the range of "almost perfect" agreement [156]. The remaining data was then annotated by a single annotator, while challenging cases were further discussed throughout the process.

7. *Formal Quality:*

   In the seventh and final annotation step, representations of formal software citations by providers of bibliographic data were annotated for the same references as in the previous step. Specifically, the information provided by the publisher and the information provided by state of the art literature databases were analyzed regarding their data quality, as both contribute to the databasis of scientometric analyses (see Section 3). Crossref[7] was selected as the first example of a literature database. It is the result of a joint effort by an association of publishers with over 17,000 members as of June 2023, with the goal to improve linking between publications made by heterogeneous publishers [153]. The main application of Crossref is a database of metadata for scholarly articles and professional materials that enables unique identification of covered resources by incorporating and introducing persistent identifiers. Crossref was selected because the corresponding bibliographic information is integrated into the database by publishers with central quality control by Crossref. Metadata is further enriched by Crossref, mainly by adding citation links, but also through adding further information such as funder registry information or journal classification codes [109]. Moreover, Crossref is widely used for scientometric analyses, for instance, for citation analyses by Dion et al. [71] and Peroni et al. [214] and citation network analyses by Cho and Yu [51], and enables such analyses by making its database openly available without any license restriction through an API. Regarding software citations, Crossref does provide information to publishers, but does not consider a specialized format taking into account aspects such as versioning, and recommends a submission as unstructured citations [154].

   Semantic Scholar[8] was chosen as a second example for a literature data base because it is also well established but based on a different methodical approach. It is a discovery service for scientific literature [295] developed and maintained by the Allen Institute for Artificial Intelligence (AI2). The service is based on S2AG containing metadata of scientific publications (see Section 2.2.1). A major aspect of Semantic Scholar is to integrate ML methods to enhance data quality and search. This includes the development of a system for publication deduplication named S2APLER and citation linking based on fuzzy text-matching heuristics [143]. Data is provided free and open by Semantic Scholar via an API. Semantic Scholar it is widely used as a search mechanism for academic publications, while the underlying KG also enables scientometric analyses [194].

   The publishers' references were automatically gathered while the corresponding Semantic Scholar and Crossref entries were manually gathered in August, 2022. An automatic collection was not possible due to partially missing entries in both databases that hindered precise matching. Additionally, multiple entries per reference are in some cases present in Semantic Scholar, which were all extracted and annotated separately. The extracted information was then annotated for the same information considered as in the previous annotation step. Additionally, to allow an assessment of representation data quality, specific tags were introduced in the annotation:

   - *unstructured* marks information that is not labeled within the database, but instead part of a single field containing multiple information about the software. Entries can be partially structured, e.g., the developer and publication date being labeled, but version and software name being unstructured within one field.

[7] https://www.crossref.org/, accessed 19 January 2024.

[8] https://www.semanticscholar.org/, accessed 19 January 2024.

```
{'paperId': None, 'externalIds': None, 'url': None, 'title': 'DS Viewer Pro 6.0 [computer program]',
'venue': 'Accelrys Software', 'year': 2005, 'publicationDate': None, 'journal': None, 'authors': []}
```

Listing 13: Semantic Scholar reference entry corresponding to the JATS entry in Listing 12 [ID: PMC2134966, Semantic Scholar ID: 1116831, [64]]. Metadata is highlighted for: developer (🟧), year (🟥), name (🟦), version (🟩), type of citation (🟫). Metadata represented in an unstructured manner (⬚), for which no label is provided, and metadata represented with the wrong label (⬚) are marked. Information on the URL and date of access from the original publisher information is missing.

```
{'key': 'pcbi-0030239-b042', 'unstructured': 'Accelrys Software 2005 DS Viewer Pro 6.0
[computer program] Available: http://www.accelrys.com/dstudio/ds_viewer/index.html.
Accessed 3 November 2007.'}
```

Listing 14: Crossref reference entry corresponding to the JATS entry in Listing 12 [ID: PMC2134966, [64]]. Metadata is highlighted for: developer (🟧), year (🟥), name (🟦), version (🟩), type of citation (🟫), URL (🟦), and date of access (🟪). Metadata represented in an unstructured manner (⬚) is marked.

- *wrong place* indicates that information is structured but with a false underlying concept, e.g., a developer being labeled as a publication venue;
- *wrong content* indicates that the information in a database is wrong;
- *incomplete content* indicates that some information is only partly represented, and part of the original information is lost;
- *duplicate* indicates duplicate information introduced by a database. Note that this entry refers to duplicate information within one reference, not the duplicate entries for one reference within Semantic Scholar mentioned above.

Since this annotation proved to be challenging from the beginning, all references were annotated by two annotators with discussion of all challenging cases to achieve high quality data. The result for this annotation is illustrated in Listings 12, 13, and 14, which illustrate the respective JATS, Semantic Scholar, and Crossref representations. Potential errors are illustrated in both Listing 13 and 15, including unstructured representation, incomplete content, and wrong content.

3.2.4 Annotation Effort

The overall annotation effort for creating the dataset is estimated at about 593 h, with the textual annotation taking up the most time with about 480 h. In detail, the annotation of *PLoS_m* took up the largest part with 320 h. The value is estimated based on an average duration of 20+5 min for the two annotation runs per article, followed by 15 min of identity annotation for an average of three software mentions, and their developers, licenses and citations. Annotation of *PMC fulltexts* took 112 h with an estimated 45 min for both annotation passes followed by 22 min for identification of software identities. For *PLoS_s* an average of 3 min per article is estimated and for *Creation_s* 10 min, summing to a total of 34 h and 18 h, respectively.

The remaining 113 h were allotted to formal reference annotation. In detail, the annotation of 603 references for citation type took an estimate of 30 s per reference, summing to a total of 11 h for two annotators and the subsequent re-annotation of 68 references. The annotation of ≈29,000 references to identify software citations without plain text mentions is estimated to take 5 s per reference in the first run summing to 40 h, and 20 s in the second run summing to 4 h. The

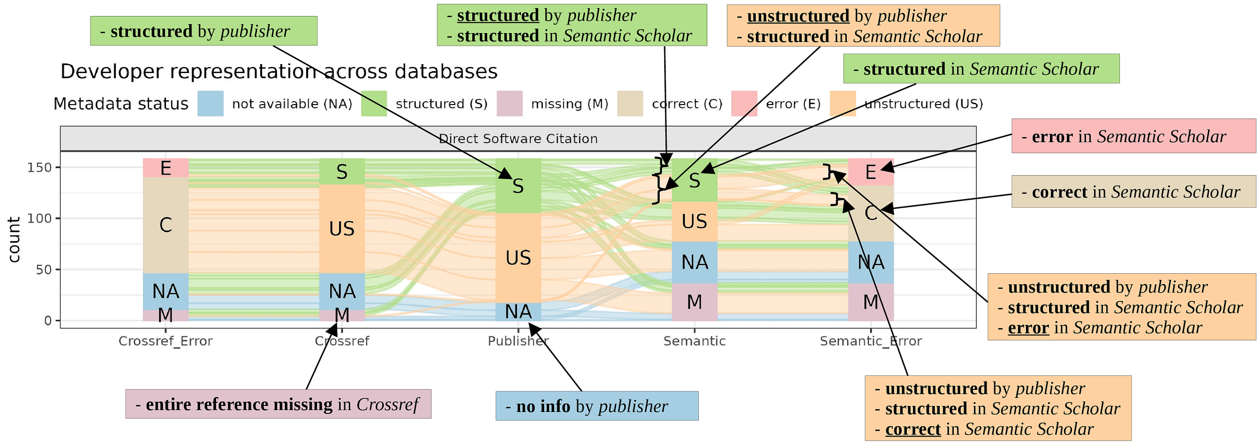


Figure 3.6: Adapted alluvial plot with added explanation of information flow. The detailed plot design is described in the text.

annotation of plain text annotation completeness for all direct software citations is estimated with around 2 *min* per 205 references, summing to 8 *h* including the overlap for quality control. The final annotation of JATS and database entries—including gathering the corresponding entries—is estimated around 7 *min* per reference, summing to 48 *h* as all references were examined by two annotators with additional 2 *h* for manual identification of database entries.

3.2.5 Adapted Alluvial Plots

The results regarding database data quality are illustrated through adapted alluvial plots, with an example illustrated in Figure 3.6. The plots assume an annotated set of samples according to annotation step seven described above, and take into account one specific metadata at a time, e.g., the developer in Figure 3.6. All annotated samples are individually listed in the plot from top to bottom while their order can change from left to right. The flow of a specific sample is indicated by the color originating from the middle column named *Publisher*. If multiple samples have the same information flow their lines are summarized. The middle *Publisher* column shows the information available from the publisher's JATS file, and indicates whether the information is available and whether it is correctly structured. The columns left and right from the middle show the same information for Crossref and Semantic Scholar (*Semantic*), respectively. The outermost columns *Crossref_Error* and *Semantic_Error* show whether the represented information is correct or whether an error is present, for Crossref and Semantic Scholar, respectively. Since it was observed that some references are missing entirely in Crossref and Semantic Scholar they are shown with the special label "missing" to indicate that no information is available in these cases. As a result, the information flow illustration allows a direct comparison of how the different sources structure metadata and whether errors are introduced. Particularly, the difference between the structure provided by the publisher and the corresponding representation by the databases can be observed.

3.3 Results

This section summarizes the results of the manual analyses of software in science. It initially gives an overview of the established dataset—from here on referred to as SoMeSci (SOftware MEntions in SCIence)—and compares it to existing resources of software in scientific publications. Then, it

		Mention Type				Metadata									
	Total	Allusion	Usage	Creation	Deposition	Version	Developer	Extension	Alt. Name	Citation	Release	URL	License	Abbreviation	PlugIn
Appl.	2699	193	2207	197	102	1222	772	47	31	411	50	236	34	65	70
PlugIn	425	51	275	70	29	30	12	–	4	142	1	56	12	12	6
OS	173	30	143	–	–	31	9	5	–	–	1	–	–	–	–
PE	421	47	374	–	–	70	73	–	–	41	35	8	–	–	137
Developer									–			7		2	
License						28			–			2		2	
Overall	3718	321	2999	267	131	1381	866	52	35	594	87	309	46	81	213

Table 3.2: Overview of the frequency of the different label types. Note that only the most specific Mention Type is counted. Combinations that are not possible were left blank, while combinations that did not occur were indicated by –.

outlines some general, preliminary analyses of software in science that build the basis for subsequent large-scale analyses, followed by an analysis of software citation completeness for both formal and informal citation, and their combination. Lastly, it investigates how well formal software citations are currently represented by providers of bibliographic data to investigate if their data can be integrated in large-scale analyses on software in science.

The SoMeSci dataset contains 47,837 sentences out of 1367 scientific articles split into four subsets. Software is mentioned within 2703 (5.7%) of these sentences, with a total of 7128 annotations split between 3718 software mentions and 3410 metadata entities, which disambiguate to 884 distinct software. The remaining sentences further function as negative samples for training to improve Precision. A detailed overview of the software, software type, mention type, and metadata contained in the established data is provided in Table 3.2[9]. As described, the resource was iteratively updated, where later steps serve as quality control for prior steps. Therefore, the reported numbers differ from prior publications, and include fixes of annotation errors from previously published versions. The annotation goal formulated in Section 3.2, to reach the same order of magnitude in annotation as the NER standard benchmark CoNLL [281], was reached for the main annotation target software, while fewer data is available for more specific annotations. Here, it can be argued that recognition of these specific aspects is easier than a standard NER problem, as they are defined in the context of software. However, thorough analyses are required to investigate which tasks can be solved in practice based on the established dataset.

With respect to formal citation, 603 reference entries to in-text citations were annotated with citation type, which correspond to 594 citations (see Table 3.2) because one citation string can contain multiple references. Moreover, 28,903 references were annotated for citation types of interest as described in Section 3.2.3, with a reduction to 1,392 references in the first annotation step. In the second annotation step, 32 additional references referring directly to software, software websites, or software manuals were identified. A total of 205 direct software and software manual references

[9] During annotation of informal mentions it was not distinguished between URL, archive, and identifier, due to the iterative annotation process this distinction was only introduced for the final annotation steps concerning formal citation.

	BioNerDs	Softcite	Softcite	SoMeSci
			2.0	
	2016	2020	2023	2021
# Documents	85	†4971	†4971	‡1367
# Software	2625	4093	5134	3756
# Other	0	2541	3091	3481
Software Type; Mention Type	□	□	■	■
Developer; Version; URL	□	■	■	■
Citation; Alt Name; Extension; License	□	□	□	■
Entity Disambiguation	□	□	□	■
Entity Linking	□	□	□	■
Formal Citation (Type; Metadata; Quality)	□	□	□	■
Indirect Mention	□	□	■	□
Source Domain	BI	L,E	L,E	L
Source Year	2002–10	2000–10	2000–10	2001–20

Table 3.3: Comparison of the publicly available corpora for software mentions in scholarly articles. Domain: BI=Bioinformatics, L=Life Sciences, E=Economics; †=not all negative sentences were published; ‡=set includes full-texts, methods sections, and single sentences, see Section 3.2.1.

were annotated for completeness and representation quality.

A detailed comparison between available ground-truth resources and the resource established in this work is provided in Table 3.3. Note that the development between Softcite and SoMeSci was partly overlapping, with the first annotation of the described data being published before Softcite (in [251]), but the full annotation being published after (in [244]), with Softcite v2 being published most recently. From Table 3.3 it can be observed, that BioNerDS covers only basic information about software, while Softcite and SoMeSci include additional metadata. The main advantages of SoMeSci are the coverage of information required for ED, EL, formal citation, and contextual information, enabling analyses beyond the scope of other datasets. Further, its article selection can be considered as advantageous because newer articles and a broader article range are included. On the other hand, Softcite has the advantage of including not only Biomedicine, but also Economics, which allows deeper analyses into disciplinary differences, particularly, because the number of software per articles is already known to strongly differ between the disciplines, as outlined in Section 2.1.

With Softcite v2, Softcite and SoMeSci became more similar in their annotations, which motivated efforts to systematically investigate differences and similarities between the datasets. An analysis was performed in the scope of a Chan Zuckerberg Initiative (CZI) Hackathon [46] regarding the topic of software citation, and resulted in a detailed list of differences that need to be taken into account when merging the datasets [22]. It includes both, semantics differences, e.g., with respect to the annotation or programming languages, and differences in specific annotation decisions, e.g., the annotation of software name abbreviations, and can serve as an instruction for merging both datasets in future work. However, some information from both datasets would need to be omitted in the process or added by further annotations. Overall, considerable curation effort is required to unify both datasets with high data quality, which is out of the scope of this work.

Aside the information outlined in Table 3.3, the IAA between sets also differs, but cannot be directly compared. While an FScore of F=80% is reported for BioNerDs, an overlap of 76% is reported for Softcite, followed by additional expert review of annotations. This step can, however, only improve Precision and not the Recall of the original annotations. In this work, the annotation was split into several steps to reduce the complexity of the annotation task with individual evaluation at each step, further impeding a direct comparison.

3.3.1 Analysis of Software Citation

Large-scale statements about software usage over time or software usage between disciplines are not possible based on the manual analyses because too few samples are available. However, the analysis allows to gain first insights on software citation that can be further explored and corroborated in a large-scale analysis. Figure 3.7 shows the distribution of the publication time over all articles in SoMeSci. It can be seen that a broad range of articles from 2000 to 2020 is included, with most articles published from 2012 to 2018. Given that the overall number of scientific publications is rising, it should be expected that more articles from the year 2019 and 2020 are included in the dataset. However, the dataset was iteratively developed, with most articles sampled in 2019, and only inlcuding articles up to 2018, explaining the specific distribution shown in Figure 3.7. Since SoMeSci consists of four distinct subsets differing in sample selection and annotation, the distribution of software mentions between the subsets is illustrated in Figure 3.8. *PLoS_m* contains the most mentions with 1563 (42%), followed by *PLoS_s* (25.7%), *Creation_s* 783 (21.1%), and *PubMed_f* 415 (11.2%).

A central aspect regarding software in science, which has been explored in most related work on software in science (see Section 2.1), is how often software is overall applied in research. Figure 3.9 summarizes both, the relative amount of articles using software, and the number of distinct software used within articles using software, split between the SoMeSci subsets. An aggregated analysis would not make sense at this point, as *PLoS_s* and *Creation_s* would add a clear selection bias, which can also be observed from the plot. It can be observed that there is a difference in the number of articles mentioning software between *PLoS_m* and *PubMed_f* with 80.8% to 57%, respectively. There is also a difference in the number of software mentioned, with a higher number observed in *PubMed_f* with 4.8 software per article compared to 2.85 software per article in *PLoS_m*. The higher number of software could be explained by the annotation methods, which only considered methods sections in *PLoS_m*, while the different ratio of software articles might suggest journal or discipline specific differences which require further exploration in large-scale analyses.

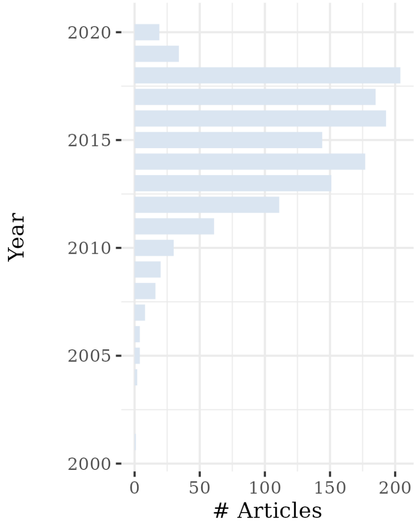


Figure 3.7: Distribution of publication year over all annotated articles.

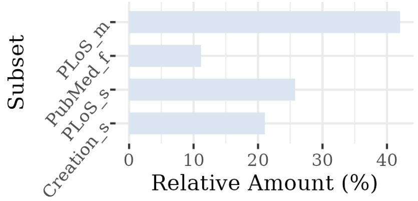


Figure 3.8: Relative amount of software mentions per subset.

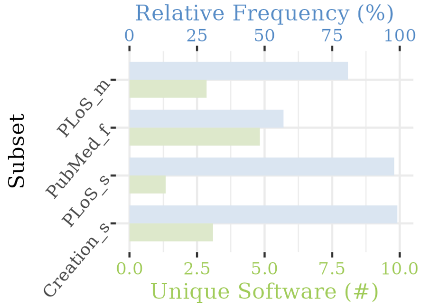


Figure 3.9: Amount of articles mentioning software (blue) and average number of different software in articles with software (green) per subset.

Another aspect of interest is whether software is cited without being mentioned in-text. As described above, 32 references were identified in the annotation where this is potentially the case, while further manual analysis showed that only 14 truly reflect the case of interest while the rest is due to the annotation process (e.g., references contain software in non-annotated parts of *PLoS_m*) or errors (e.g., mixed up references within articles). The 14 cases consist of seven direct citations, four manuals and three websites. In relative numbers this amounts to 8.5% of direct citations for a total of 82 direct citations in *PLoS_m* and *PubMed fulltexts*, 13.8% of 29 manuals, and 37.5% of 8 websites (details on formal reference types are provided in Section 3.3.2). Note that the sample size for manuals and websites is quite small and that all additionally identified websites result from the same article. The manual analyses of the underlying citation practices showed that in four cases generic terms were mentioned instead of the software name (e.g., "processing was done with [19]", where [19] is the software citation), in seven cases the use of a software was not indicated, and in three cases knowledge from the software was cited, e.g., the FAQ of the software instead of the software itself. Overall, the trend can be considered negligible since only a fraction of software is formally cited, and a further fraction is cited by direct citations and citations to manuals, as described in Section 3.3.2.

The cumulative distribution of software mentions by distinct software is illustrated in Figure 3.10. As before, it only makes sense to include *PLoS_m* and *PubMed_f* due to the selection bias in the other two subsets. It can be observed that the distribution is highly skewed towards few distinct software which account for a high number of overall mentions, while a long tail of rarely mentioned software exists. In the given plot the long tail distribution becomes linear after about 50% of distinct software, with all remaining software being mentioned only once. Moreover, it can be observed that the skewedness of the plot increases with the number of considered software mentions from *PubMed_f* over *PLoS_m* to their combination. In general, it can be expected that the long tail of rarely mentioned software grows beyond the software covered in SoMeSci, while the often mentioned general purpose tools stay the same. Therefore, it can be assumed that the distribution shifts further, making large-scale analyses necessary to reliably estimate the distribution of software in science.

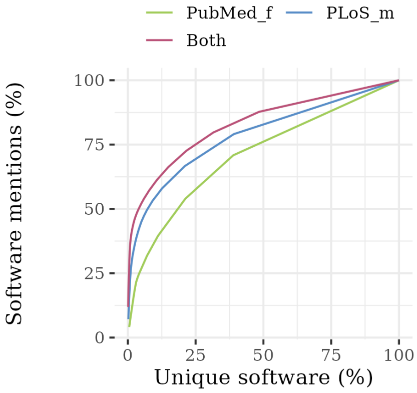


Figure 3.10: Cumulative distribution of software mentions.

The 884 distinct software contained in SoMeSci can further be distinguished by software type, with the results illustrated in Figure 3.11, split between number of software and number of mentions. In total, 636 (71.9%) of identified software are *Applications*, followed by 214 (24.2%) *PlugIn*, 24 (2.7%) *PEs*, and 10 (1.1%) *OSs*. The numbers differ strongly when considering the relative amount of mentions, with 72.6% referring to *Applications*, 11.4% to *PlugIns*, 11.3% to *PEs*, and 4.7% to *OSs*. This suggests that the distributions differ between software types. Particularly, few *PEs* and *OSs* are used quite commonly, while a large number of different *PlugIns* is used but each mentioned only few times. Since the number for all types except *Application* is considerably low, further large-scale analyses are required to corroborate these

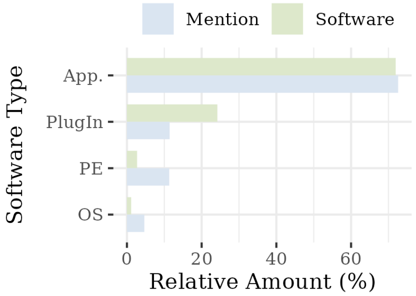


Figure 3.11: Relative amount of software types in terms of unique software and mentions.

findings. It was also found that the distribution of software types differs between the individual sets, with *Creation_s* containing more *PEs* and *OSs*, which is expected as they describe the development of software. Further, *PLoS_s* contains fewer *PlugIns*, which is also expected due to the article selection approach.

The identity annotation mapped all 884 distinct software to unique identifiers based on Wikidata, package repositories, Github, and general websites, in the given priority. The resulting ratio of link locations is summarized in Figure 3.12. Overall, 23.2% of unique software, covering 59.9% of mentions, could be linked to Wikidata, while package repositories (CRAN, Bioconductor, and PyPi) include 9.6% of distinct software covering 3.9% of mentions. Github further covers 10.2% of software, which only accounts for 6.9% of overall mentions. The remaining software was annotated for general websites, but can be further distinguished between other package repositories (Zenodo, BitBucket, Sourceforge) covering 3.9% of software and 2.8% of mentions, other websites covering 50.6% of software and 25.6% of mentions, and software that could not be identified covering 2.6% of software and 1% of mentions.

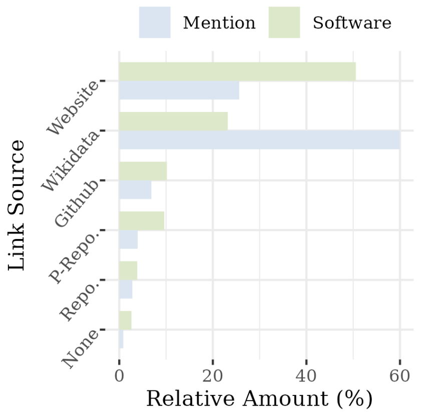


Figure 3.12: Relative amount of identity annotation targets for all annotated software. Repo: repository; P-Repo: package repository.

An overview of the ten most common software in SoMeSci is provided in Figure 3.13, taking into account the ratio of articles in which they are mentioned and the amount of overall software mentions they cover. As before, only *PLoS_m* and *PubMed_f* are considered here. The top ten software were analyzed concerning their absolute mention number and the amount of articles in which they are mentioned. *SPSS* is the most commonly mentioned software, contained in 8.2% of articles and covering 6.5% of all software mentions, followed by *R* and *Stata* which are mentioned in 4.6% and 4.5% of articles, respectively. Overall, eight out of the ten most common software are general purpose statistical software, with only *Windows* and *ImageJ* building the exception as an *OS* and an image analysis software. Further, the difference in software spelling was analyzed for the most common software, since the most data is available for them. The results are summarized in Table 3.4, and show that multiple names exist for all considered software. For some, such as *ImageJ* or *Stata*, only minor differences are found in the names, while strong differences are present for other software, such as *SPSS* or *R*.

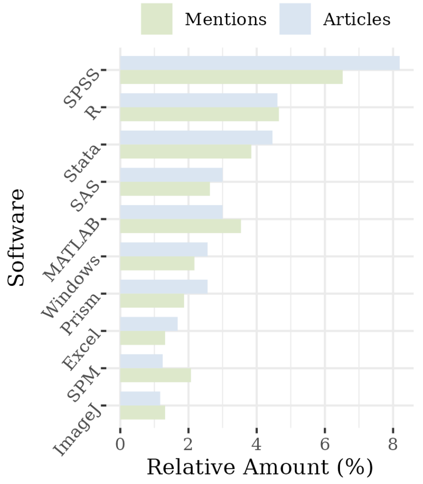


Figure 3.13: Most common software in terms of article mentions and relative amount of overall mentions.

3.3.2 Citation completeness

The citation completeness refers to the amount of metadata that is provided with formal software citations, required to capture provenance information of the investigation and to provide attribution for its development. As described in Section 1.4, the completeness can be considered along the dimension of identification of the (1) software, (2) developer, and (3) codebase. In the following,

Software	# Spellings	Top five Spellings
SPSS	12	SPSS, SPSS Statistics, Statistical Package for the Social Sciences, Statistical Package for Social Sciences, PASW Statistics
R	2	R, GNU R
Stata	2	Stata, STATA
MATLAB	3	MATLAB, Matlab, MatLab
SAS	3	SAS, Statistical Analysis System, SAS Enterprise Guide
Prism	4	Prism, GraphPad, PRISM, prism
Windows	2	Windows, windows
Excel	5	Excel, EXCEL, Office Excel, excel, Microsoft Office Excel
SPM	3	SPM, Statistical Parametric Mapping, statistical parametric mapping
ImageJ	3	ImageJ, Image J, Image-J

Table 3.4: Overview of spelling variations for the most common software.

completeness is first considered for informal software citation, then for formal citation, and finally with respect to their combination.[10]

Informal Citation Completeness The results for citation completeness for informal software mentions are summarized in Figure 3.14, with all Confidence Intervals (CIs) provided in the scope of this work being calculated as described in Appendix Section B. As before, only *PLoS_m* and *PubMed_f* are included. In total, 31% (CI_{95}=[28.5, 33.4]) of in-text software mentions include a formal citation to a bibliographic reference, which can provide additional information about the software. Information on the software developer is provided in 30.7% (CI_{95}=[28.2, 33.1]) of cases, while the version is indicated with the highest ratio of 46.6% (CI_{95}=[44, 49.2]). URLs are only provided sparsely in 10.1% (CI_{95}=[8.5, 11.6]) of cases, while extensions and alternative names are mentioned the least, with respective 1.8% (CI_{95}=[1.1, 2.5]) and 3.6% (CI_{95}=[2.6, 4.5]).

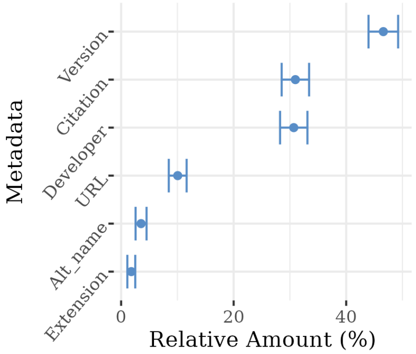


Figure 3.14: Informal software citation completeness.

The analysis is further extended to explicitly represent the identifiability of software, developer, and codebase. The results are summarized in Figure 3.15. Software is considered as precisely identifiable when either the developer is provided or when a formal citation is provided. Proper attribution is considered as given in the same cases. While formal citations are not always complete themselves, as outlined below, they usually provide sufficient information to identify the software in combination with the given name, and also provide attribution to the developer. Provided URLs are considered as conditionally identifiable because they are often not persistent or might point to the developer instead of the software. Overall, 60.8% (CI_{95}=[58.2, 63.4]) of software is precisely identifiable, 67.2% (CI_{95}=[64.7, 69.7]) is identifiable with uncertainty considering URLs, and codebase identification is given for 33.4% (CI_{95}=[30.9, 35.9]), when the version is stated and the software itself is identifiable.

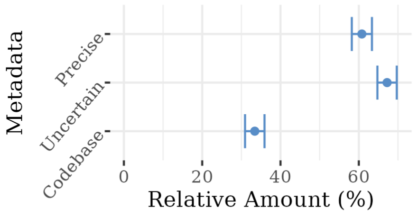


Figure 3.15: Informal software citation completeness with respect to software identifiability.

[10] Releases are included as versions in all following analyses, because they are rare in informal mentions.

The criterion used above describes a theoretic criterion of when software is considered identifiable. The same is true for codebase identification as it formally requires identifiable software, while an upper bound for codebase identification is given by the number of provided versions. However, in practice, cases exists where software is identifiable from an article based on the name alone, e.g., when SPSS is mentioned in the context of statistical analyses. However, more information is required in general due to problems such as software ambiguities (see Section 3.1.3) and legacy software. As described in Section 2.1, Howison and Bullard [118] analyze how much software can be identified by manual web search based on information provided in scientific publications.

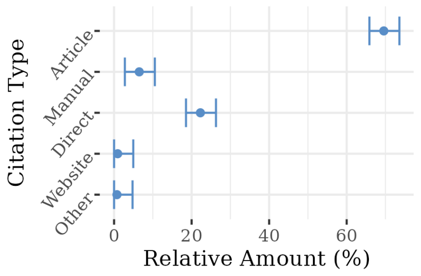


Figure 3.16: Distribution of formal citation types.

Formal Citation Completeness In mention contexts in which a formal citation was provided, the bibliographic reference can provide further information about the mentioned software. However, not all references do directly refer to software, as described in Section 3.1.4, with the distribution of the introduced resource types illustrated in Figure 3.16. The results show that most references are software articles in 375 or 69.6% (CI_{95}=[65.9, 73.6]) of cases, followed by direct citations in 120 or 22.3% (CI_{95}=[18.6, 26.3]), and manuals in 35 or 6.5% (CI_{95}=[2.8, 10.5]). Websites and other references were only found in 5 or .9% (CI_{95}=[0, 5]) and 4 or .7% (CI_{95}=[0, 4.8]) of cases, respectively. Note that some annotations were excluded from the analyses: nine duplicate entries with one software cited twice in one article; 30 references contained in *Creation_s* since they might add a bias as authors developing new software might be more particular about software citation; 13 entries because the citation was unrelated to software, e.g., references were mixed in a publication; and twelve entries because the citation only refers to an application of a software.

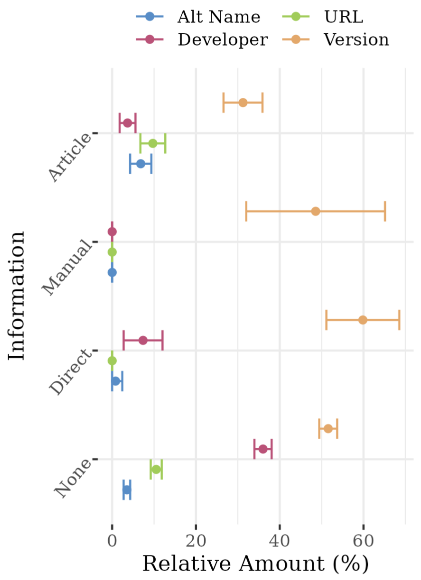


Figure 3.17: Informal citation completeness in dependence of the used formal citation type.

Not all citation types include all information necessary to fully describe used software, e.g., software article references do, in general, not capture software versions. Therefore, it is necessary to further investigate if authors systematically provide the missing information in the full-text document when they utitlize such citation types. For this purpose, the amount of informally provided metadata, in terms of versions, developers, URLs, and alternative names, is compared between the different citation types, including cases where software was not formally cited as an additional category. The citation types website and other are excluded because there are too few data points available for them. The results are illustrated in Figure 3.17 and show that less versions are provided informally when software articles are cited with 31.3% (CI_{95}=[26.7, 36.0]) compared to cases where software is not formally cited with 51.5% (CI_{95}=[49.4, 53.7]) and in comparison with mentions including direct citations with 59.8% (CI_{95}=[51.1, 68.5]). The difference is found to be significant by a χ^2 test, $\chi^2(1, N{=}2482){=}51.8$, $p<.001$, testing whether the number of informally provided versions systematically differs between not cited software and software cited by a software article, with an effect size of V=.15 estimated by Cramer's V. Further, developers were found to be less frequently informally mentioned in all cases where software is formally cited as compared to not formally cited software.

The informal mention of URLs is at a similar level between software cited with software articles and software that was not cited, while URLs have never been provided informally when software was cited directly or through a manual. A similar picture is found for alternative names.

The completeness of formal software citations was analyzed for 153 direct software citations—including 8 websites and 4 other citations as incomplete direct citations—and 44 manuals, as before, excluding 8 references from the *Creation_s* set. The corresponding results are summarized in Figure 3.18. Regarding the large sample of direct software citations, name, developer and publication year are commonly included in 146 or 94.8% (CI_{95}=[91.3, 98.3]), 141 or 91.6% (CI_{95}=[87.2, 95.9]), and 132 or 85.7% (CI_{95}=[80.2, 91.2]) of instances, respectively. Version, description, and URL are less commonly stated with 100 or 64.9% (CI_{95}=[57.4, 72.5]), 77 or 50% (CI_{95}=[42.1, 57.9]), and 62 or 40.3% (CI_{95}=[32.5, 48.0]), while the type of citation and date of access are only rarely provided in 44 or 28.6% (CI_{95}=[21.4, 35.7]) and 25 or 16.2% (CI_{95}=[10.4, 22.1]) of cases. Release date (3, 2% (CI_{95}=[0, 4.13])), identifier (1, .6% (CI_{95}=[0, 1.92])), and archive (1, .6% (CI_{95}=[0, 1.92])) were only sporadically found. Regarding manuals, most results are at a comparable level, with a notable difference in version, which are less often contained in citations to manuals with 25% (CI_{95}=[12.2, 37.8]).

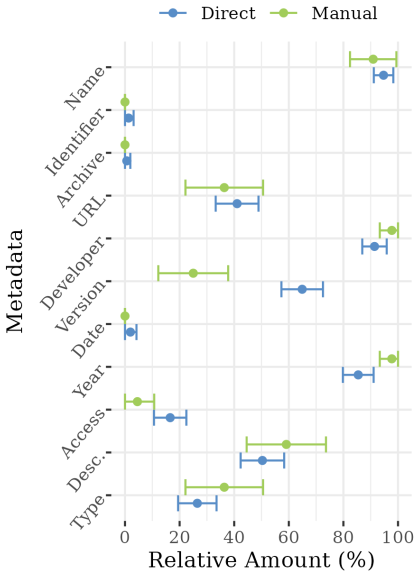


Figure 3.18: Amount of metadata provided in formal software citations. Desc: description.

The analysis was extended to directly cover the identification of software, developer, and codebase based on provided metadata. The corresponding results are illustrated in Figure 3.19. Regarding direct citation, software can be identified with high confidence in 132 or 87.4% (CI_{95}=[82.1, 92.7]) of cases based on the mention of name and developer, while archive and identifier are only stated in one case each and overlap with mentions of name and developer. The overall number of identifiable cases increases to 149 or 98.7% (CI_{95}=[96.9, 100]) when including cases with URLs, however, as described in Section 3.1 URLs can be subject to link rot. The exact codebases can be identified with high confidence in 101 or 65.6% (CI_{95}=[58.1, 73.1]) of cases, with versions being provided in 99 cases and release dates in three. Note that the software itself also has to be identifiable for codebase identification. Considering a date of access sufficient, under the assumptions that the newest available version at the date of access was used, the ratio increases to 113 or 73.4% (CI_{95}=[66.4, 80.4]). Regarding citations to manuals, the values for software identification and developer attribution are at equal levels, however, the value for codebase identification is lower with only 25% (CI_{95}=[12.2, 37.8]) and 27.3% (CI_{95}=[14.1, 40.4]) being identifiable with and without considering a date of access, respectively.

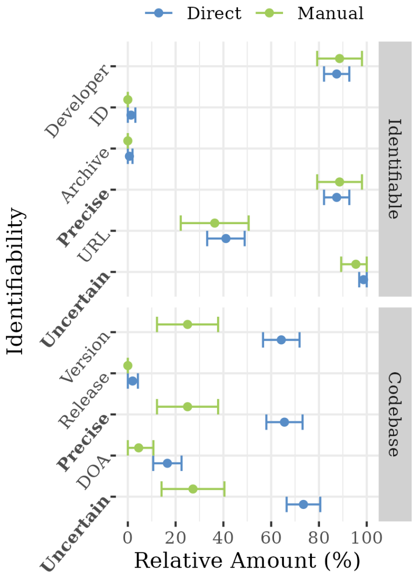


Figure 3.19: Relative citation completeness of direct software citations and software manuals with respect to identification, including 95% CIs. "precise": clearly identifiable; "uncertain": includes identification by a URL or date of access.

Combined Completeness The last analyzed aspect of citation completeness is whether informally and formally provided metadata complement each other when both are provided. Only the common metadata of version, developer, and URL are considered here, while all other metadata provided in formal citations does by definition complement informal citations. The name on the other hand, is by definition always given for informal mentions and, therefore, complements formal citations. The results are illustrated in Figure 3.20. Developers are only rarely provided when software is cited with a direct software citation in 6.9% (CI_{95}=[2.6, 11.3]) of cases and for manuals in 0 cases. All of the cases where a developer was mentioned in-text overlapped with cases where the developer was also provided in the formal references, therefore, not improving the overall coverage. Versions are provided quite often with the informal mention when software is cited with a direct citation with 58.5% (CI_{95}=[50, 66.9]) and 47.1% (CI_{95}=[30.3, 63.8]) for citations to manuals. Through aggregating over formal and informal information the overall coverage for versions improves up to 80% (CI_{95}=[73.1, 86.9]) for direct citations and to 55.9% (CI_{95}=[39.2, 72.6]) for manuals. URLs were never mentioned in the full-text document when software was formally cited by either a direct citation or a manual.

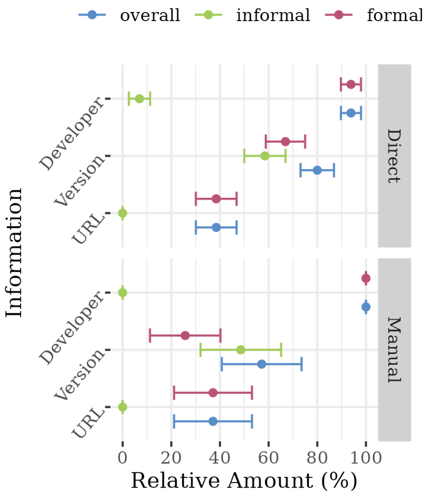


Figure 3.20: Citation completeness of direct software citations and software manuals for the combination of formal and informal metadata.

3.3.3 Data Quality of Literature Databases

The quality of database representation allows to assess whether existing infrastructure for representing bibliographic references can be applied for systematic analyses of software citation in science. It was evaluated on the same references as formal citation completeness, additionally including the references from *Creation_s*, because they do not add a bias in this analysis as only the representation of the information is investigated. As outlined in Section 3.2.3, the annotation covers the original publishers and the databases Semantic Scholar and Crossref, and contains information on whether the originally published information is covered, how it is structured, and whether it is correct. The quality of database representation was investigated individually for the considered metadata, and is illustrated in newly established, adapted alluvial plots described in Section 3.2.5.

As described in Section 3.2.3, multiple entries for one reference can exist in Semantic Scholar. Overall, 40 (24.8%) out of 161 represented citations have duplicate entries, with 33 (27% of 121) in direct citations and 7 (17.5% of 40) in manuals. For all following analyses the most complete entry, covering most relevant information, was selected.

```
PDF:
Boersma P, Weenink D. Praat: Doing phonetics by computer [Version 6.0.05, Computer program].
Retrieved Nov 8 2015.

Semantic Scholar:
{
    'title': 'Praat : doing phonetics by computer (version 4.4.24)', 'year': 2006, ,
    ...
    'authors': ['authorId': '145317765', 'name': 'P. Boersma']
}
```

Listing 15: Original PDF reference information and the corresponding Semantic Scholar reference entry for *Direct Software Citation* in the presented dataset [ID: PMC5574543, Semantic Scholar ID: 27589464, [312]]. Several errors are present in Semantic Scholar: information on date of access and citation type are lost, the content for version is wrong, author information was partially lost and wrong information on the publication year is added. Metadata is highlighted for: developer (🟨), name (🟦), description (🟧), version (🟩), type of citation (🟫), and date of access (🟪). Metadata represented in an unstructured manner (⬚), with incomplete (⬚) information, or errors (□) is marked. Note that the version in Semantic Scholar is both unstructured and wrong.

General Analysis

Missing entries for references were identified in Crossref and Semantic Scholar, where it is necessary to distinguish two cases of missing references: entire articles missing and individual references missing. References that are missing because the entire article is not contained in a database are ignored because this is a problem of overall coverage and does not provide any information about the quality of software citation representation. However, individual references missing, even so an article is represented, can point to a problem regarding software citation representation and needs to be investigated. Overall, 36 of 157, 22.9% (CI_{95}=[16.4, 29.5]) of direct citations and 5 of 45, 11.1% (CI_{95}=[1.93, 20.3]) of manual references were individually missing from Semantic Scholar. In Crossref, 8 of 155, 5.2% (CI_{95}=[1.7, 8.6]) of direct citations and 6 of 47, 12.8% (CI_{95}=[3.2, 22.3]) of manuals were missing[11]. To investigate if this is a systematic bias concerning software citations, the amount of software articles missing from the databases was also investigated, serving as a sample of regularly published articles, with 4 of 382, 1.1% (CI_{95}=[0, 2.1]) and 1 of 381, .3% (CI_{95}=[0, 1]) of software articles missing from Semantic Scholar and Crossref, respectively. To test if the amount of missing references differs between software articles and direct software citation a χ^2 test was employed for each of the databases, with Semantic Scholar $\chi^2(1, N{=}539){=}74.4$, $p{<}.001$ and Crossref $\chi^2(1, N{=}536){=}13.2$, $p{<}.001$ with respective effect sizes of V=.38 and V=.17, estimated by Cramer's V. Manuals are not further tested because fewer data is available and statements would be less reliable.

During annotation it was further observed that Semantic Scholar sometimes adds wrong information without relation to the original reference (see Listing 15), and that information is sometimes duplicated in a wrong location, e.g., the software name is represented as both title and publication venue. In total, wrong information is added to reference representations of direct citations in 19, 15.7% (CI_{95}=[9.2, 22.2]) and for manuals in 25, 62.5% (CI_{95}=[47.55, 77.5]), while duplicate information is added in 26, 21.5% of direct citations and 2, 5% of manuals, with 3 cases overlap. Both problems were not observed for Crossref.

[11] Note that the overall number of references differs between databases due to missing articles.

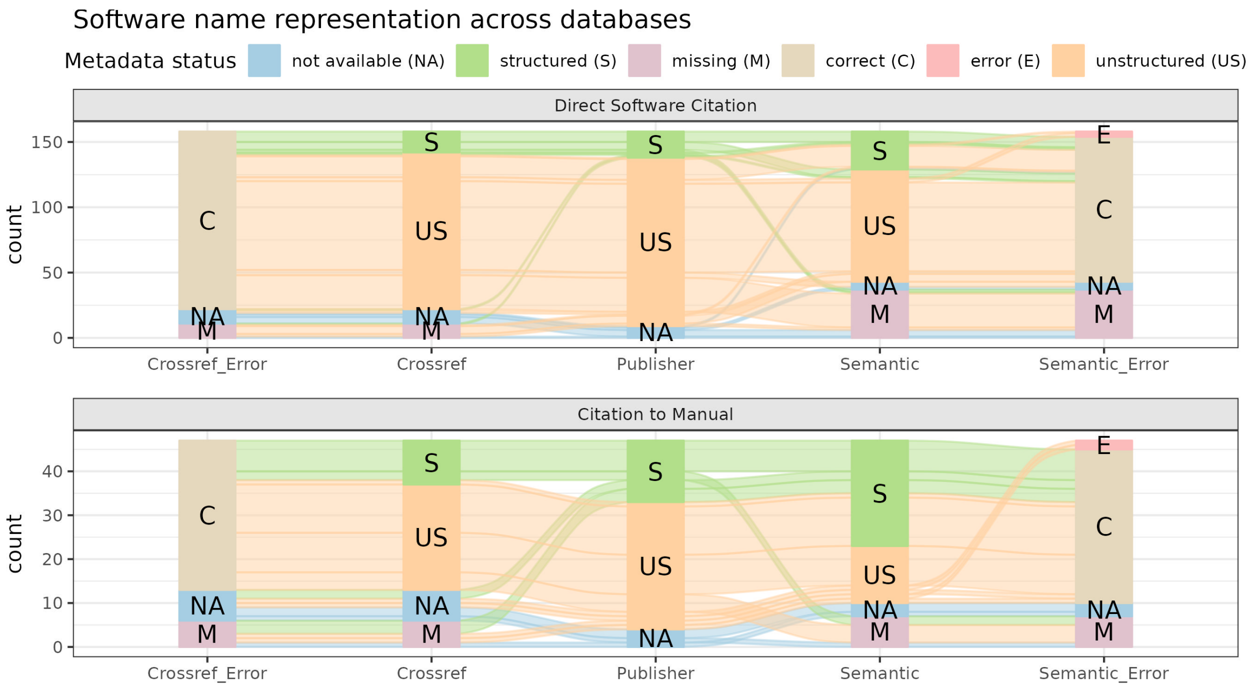


Figure 3.21: Adapted alluvial plot illustrating data quality of software name representation in terms of availability, structure, and correctness within direct software citations and citations to manuals. The results for availability and structure for the publisher's representation in the middle column are compared to the results for Crossref on the left and Semantic Scholar on the right. The outermost columns show the correctness (Error) for Crossref on the left and Semantic Scholar on the right. Missing samples in Crossref and Semantic Scholar are added for comparability between columns. The illustration follows the principle illustrated in Figure 3.6.

Software Name

The results for database data quality of software names are given in Figure 3.21, and a further summary of the results is provided in Appendix Table A2. The software name is commonly included by publishers in both direct citation (149 of 158, 94.3%) and citations to manuals (43 of 47, 91.5%), with only some information represented in a structured manner with 20 (12.7%) direct citations and 14 (29.8%) manuals. Crossref and Semantic Scholar only lose information on software names in rare cases with 3 of 147 (2%) and 4 of 121 (3.3%), respectively, for direct citations, and 4 of 41 (9.8%) and no losses for manuals. In turn, information is added by Semantic Scholar in 1 (.8%) direct citation. Semantic Scholar manages to increase the ratio of structured information for direct citations by 11.3 *pp* to a value of 24% (29 of 121) and for manuals by 30.2 *pp* to 60% (24 of 40), while Crossref directly reflects publisher structure, when information is not lost[12]. Notably, Semantic Scholar does not retain structure for all direct software references, but instead loses structure for 7 (5.8%), and adds structure for 18 (14.9%) references. Regarding manuals, Semantic Scholar does not lose structure, but adds it in 12 (30%) cases. All information on software names contained in Crossref are correct, while Semantic Scholar introduces a small amount of errors in both direct citations (4 of 115, 3.5%) and manuals (2 of 37, 5.4%[13]). Regarding manuals, all errors are due to misrepresentation of software names as other information, while for direct citations 75% of errors

[12] The results are always reported excluding entirely missing references (M).
[13] The amount of errors is always reported excluding not covered information (NA).

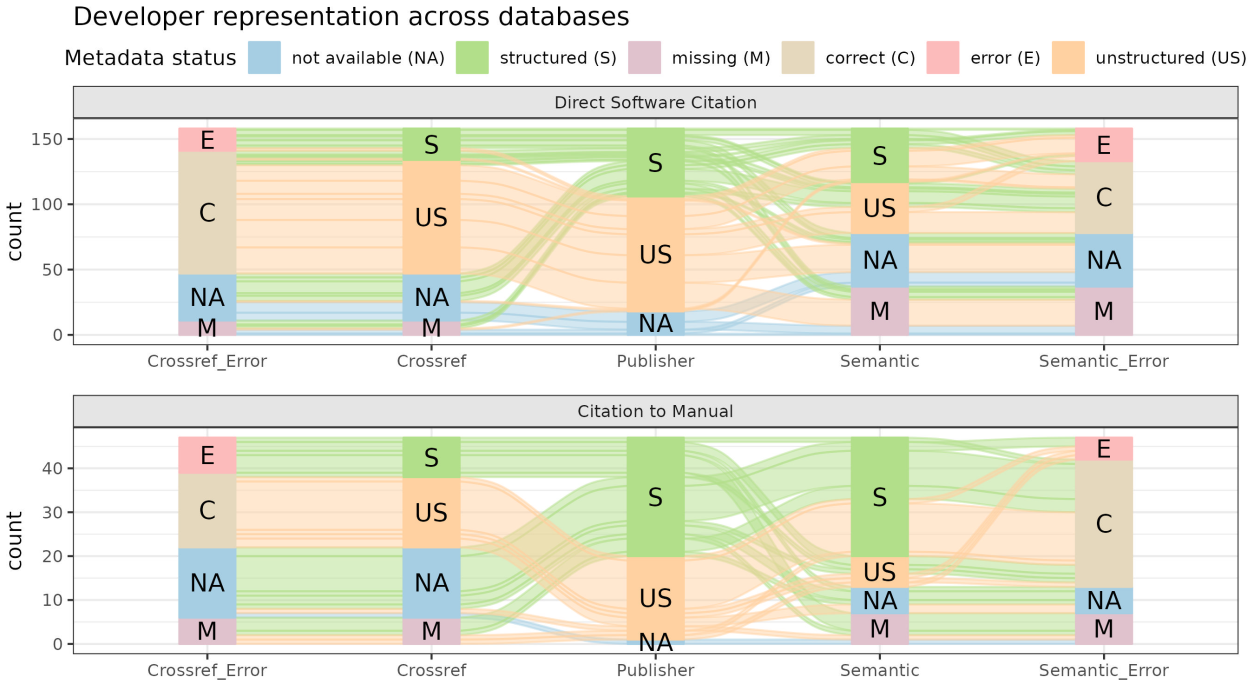


Figure 3.22: Adapted alluvial plot illustrating data quality of software developer representation in terms of availability, structure, and correctness within direct software citations and citations to manuals, following the principle of Figure 3.6.

are due to misrepresentation and 25% due to wrong information.

Developer

The results for database data quality of software developers are given in Figure 3.22, and a further summary of the results is provided in Appendix Table A2. The software developer is commonly included by publishers in both direct citations (140 of 158, 88.6%) and citations to manuals (46 of 47, 97.9%). It is structured in a majority of 27 (57.4%) manuals but less frequently for direct citations with a value of 52 (32.9%). Crossref and Semantic Scholar both lose information on software developer in a notable amount of cases for direct citations (22 of 147, 15% and 30 of 121, 24.8%, respectively) and manuals (15 of 41, 36.6% and 6 of 40, 15%). Semantic Scholar manages to increase the ratio of structured information slightly for direct citations by 1 *pp* to a value of 33.9% (41 of 121) and strongly for manuals by 10.1 *pp* up to 67.5% (27 of 40), with structured samples outweighing unstructured samples. Same as for software name, Crossref mostly reflects publisher structure for developers, when information is not lost. Again, Semantic Scholar does not retain structure for all direct citations, but instead loses structure for 19 (15.7%), and adds structure for 26 (21.5%) references. Regarding manuals, Semantic Scholar also loses structure for 4 (10%) but adds it in 13 (32.5%) references. Semantic Scholar introduces a notable amount of errors in both direct citations (25 of 80, 31.2%) and manuals (5 of 34, 14.7%). For direct citations they are distributed between wrong information (44%), incomplete entries (32%), and misrepresentation (28%)[14], and for manuals between misrepresentation (80%) and incomplete entries (20%). Crossref also introduces a notable amount of errors in both direct citations (17 of 111, 15.3%) and manual (8 of 25, 32%). In Crossref almost all errors (direct citation 94.1%, manual 100%) are due to incomplete

[14] Errors are not exclusive, therefore, the percentages do not necessarily sum to 100%.

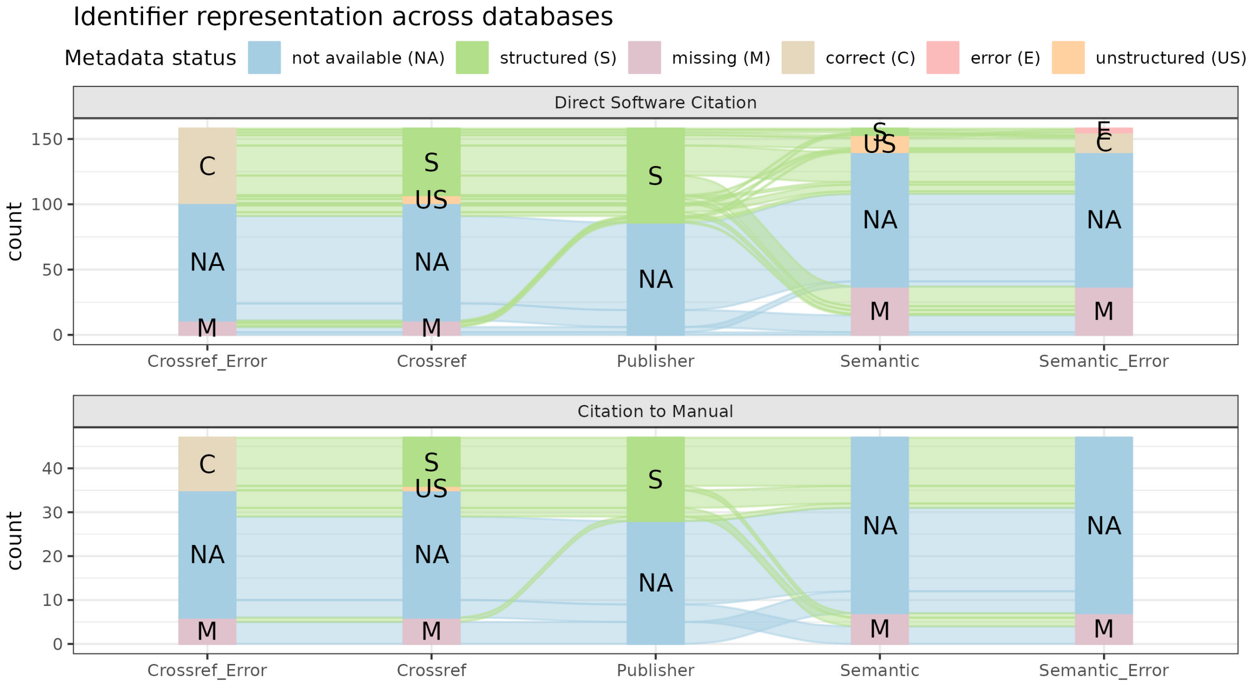


Figure 3.23: Adapted alluvial plot illustrating data quality of software identifier representation in terms of availability, structure, and correctness within direct software citations and citations to manuals, following the principle of Figure 3.6.

entries because Crossref only includes the first author when representing article references. This is no problem with respect to scientific articles since the full author information can be gathered from the article entry corresponding to the reference. However, software citations generally do not have a corresponding entry from which the information can be gathered, which currently leads to a loss of information regarding software developers for direct software citations and software manuals.

Identifier

Information on the publication venue of a software is analyzed as a combination of ID, Archive Link, and URL, where the most relevant information is chosen in the given order if available[15]. The results for database data quality of software identifiers are given in Figure 3.23, and a further summary of the results is provided in Appendix Table A2. A software identifier is included by the publisher in almost half of all references in both direct citation (72 of 158, 45.6%) and citations to manuals (19 of 47, 40.4%), always in a structured manner. Semantic Scholar loses information about identifiers in a high amount of 32 of 121 (26.4%) direct citations and always loses it for manuals. Crossref loses information less frequently with 10 of 147 (6.8%) direct citations and 6 of 41 (14.6%) citations to manuals. Further, Semantic Scholar loses structure for 13 of 121 (10.7%) direct citations, while Crossref loses structure for 6 of 147 (4.1%) direct citations and 1 (2.4%) citation to a manual. Errors are only present in rare cases concerning Semantic Scholar and direct citations, affecting 3 of 18 (16.7%) covered references. The errors are due to misrepresentation in 33.3% of cases and wrong information in 66.7%.

[15] All metadata is semantically related and samples for ID and Archive are too rare to analyze individually.

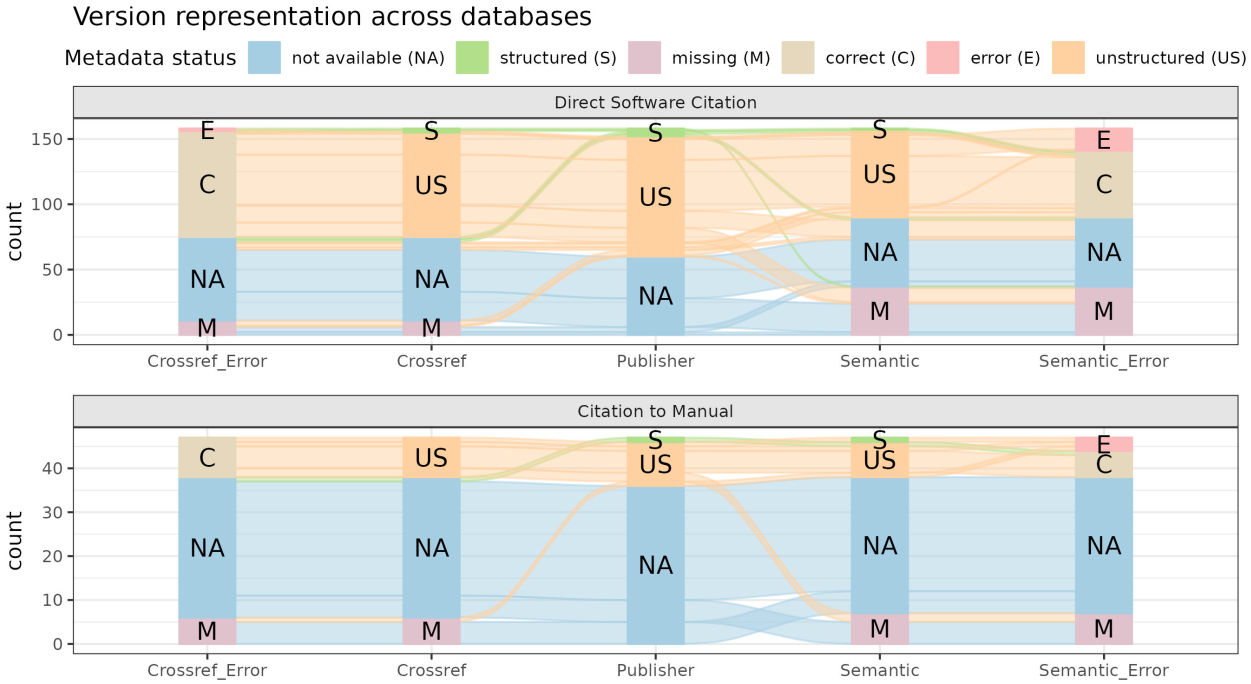


Figure 3.24: Adapted alluvial plot illustrating data quality of software version representation in terms of availability, structure, and correctness within direct software citations and citations to manuals, following the principle of Figure 3.6.

Version

The results for database data quality of software versions are given in Figure 3.24, and a further summary of the results is provided in Appendix Table A2. The software version is commonly included by publishers in direct citations (98 of 158, 62%), but less frequently in citations to manuals (11 of 47, 23.4%). For both citation types, versions are rarely represented in a structured manner with 6 (3.8%) in direct citations and 1 (2.1%) in manuals. Crossref rarely loses information on software versions in 10 of 147 (6.8%) direct and 1 of 41 (2.4%) citation to manuals. Semantic Scholar, on the other hand, loses version information in a considerable amount of direct citations (17 of 121, 14%), but never in manuals. Crossref does not lose structure information when versions are represented, but adds structure for 1 of 147 (.7%) direct citations. Semantic Scholar loses structure for 2 of 121 (1.7%) direct citations and 1 of 40 (2.5%) manuals. No errors are present in Crossref for manuals and only few (2 of 83, 2.4%) for direct citations, all due to misrepresentation of the version as other information. For Semantic Scholar a notable amount of errors is present in direct citations (17 of 68, 25%) and manuals (3 of 9, 33.3%). The errors in Semantic Scholar for direct citation are mainly due to wrong information (76.5%), followed by incomplete information (17.6%), and misrepresentation (5.9%), while for manuals they are due to incomplete information (100%) and misrepresentation (66.7%).

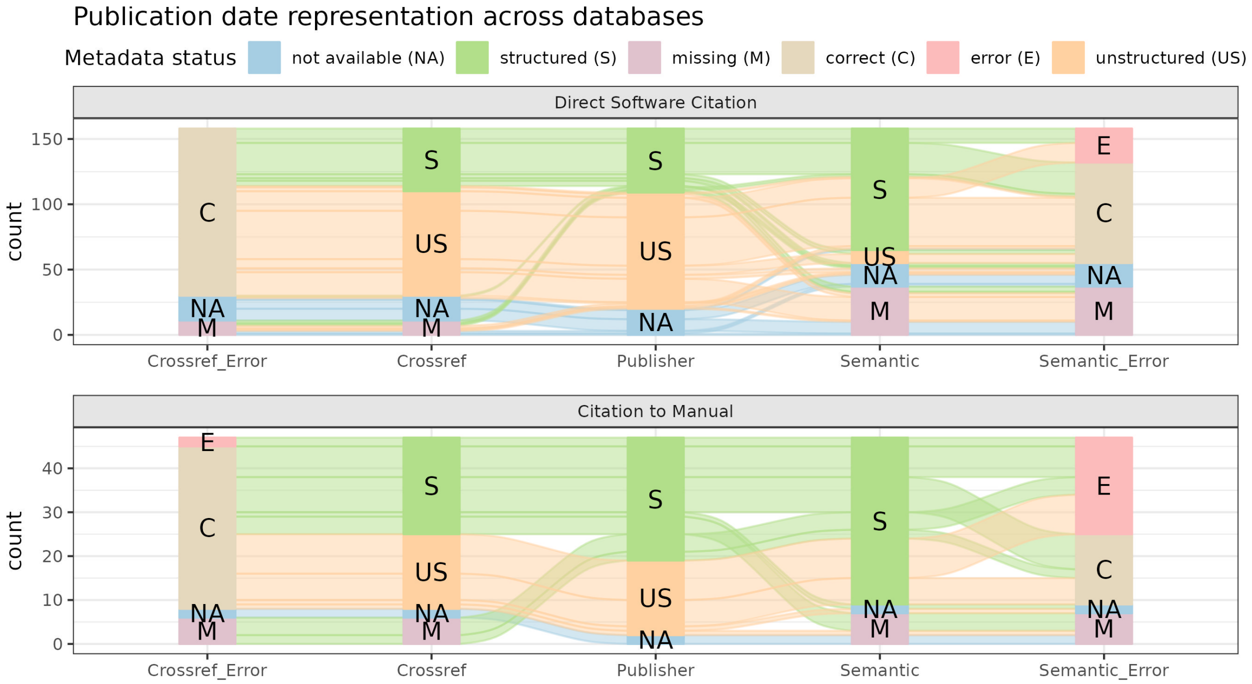


Figure 3.25: Adapted alluvial plot illustrating data quality of software release date representation in terms of availability, structure, and correctness within direct software citations and citations to manuals, following the principle of Figure 3.6.

Release Dates

Same as for the identifier, information on release dates is summarized for analyses, where the release date, date of access, and publication year are prioritized for analyses in the given order to always select the most complete information. The results for database data quality of release date information are given in Figure 3.25, and a further summary of the results is provided in Appendix Table A2. The publication date is commonly included by publishers in both direct citation (138 of 158, 87.3%) and citations to manuals (45 of 47, 95.7%). It is often represented in a structured manner for manuals (28, 59.6%) but less frequently for direct citations (49, 31%). Crossref and Semantic Scholar only lose information on publication date in few references with 2 of 147 (1.4%) and 9 of 121 (7.4%) direct citations, respectively, and 0 of 41 and 2 of 40 (5%) citations to manuals. In turn, information is added by Semantic Scholar for 1 (.8%) direct citation. Semantic Scholar manages to strongly increase the ratio of structured information, for both direct citations by 45.9 *pp* (93 of 121, 76.9%) and citations to manuals by 35.4 *pp* (38 of 40, 95%), while Crossref mainly reflects the structure of the publisher. Notably, Semantic Scholar retains structure in all cases except for 3 (2.5%) direct citations, and adds structure for 55 (45.5%) direct citations and 15 (37.5%) manuals, while Crossref adds structure for 4 (2.7%) direct citations. A high number of errors is present in Semantic Scholar for direct citations (26 of 103, 25.2%) and manuals (22 of 38, 57.9%). For Crossref, few errors are present in manuals (2 of 39, 5.1%), all due to incomplete information. Errors in Semantic Scholar for direct citation are distributed between wrong information (88.5%), incomplete information (7.7%), and misrepresentation (3.8%), and for manuals between wrong information (90.9%) and incomplete information (9.1%).

Description and Type of Citation

Description and type of citation are considered less crucial to software citation as the information discussed so far. Therefore, only the results are summarized with the corresponding alluvial plots available in Appendix Figures A1 and A2. A software description is included by publishers in about half of direct citations (78 of 158, 49.4%) and citations to manuals (27 of 47, 57.4%), with only some information represented in a structured manner with 13 (8.2%) direct citations and 15 (31.9%) citations to manuals. Crossref and Semantic Scholar only lose information on software descriptions in rare cases for manuals with 1 of 41 (2.4%) and 2 of 40 (5%), respectively. Semantic Scholar manages to increase the ratio of structured information, for both direct citations by 10 *pp* (22 of 121, 18.2%) and citations to manuals by 23.1 *pp* (22 of 40, 55%), while Crossref directly reflects publisher structure, when information is not lost. Errors are rare for descriptions and only appear in 2 of 67 (3%) direct citations in Semantic Scholar and 1 of 75 (1.3%) in Crossref.

The type of citation is commonly not included in both direct citation (42 of 158, 26.6%) and citations to manuals (16 of 47, 34%), and never represented in a structured manner. Information is in some cases lost for direct citations (Semantic Scholar: 11 of 121, 9.1%; Crossref: 5 of 147, 3.4%) and manuals (Semantic Scholar: 1 of 40, 2.5%; Crossref: 3 of 41, 7.3%). Structure is mainly represented as by the publisher for both databases, with Semantic Scholar adding structure in 1 of 121 (.8%) direct citations. Errors appear in 3 of 19 (15.8%) direct citations within Semantic Scholar and 1 of 35 (2.9%) in Crossref.

3.4 Limitations

While the annotation quality of SoMeSci was ensured by high IAA and duplicate annotation, errors are still present in the dataset. Some errors of previous annotation steps were later identified through the iterative annotation process, however, this approach can only improve Precision and not increase the Recall, as only the context of annotated software is re-examined. Throughout the annotation process, several annotation targets proved to be challenging for human annotators. This concerns the annotation of contextual information, i.e., software and mention type. Here, a challenging aspect was distinguishing between *Applications* and *PlugIns*. Similarly, the distinction between mentions of algorithms and software proved difficult, as authors tend to name software based on the implemented algorithm. Software identity annotation also proved challenging due to high variance in annotation targets, i.e., distinguishing standard and canonical links, identification of old software without valid persistent identifier, or accessing pay-walled information. It was found that external knowledge is required for identification of software identities by human annotators. Therefore, it should also be considered to include external knowledge in automatic ED approaches. Moreover, annotation of formal citation quality proved challenging as it adds the dimension of representation errors. Finally, cases were observed (<10) where software and provided metadata stood in obvious conflict, e.g., the provided developer did not reflect the actual developer. This information was annotated to represent the author's intent to provide additional information for the software. This shows that errors are made by authors regarding the citation of scientific software.

In general, the data selection for SoMeSci can lead to a bias in the resulting dataset. SoMeSci is based on the PMC OA subset, leading to a selection bias towards Life Sciences, while previous work has shown that there is a difference in software usage between scientific disciplines (see Section 2.1). Different disciplines are likely to use different software based on specific requirements, research methods, and specialized tool-support, e.g., ML frameworks applied in Computer Science are less likely to be used in biomedical research. Dependent on the use case and type of software, it might also be mentioned in different contexts impeding generalization to other research domains. It is, for instance, possible that the use of archive links is more common in disciplines that frequently reuse and adapt source code, e.g., in Computer Science. As all included articles stem from PMC OA, the data selection might also exhibit a bias towards open-access publications. Researchers choosing to publish under open-access might also be supporters of the open data movement and, therefore, have a better awareness for attributing other open work such as software. Furthermore, the majority of articles included in the dataset was published between 2012 and 2018, with the overall dataset spanning a range from 2000 to 2020. It can, therefore, only make statements about software citation in the examined time span, and the amount of available data is not sufficient to analyse trends over time. A benefit of the given article selection, on the other hand, is that bibliographic data providers had sufficient time to react and represent the formal software citations given within the articles, making the examined dataset well suited to assess the representation quality of formal software citation.

Another selection bias concerns scientific publishers because a large amount of articles is published by PLoS. This specifically affects the analysis on database accuracy, where the data representation of scientific publishers is investigated. Here, it is argued that PLoS is a representative choice regarding the handling of software citation because PLoS has a high interest in software. PLoS ONE, the largest journal published by PLoS by a considerable margin, allows software submission and publishes corresponding software articles, but also encourages proper software mention (see Section 1.3). Moreover, the same general trends with the same major issues from both the publisher side and the bibliographic databases are also observed when articles published by PLoS are excluded from the analysis, however, based on a notably smaller sample size. This highlights that the identified issues of formal software citation representation are not specific to one publisher but exist broadly across the current bibliographic infrastructure. However, it should be noted that specific publishers might already handle software citation in a suited manner.

The dataset might further contain a bias towards specific article sections and overrepresent software mentions. Since the annotation process was expensive, a trade-off between annotating sufficient data and available time for the annotation had to be found. The dataset was created from four subsets with different data selection strategies, which led to an overall higher sample number and enables analyses such as *Creation* and *Deposition* classification. However, specific sub-datasets are likely to contain a higher average of software mentions per sentences, than would appear in full-text articles, which also concerns the largest set *PLoS_m* containing 42% of extracted software mentions. This means ML methods trained on it might learn to represent a wrong prior probability for software mentions, potentially creating more false positives when applied on full-text document articles. For this purpose it should be explicitly assessed if trained ML models overrepresent software, by applying systematic test cases during evaluation. Moreover, *PLoS_s* should, in general, not be included in any evaluation setting, since it strongly overrepresents software mentions, and would lead to an overestimation of performance, while *Creation_s* has to be considered during evaluation because it contains specialized mention types that are not present in other subsets. Here, a potential bias also has to be tested in a corresponding evaluation setting.

Despite the enrichment strategy, the frequencies for some types of entities are still low, for instance, regarding software deposition statements. This is argued to reflect the general rareness of those entities within articles of the represented domains. Metadata such as license or extension for which only few samples are available are still relevant for practical analyses, e.g., regarding software availability. Here, further tests are required to assess if the information can be recognized in practice. This might be possible even so the sample size is small, since the extraction of metadata is less complex than other NER problems, because they can only appear in combination with software citations.

Lastly, the performed annotation was restricted to named software, and did not include indirect mentions of software. The extent of this problem is partially investigated by analyzing the occurrence of software citation without informal mentions in the article text, showing that this practice is neglectable compared to the overall number of software mentions. However, further cases might exist where authors only refer to software by generic terms without ever stating the name of the software, e.g., by terms such as “source code”. These cases are only of interest regarding the applications outlined in Section 1.2 when additional information about the software was provided, since only then software can be properly disambiguated and represented for large-scale analyses. While these instances are not represented here, they are encoded in Softcite v2 [120] (see Table 3.3), and found to be uncommon accounting for ≈2% of annotated software.

3.5 Requirements for Large-scale Analysis

The ground-truth dataset SoMeSci for software in scientific publications was introduced. It covers several aspects of software citation for which no ground-truth data is available and that have only been superficially analyzed in existing literature such as software type, mention type, software identity, and formal software citations, with a detailed comparison outlined in Table 3.3. The corpus was created by manual annotation of 1367 articles from PMC OA in a broad range of publications in Life Sciences and related disciplines, where entities were linked to unique identifiers. High quality is underlying the manual annotation process, which was ensured by assessing the IAA at every annotation step. The dataset was created to serve as ground-truth data for developing automatic methods for software analysis, to perform preliminary analyses of software in science, and to assess the requirments for a subsequent large-scale analysis.

The annotation process underlying the creation of SoMeSci was designed to decrease annotation complexity by splitting the annotation task into multiple steps. While the benefit of this approach was not systematically documented, it was found that the re-iteration of existing annotations in subsequent annotations steps led to enhanced discussion between annotators, discovery of errors, and re-annotations. Particularly, ED by manual EL led to further research on software and served as an additional quality check for annotation decisions. Certain annotation targets and steps also proved to be challenging for annotators, which is taken as further indication that splitting the annotation and reducing the complexity benefited the overall annotation process.

First preliminary analyses of software in science revealed aspects that require further investigation in large-scale analyses. Particularly, the ratio of articles using software and the number of software per article were found to differ between the annotated subsets. This could be partly explained by the annotation methods, which only considered methods sections in one of the sets. However, this does not explain the ratio of articles using software that was identified as the trend points in the opposite direction. This means the underlying effect is more likely due to disciplinary or even journal specific differences in software usage, effects which have already been observed in prior work (see Section 2.1). Further, the distribution of software was found to be highly skewed towards few software with high mention numbers, while a long-tail of software exists that is only rarely mentioned. However, an analysis further showed that, due to the small sample size, the distribution cannot be fully assessed based on the given dataset, suggesting that the skew might be stronger than estimated. Moreover, the results showed that the distribution might differ between different types of software such as *PEs* and *PlugIns*. The results further show that the majority of the most common software is specialized on statistics allowing a broad application across scientific disciplines. Here, further analyses can provide background on discipline specific software. All outlined analyses are of preliminary nature and require further large-scale analyses to make reliable statements, particularly, regarding the following research questions:

- How is software usage distributed in scientific publications?
- Do disciplinary differences in software usage exist?
- How have software citation and its completeness changed over time?

The results of analyzing SoMeSci also provide valuable insights on software ED and the requirements to implement a large-scale software ED. Initially, the results highlight the importance of performing an ED because all common software were found to have multiple names, which makes their aggregation a necessity for reliable reasoning. Further, it shows that the same name can refer to different software adding an additional layer of complexity to software ED as it proves that disambiguation by software name alone can lead to errors. The results further show, that ED by EL is, in general, not possible in large-scale analyses due to the low coverage of software in general purpose knowledge bases, with only 23.2% of distinct software in SoMeSci covered. While general purpose software that is broadly applied in scientific research is covered by knowledge bases, specialized research software is mostly not. Moreover, EL against repositories such as Github or PyPi is also challenging due to the high number of contained software and the identified ambiguities in software names.

Regarding formal and informal software citation, the results indicate that informal citation is the dominant practice. While 31% of informal mentions were found to be additionally accompanied by a formal citation, the trend to cite software without informal citation was found to be negligible. The results further show that the information provided in informal citation does complement the information authors provide informally, particularly regarding URLs, publication dates, software descriptions, and type of citation. However, this does only apply to some practices of formal software citation, and not the most common type of software articles. Moreover, an inverse trend was observed regarding software articles, with a χ^2 test showing that authors citing software articles provide significantly less information in article full-text documents to enable the identification of software codebases. Overall, software articles were found to account for 69.9% of formal citations, while direct citations, found to be the most complete class, cover only 22.3%. With respect to prior studies, this corresponds to a higher amount of direct citations but confirms the results that software articles are the most cited resource for scientific software. A reason why this practice might be well adapted is that authors are familiar with article citation. In practice, the choice of formal citation type can also be influenced by citation recommendation of software authors [75], which are placed on software websites or repositories. Authors are likely to follow these recommendations in order to provide the desired attribution to the software developers, and because the provided information can be readily used. However, these recommendations often do not recommend direct citations but other citation types such as manuals and, therefore, omit essential information such as the version that identify the codebase and are part of the research provenance. For the widely used statistical framework *R*, for instance, the recommended citation style is a citation of the manual, which does not include a version number[16]. Further, authors that publish software articles have a vested interest in the article being cited and are likely to recommend the citation of the software article because they do otherwise not receive attribution and impact for the development of the software.

The results showed that software citations are often incomplete with respect to software, author, and codebase identification. Considering informal citation, information to identify a software beyond the software name is provided in 67.2% of cases, while information for developer attribution is only provided in 30.7%. Metadata allowing the identification of used codebase is also only available for 33.4% of software. Regarding formal software citation completeness was explicitly assessed for direct citation of software and citation of software manuals. The analysis of citation completeness

[16] The citation recommendation can be obtained by the function "citation()" in *R*; tested for *R* version 4.3.2.

showed that software is almost always identifiable from direct citations in 88%–99% of cases, even so the practice is only used in about 23% of formal software references. However, unique identifiers and archive links have almost never been utilized to identify software, showing that this is not a common practice, yet. Attribution of developers is also possible for a majority of references in 88% of cases, while the information regarding codebase is only identifiable in 66%–73%. In general, the use of version numbers was found to be a common practice, while release dates are almost never used, and mostly in cases where software developers utilize release dates, e.g., "Matlab r2023a". The situation regarding citations to manuals is similar, with software identification and attribution being given in most cases, while codebase identification is at a notably lower level (see Figure 3.19).

The data quality for representation of software citations by providers of bibliographic data was analyzed to assess if the existing infrastructure can be utilized for systematic analyses of software in science. While the sample size is small, it is considered sufficient to make reliable statements about the quality of formal software citation and its representation in bibliographic databases. The results showed that information provided by publishers is mostly unstructured for direct citations, hindering systematic analyses of software usage in science, which rely on structured information. Developer and date information are structured to some extend, but still less than 50% of cases, while the name is structured in even less cases and versions only in single instances. The only information that is almost always consistently structured are identifiers. In general, information from citations to manuals is more often structured than information from direct citations (see Table A2), which reflects the similarity of manuals to scholarly articles in contrast to other research objects such as software and data.

Crossref was found to mostly retaining the structure and information of the publisher. Especially, names and identifiers are almost always identical in their representation, while some information for versions, developers, and identifiers is lost. With respect to completeness, Crossref systematically represents author names by the last name of the first author for all references, but maintains all information for the actual elements. To access the data, a persistent identifier, i.e., DOI can be used to link the reference of a citing article to the actual entry in Crossref. However, this raises a problem, when such an identifier is not present, as it is the case for most software citations. Moreover, Crossref omits a small amount of direct software citations and manuals, as compared to software articles. Overall, Crossref is not performing any special treatment for software citation, but is mostly successful in representing the information available from publishers.

Semantic Scholar omits a significant amount of 22.9% of direct software references. It is successful at increasing the structure of information to some extend, while introducing errors when it comes to direct software citations and software manuals. Wrong information (14%) as well as duplicated information (18%) is present in 30% of represented references in Semantic Scholar, which likely results from adding information that was found by erroneous linking of different references to the same element (see Listing 15). Moreover, Semantic Scholar loses information within the software citations it retains. It does, for instance, drop a high number of versions, and the majority of URLs. It also introduces a high number of errors in versions, developers, and dates. Overall, software citation representation in Semantic Scholar is poor, because it seems that the underlying implementation has not been adapted to handle direct software citations. Specifically the concepts of versions and URLs that are common in software citations are not represented in Semantic Scholar. However, it succeeds in improving some references, and it can be assumed that with proper adaption it could be successful in representing and even adding structure to software citations, when the original published information lacks it.

Overall, based on the current scientometric systems, systematic analyses of formal software citations in scientific articles are not possible. Both analyzed databases are currently unable to adequately represent software citations, as proper handling of the direct citations does not seem to be implemented by both. Some fault does also lie with the publishers because in all annotated cases no specialized format for the citation of scientific software was used. Additionally, a spot check was performed on the Scopus database[17], which is generally known as a high quality data source for bibliometric studies [18]. It provides a large curated abstract and citation database, including metadata records of articles, author and institution profiles, and has been the basis for different bibliometric analyses [296, 162, 217]. The spot check revealed similar issues, for instance, unstructured data, missing specialized treatment for software with different version, or citations to the same software that are not linked and consequently evaluated independently. The data annotated here, can support publishers to better represent software citations as it highlights errors that exist in current practices. Necessary updates to the current infrastructure in order to represent the intricacies of software citation are discussed in Chapter 7.

In summary, establishing SoMeSci highlighted the need for large-scale analyses of software in science including an ED of software mentions. The results further outline the requirements of implementing such analyses. In general, they should focus on informal software citation rather than formal citation since it is more common, and several issues in the automatic analysis of formal software citation were identified. The analyses should also include metadata associated with software and its mention context. Moreover, it is necessary to also extract information on how software and metadata are related, as cases were identified where multiple software are mentioned in the same context. It is also required to perform a software ED in order to identify different articles mentioning the same software. Here, the focus should be on methods that perform an ED rather than an EL because the coverage of current databases was found to be insufficient for EL. Further, it should be investigated how external databases can be integrated for ED, as the information was also required by human annotators. In general, the task has proven to be challenging for human annotators and state of the art ML methods should be considered for solving it. Particularly, the identification of software, the classifcation of mention context, and software ED have proven to be challenging.

[17] `https://www.scopus.com`, accessed 3 March 2024.

4 | Automatic Information Extraction

The content of this section is based on the following publications:

> David Schindler et al. "Investigating Software Usage in the Social Sciences: A Knowledge Graph Approach". In: *The Semantic Web – ESWC 2020.* 2020, pp. 271–286. DOI: 10.1007/978-3-030-49461-2_16 [251], covers the development of a Bi-LSTM-CRF model for software entity recognition and of a manually engineered, rule-based software ED method.

> David Schindler et al. "SoMeSci- A 5 Star Open Data Gold Standard Knowledge Graph of Software Mentions in Scientific Articles". In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management.* CIKM '21. Virtual Event, Queensland, Australia, 2021, pp. 4574–4583. DOI: 10.1145/3459637.3482017 [244], covers establishing a Bi-LSTM-CRF benchmark model for recognition of software, related metadata, and classification of mention context, and the development of a Random Forest classifier for RE between identified entities.

> David Schindler et al. "The role of software in science: a knowledge graph-based analysis of software mentions in PubMed Central". In: *PeerJ Computer Science* 8 (2022), e835. DOI: 10.7717/peerj-cs.835 [245], covers the development of a large-scale extraction pipeline for software in scientific articles including recognition of software entities, related metadata, and context classification based on LLMs; RE between entities by an updated Random Forest approach, and software ED by automatic weighting of manually engineered rules by an ML model.

The requirements for large-scale analyses, outlined in Section 3.5, showed that it is necessary to extract information from in-text software mentions to obtain information on how software is used in scientific investigations. Therefore, full-text documents of scientific publications are the main analysis target from here on, while bibliographic references are no longer considered. The annotation effort further revealed that the problem is complex and requires the use of state of the art ML methods. Moreover, the automatic Information Extraction (IE) pipelines for the problem needs to integrate several different steps to extract all required metadata on software (see Section 3.1), its mention context (see Section 3.1.1), and identity (see Section 3.1.3). The required methodical approaches used for IE are derived from established methods in the domain of NLP because the main resource are human written texts and most problems that need to be solved are strongly related to existing NLP problems. The recognition of software and its related metadata is closely related to Named Entity Recognition (NER) and can be approached by the same methods, e.g., by sequence tagging. Context classification (software type and mention type) can be broader considered as a text classification problem, but can in this scope be integrated with the NER problem to save runtime and potentially improve performance. The requirements further showed that its necessary to match software and its metadata, which can be considered a Relation Extraction (RE) problem. Lastly, determining software identities is an Entity Disambiguation (ED) problem. In general, the outlined tasks have different levels of complexity and require different methodological approaches. SoMeSci is used as ground-truth data for the development and evaluation of the entire IE pipeline. The remainder of this chapter describes the methodical approaches for the problems of NER, RE, and ED, the chosen evaluation methods, and summarizes the corresponding performance of the IE pipeline. Lastly, it describes how the pipeline was applied on a large-scale dataset of $3.2\,M$ scientific publications.

4.1 Data and Preprocessing

All established methods are based on supervised training and require annotated ground-truth data. Table 3.3 contains an overview of all datasets available for this purpose, including the one established within this work. It was selected as the best suited ground-truth dataset because it has the best coverage of metadata and includes information on mention context and software identities. Furthermore, its annotation basis is more recent and provides a broader time range, which can benefit generalization, especially since the scientific software landscape is consistently changing. As described in Section 3.3, it was not possible to further combine datasets without considerable manual annotation effort due to annotation differences.

The data was split into training, development and test set in a 60:20:20 ratio to allow an unbiased evaluation of generalization performance as described in Section 2.4.1. Specifically, the *PLoS_m*, *PubMed_f*, and *Creation_s* subsets of SoMeSci were all separately split in a 60:20:20 ratio, with the *Creation_s* subset included because it contains mention contexts that are too rare in the other subsets to allow a proper evaluation. The *PLoS_s* set is entirely added to the training set to not introduce an evaluation bias by overrepresenting software[1]. The resulting number of validation samples for all specific tasks is provided together with the corresponding evaluation results. Moreover, specific test setups are designed in order to reliably estimate how well the established models generalize, e.g., regarding an overrepresentation of software mentions discussed in Section 3.4.

All article documents are preprocessed in order to apply ML methods[2]. The two main steps of the preprocessing are *sentenization*, splitting a text into individual sentences, and *tokenization*, splitting a sentence in its individual tokens. Here, a custom preprocessing is implemented for both problems to best represent software mentions because this is not a focus of existing methods. Particularly, the methods integrated in the Genia Tagger [151, 283, 284], which were developed to handle the special terminology and structure of scientific articles was adapted for software. Both, sentenization and tokenization are based on regex expressions, which were updated to incorporate aspects such as splitting between software and version numbers and false positive sentence splits where software is concerned, e.g., on publisher names that are often abbreviated ("SPSS Inc."). An example of a software specific tokenization is provided in Listing 16.

Moreover, preprocessing handles text normalization with respect to particularities of scientific articles that are not relevant to software mentions, often due to practical reasons. Mathematical formulas are replaced by a general placeholder token, because they cannot be reasonably split by a regex expression and lead to long sentences in terms of tokens, which makes applying sentence based classification more challenging. Greek letters and other special characters are normalized to keep them consistent between articles, and to reduce the number of overall characters that need to be represented, and, in turn, to increase the number of contexts available for each character. Moreover, citation formats are automatically adjusted, and missing brackets and semicolons are added. Lastly, preprocessing also handles the transformation between different data formats.

[1] This bias might also exist for *PLoS_m* and *Creation_s*, with details described in Section 4.2.3

[2] The terms used here were introduced in Section 2.4.1.

```
Sentence:           We used SPSS16 (SPSS Inc.) for all analyses.
Tokenized Sentence: We used SPSS        16        ( SPSS    Inc.     )  for all analyses .
Metadata:           O  O    B-Software  B-Version O B-Devel I-Devel  O  O   O   O        O
Mention Type:       O  O    Usage       O         O O       O        O  O   O   O        O
Software Type:      O  O    Application O         O O       O        O  O   O   O        O
```

Listing 16: Illustration of entity recognition and classification of software and mention type based on custom tokenization. “B-” indicates a begin tag, “I-” an inside tag, “O” an outside tag, “Devel” refers to a developer.

4.2 Named Entity Recognition (NER)

The goal of the NER task is to extract software and the related metadata from text. This concerns all metadata covered by the manual annotation introduced in Section 3.1. Strictly, not all metadata are named entities, but all can be extracted by sequence tagging methods used for NER [177, 155, 50]. Therefore, the main goal of this task can be defined as identifying all continuous selections of tokens in a text that refer to software or related metadata, including a classification of the metadata type. NER is commonly approached as sequence tagging problems where each sentence of text is individually processed, and each token in a sentence is tagged with a classification label. The task is usually framed with IOB2 format, where the labels indicate both, the type of entity for each token, and if a token is part of a longer entity. A *begin* tag (“B-”) indicates that a token is the first element of an entity, an *inside* tag (“I-”) indicates that it is part of an entity, and an *outside* tag (“O”) indicates that a token does not belong to an entity. An example of the given NER problem in IOB2 format is illustrated in Listing 16. Furthermore, the classification of *software type* and *mention type*, is integrated in this task as two additional objectives as depicted in Listing 16. This is a consideration in terms of efficiency, since large-scale prediction requires analyzing millions of articles and run-time can be saved when one instead of three models is applied. This multi-task setup can also improve recognition performance and lead to learning of better feature representations because training with multiple related tasks implicitly increases the sample size [232].

4.2.1 Multi-task Setup

The combined classification tasks, outlined above, are handled as a hierarchical, multi-task, sequence labeling problem as illustrated in Figure 4.1. A tokenized sentence (bottom-left, blue) is expected as an input, which is then processed by the shared layers to generate a feature representation (bottom-left, green). For now, details on the shared layer implementation are ignored and described in Section 4.2.2. However, the shared layers contain the main amount of trainable parameters and the capacity for feature representation. Based on the shared-layer feature representation the classification mechanism is applied. Each sub-task has separate output layers for sequence prediction, always consisting of a fully-connected layer and a second layer classifying IOB2 labels, which can either be implemented by a softmax or a CRF, dependent on the implementation of the shared layer, described in Section 4.2.2. Since multiple outputs are generated there are also multiple loss functions, all calculated by a cross-entropy loss as all are multi-class classification problems. For backpropagation the loss weights are summed since the loss values are assumed to be at a comparable level as all were generated by the same loss function. Therefore, techniques such as learning task weights [139] or normalizing gradients between tasks [49] are not applied. However, a grid search for loss weights is applied, which has been reported to be competitive compared to other task balancing approaches [290].

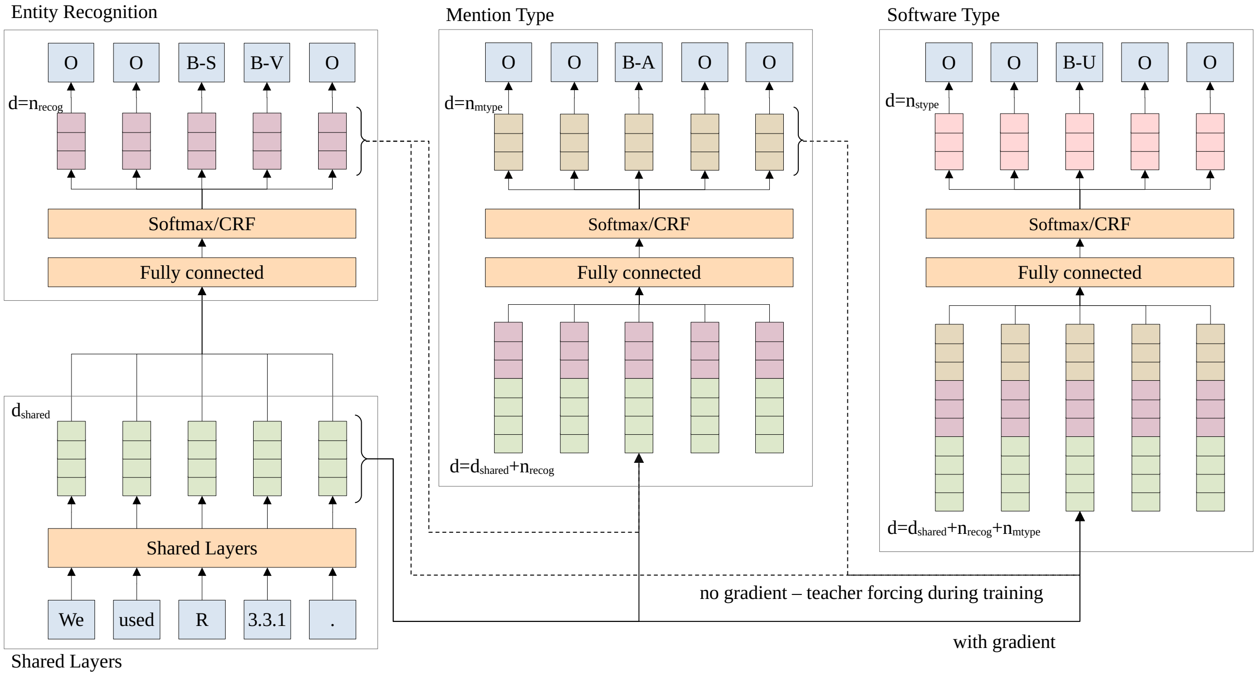


Figure 4.1: Illustration of the multi-task, hierarchical, sequence labeling architecture. Features are generated based on shared layers (bottom-left). The features are passed to three separate tasks and loss signals are summed to update shared weights. Outputs of entity recognition classification and mention type classification are passed back to the network as input features to higher level tasks, depicted from left to right in the image. Teacher forcing—replacing lower level classification outputs with gold label data—is conditionally performed during training. Colors represent similar types of information.

As depicted in Figure 4.1, the tasks are modeled hierarchically. This mechanism is implemented by adding the classification result of lower level sub-tasks as input to higher level sub-tasks. Specifically, the classification layer for mention type receives the software recognition output, while the software type classification layer receives the concatenated output of both software recognition and mention type layers. During training, no gradient is passed backward through the hierarchical connections, but only through the shared layers so the weight updates in each classification layer are only based on the individual task loss. Moreover, teacher forcing—passing the correct prediction regardless of the actual model prediction—is performed during training with respect to the output of the lower layers in the hierarchical classification because the tasks are strongly related and through this approach the model should reliably learn to only classify mention and software type if a software is actually present. It is expected that this approach leads to better update steps and a faster learning convergence. However, teacher forcing can also lead to errors in the test setting when the model receives inputs that vary from the correct input it receives during training time [95]. Therefore, it is investigated if teacher forcing improves performance by evaluating settings with different frequencies of teacher forcing.

Due to the multi-task learning objective the labels of multiple outputs have to be combined with potential tagging inconsistencies between tasks. The tags are automatically combined based on a rule set that enforces consistency: (1) all I-tags without leading B-tags are transformed to B-tags, including I-tags that do not match their leading B-tags; (2) entity boundaries for higher level tasks are adjusted to the base level entity boundaries; and (3) when there are multiple conflicting labels in higher hierarchy steps for one entity, the label for the first token is chosen. The process is illustrated

```
Sentence:        We  used  SPSS            Statistics        16         .
Metadata:        O   O     B-Software      I-Software        I-Version  O
Mention Type:    O   O     B-Usage         I-Mention         O          O
Fixed:           O   O     B-Software-Usage I-Software-Usage B-Version  O
```

Listing 17: Example for enforcing tagging consistency. Rules (1) and (3) are applied and tagging inconsistencies are highlighted (🟧).

in Listing 17.

The hyper-parameters of the extraction model with respect to multi-task learning are summarized in Table 4.1. One hyper-parameter determines whether multi-task learning is applied or whether a single model is trained for each of the individual tasks, with otherwise equal structure (Table 4.1 *Multi-task*). Further, the outlined task weights are varied individually for all three tasks in a fixed range (Table 4.1 *Task weights*), and the ratio of samples for which teacher forcing is applied is varied based on randomization (Table 4.1 *Teacher forcing*). The influence of these parameters is assumed to be largely independent of the shared layer implementation and is tested on the default model setup for the shared layer implementation because including them in the overall search space for hyper-parameters would increase run-time complexity to a no longer feasible level.

Parameter	Default	Range
Multi-task	yes	yes, no
Task weights	(1,1,1)	([1,5], [1,5], [1,5])
Teacher forcing	1.0	.5, .75, 1.0

Table 4.1: Hyper-parameters associated with multi-task learning, including their default value, and tested range.

4.2.2 Shared Layer Implementation

The shared layers allow different implementation methods. Methods that have been empirically proven to perform well for sequence prediction in NLP are used and compared to each other to select the best suited model for the given problem. The Bi-LSTM-CRF model combining different feature inputs could for a long time be considered state of the art on the problem [155, 177, 50]. The model is a combination of a bi-directional LSTM model and a CRF classification, both introduced in Section 2.4.3. This approach works on sentence basis and views the input as a sequences of tokens. Initially, token-based features are generated for each token in a sentence. These features usually consist of word embedding representations for the tokens, which are known to capture multi-level semantic similarities between words [184]. Here, an established Word2Vec model specifically adapted for scientific literature is used for this purpose [221]. Moreover, character based features that capture a tokens orthography are calculated by an additional Bi-LSTM model as in Lample et al. [155]. Alternative methods such as using CNNs have been proposed in the literature [177] but perform at a similar level. A Bi-LSTM is then applied over the sequence of concatenated feature representations, which captures sequence information based on the context of tokens, making information of the entire sequence available at each token due to the bidirectional nature (see Section 2.4.3). Finally, a fully-connected layer and a CRF are applied on the feature sequence generated by the Bi-LSTM to assign the most likely tag sequence. Integrating the CRF enables the model to represent tagging consistencies that exist due to the task design and are unknown to the previous steps, e.g., an *I*-tag can only occur after a *B*-tag.

There are several hyper-parameters associated with the Bi-LSTM-CRF model which are systematically optimized during training in order to select the best suited parameter combination for the given problem. Reasonable base assumptions based on prior studies [155, 177, 50] are made for parameters to keep the required effort for parameter optimization at a reasonable level. Table 4.2 summarizes all model specific parameters, their default setting, and the parameter search range. The main parameters of the LSTM define the hidden size of the model and the number of applied LSTM layers (Table 4.2 *LSTM size*, *LSTM layers*). Further, dropouts are applied on the token feature representation and on the sequential features generated by the Bi-LSTM (Table 4.2 *Dropout rate*). SGD is used to optimize the gradient descent in combination with gradient clipping to avoid exploding gradients as described by Lample et al. [155], who report that this method outperforms more complex gradient descent methods such as Adam [142] for Bi-LSTM-CRF models. The process is defined by the learning rate and the clipping value, which are set to literature values [155] (Table 4.2 *Learning Rate*, *Gradient clipping*). Additionally, a decay factor is introduced to decrease the learning rate during training (Table 4.2 *Learning Decay*). The custom setting halves the learning rate after a fixed number of epochs, and is compared to different functions reducing the learning rate either by linear, square-root, or logarithmic reduction. The negative sampling rate is a factor introduced to reduce the amount negative samples, with no software mentions, during training. The factor randomly retains negative sample with the given probability, i.e., a sampling rate of .25 retains about 25% of overall negative samples leading to an oversampling of sentences containing software that are otherwise sparse in the dataset (Table 4.2 *Sampling rate*). It is used because oversampling has been found to increase performance on imbalanced datasets [41]. Effectively, this parameter also modifies the trade-off between Precision and Recall as it increases the prior probability of software mentions in the data. Based on the different parameter setting the number of trainable parameters for the Bi-LSTM models varies between ≈151 k and ≈1, 81 M with ≈381 k in the default setting.

Parameter	Default	Range
Char LSTM size	50	25, 50, 75
LSTM size	100	50, 100, 150
LSTM layers	1	1, 2, 3
Dropout rate	.2	.0, .1, .2, .3
Learning rate	.01	[.05, .001]
Learning decay	custom	linear, sqrt, log
Gradient clipping	5.0	[1.0, none]
Sampling rate	1.0	[.25, 1.0]

Table 4.2: Hyper-parameters considered for Bi-LSTM-CRF models including their default setting, and test range.

More recently, pre-trained transformer-based models achieve state of the art results across a wide range of NLP tasks, see Section 2.4.3. Particularly BERT [67], a transformer model pre-trained on a self-supervised, masked language task, performed well and has been adapted in a broad range of applications. The main architecture of the model and its adaptions BioBERT, SciBERT, and PubMedBERT specifically pre-trained for scientific literature, has been described in Section 2.4.3. Here, the specific implementations are used as they have been shown to perform better on NLP tasks on scientific publications [160, 27]. BioBERT, SciBERT, and PubMedBERT are all utilized as pre-trained models and only adapted to the given tasks by adapting the final output layer. This is the common approach when working with large language models as the computational requirements for training new models from scratch are too high. Moreover, the approach of pre-training has been shown to work well for a broad range of NLP problems [67]. The added task specific layers for BERT models are simple fully-connected layers followed by a softmax activation function for each of the objectives outlined in Section 4.2.1. This means, BERT models have no specific layer for tagging consistency. It was briefly investigated if there is a performance gain from adding a CRF layer, but there was no positive effect and, in turn, a notable increase in required training run-time.

It is also not reported that an additional CRF layer improves NER performance with large language models, even so other work on software citation uses this approach [173][3].

Since the model architecture is fixed, the hyper-parameter space for BERT based models is smaller than for Bi-LSTM-CRFs. The hyper-parameters for this model only relate to the training setup for fine-tuning the model and selection of the pre-trained model, and are summarized in Table 4.3. BERT models are often available with different numbers of parameters, leading to higher representative power but also to higher run-time complexity and memory requirements (Table 4.3 *Model*). Particularly, BioBERT is available in two different parameter configurations with ≈110 M and ≈360 M trainable parameters, allowing an estimation of the model complexity and performance, while a comparison between the small BioBERT model, SciBERT, and PubMedBERT allows to assess if the underlying training data and method influence the representation quality of the language model for software identification, because all have ≈110 M trainable parameters. The learning rate (Table 4.3 *Learning Rate*) and dropout value (Table 4.3 *Dropout rate*) are chosen based on fine-tuning recommendations in the original BERT publication [67]. Further, gradient clipping (Table 4.3 *Gradient clipping*) is applied, which is a common approach when fine-tuning BERT models [188].

Parameter	Default	Range
Model	SciBERT	SciBERT, BioBERT small, BioBERT large, PubMedBERT
Learning rate	$5{\cdot}10^{-6}$	$[5{\cdot}10^{-5}, 5{\cdot}10^{-7}]$
Sampling rate	1.0	[.25, 1.0]
Dropout rate	.1	.0, .1, .2, .3
Gradient clipping	1.0	.5, 1.0, 2.0

Table 4.3: Hyper-parameters considered for BERT models including their default setting, test range, and a brief description.

As for LSTM models, a negative sampling rate (Table 4.3 *Sampling rate*) is also considered during fine-tuning BERT models. The training of BERT (incl. SciBERT, BioBERT, and PubMedBERT) is originally implemented using the Adam optimizer. Adam [142] uses adaptive learning weights, which can lead to faster convergence and decreased effort in optimizing for learning rate [231]. Here, fine-tuning is performed with AdamW, which changes the weight decay regularization compared to Adam, and was shown to improve generalization performance [174]. Note that parameter values and the corresponding test ranges between Bi-LSTM and BERT models vary, but were selected based on known parameter setups reported in existing literature. The differences stem from systematic differences in the training setup, e.g., Bi-LSTM models are newly trained, while BERT models are fine-tuned. The test range for the sampling rate was further determined during test time and set individually for both models because they were found to be differently affected by the parameter.

4.2.3 Evaluation

Since a multi-task problem is considered it is challenging to define a suited evaluation measure to compare different models. Here, recognition of software entities is defined as the main evaluation task for model selection. The value is selected because software extraction is the main objective of the model, and all other tasks directly depend on it, i.e., wrong software classification also leads to errors in metadata, mention type, and software type classification. Extracting information on the software itself is also most important for all intended applications of the model, as all other information can only be utilized in context to a software. The only exception is the evaluation of the multi-task hyper-parameters (Table 4.1), where the interaction of performance between tasks is

[3] Note that this work was performed in parallel to the investigations performed in this work.

of interest. Furthermore, the final performance of the selected model is reported for all tasks. In general, all results are reported through mean and standard deviation values of repeated training runs to make more reliable performance statements. This is done because NN training includes randomization in weight initialization and the training process, e.g., sample order, which can result in the optimization process leading to different local minima for repeated experiments, with potential performance differences between them.

A second evaluation setting is added to evaluate how well the model generalizes towards full-text document articles. As outlined in Section 3.4, software mentions might be overrepresented in SoMeSci due to the data selection process. This bias might also be reflected in the standard evaluation method as it includes samples from the *PLoS_m* and *Creation_s* sets. Therefore, an additional generalization test is performed by splitting the 100 full-text document articles from *PubMed_f* in five non-overlapping sets and performing a cross-validation using all remaining data as training data. The cross-validation setting is chosen to include the remaining articles from *PubMed_f* to avoid a potential domain or journal specific bias, as most articles of the largest *PLoS_m* set are samples from PLoS, which has an interdisciplinary focus. Furthermore, this test is performed based on the described default model to assess the generalization performance without the optimization process adding a bias to the evaluation. In general, this test is also a good indicator of how well the model will generalize when applied to further articles sampled from PMC.

4.3 Relation Extraction (RE)

The goal of the RE step is to classify whether and what type of relation exists between software and metadata. The set of possible relations is restricted to the relations described in Section 3.1.2. Namely, every metadata entity is required to have a relation relating it to a software, with the exception of version, which can also be related to a license, and URLs and alternative names, which can both be related to licenses or developers. Moreover, software can be related to other software, if one software functions as a *PlugIn* and the other as the host-software. RE is applied on top of NER outputs during prediction and, therefore, depends on NER performance and the resulting data quality. During training and model selection and for performance estimation the RE model is trained with the gold-standard NER data, assuming perfect prediction of software and metadata entities[4].

The RE task is considerably easier than the NER task due its restricted setting. Furthermore, it is also easier than general RE problems discussed in the literature [187, 322]. Based on the definition of the relation only certain entity combinations can be related, and it is often known that a specific relation has to be given between software and metadata. The challenging cases are instances where multiple software is mentioned within one sentence, as the association is not clear in theses cases and the relation between software is optional. Since the problem is easier, it is also possible to apply less complex classification mechanisms that do not learn representations but are based on manual feature engineering. These methods are faster during training, model selection, and prediction of new data. Moreover, the feature engineering effort stays at a comparable low level if simple rules are sufficient to represent the problem.

In the scope of this work, the following features were manually developed and implemented as basis for the classification: 1. entity order, 2. entity types, 3. entity string length, 4. entity distance in tokens, 5. number of software entities in the given context, 6. sub-string relations between entities, and 7. matching of automatically created acronyms. All features are either represented as integer or boolean values dependent on the respective context. During development of the features

[4] Perfect in terms of ground-truth quality, which can potentially still contain errors.

further candidate features were included and later pruned from the final selection because they did not improve overall performance, but negatively impacted the overall run-time. The pruning was performed by running the classifier and specifically omitting features while evaluating the performance.

The Random Forest classifier is selected for the RE problem on top of the generated features because it is a standard approach in ML based on manual feature engineering and known to achieve high performance on classification problems, while being more run-time efficient than complex NN approaches. The method is introduced in Section 2.4.3 and optimized with respect to the hyper-parameters listed in Table 4.4. The number of overall trees (# *Trees*), the impurity criterion (*Criterion*), the maximum number of features considered per split (# *Features*), and the maximum number of samples used for bootstrapping single trees (# *Bootstrapping*) were optimized. Further, maximum tree depth (*Max Depth*) and minimum number of samples required in a node for further splitting (# *Split Samples*) were used to regularize the building process of individual Decision Trees.

Parameter	Default	Range
# Trees	100	50, 100, 200, 300
Criterion	Gini	Gini, Entropy
# Features	sqrt	sqrt, log, half, all
# Bootstrapping	1.0	[.25, 1.0]
Max Depth	None	5, 10, None
# Split Samples	2	2, 4, 6

Table 4.4: Hyper-parameters considered for Random Forest RE models including their default setting, and test range.

4.3.1 Evaluation

RE is tested in the same evaluation settings as NER (see Section 4.2.3), including the standard evaluation setup and the test for full-text generalization. To allow a better overview whether “relates-to” relations are more often wrongly classified in combination with a specific entity, they are relabeled during training and evaluation, e.g., “version-relates-to” in combination with a version entity. This is assumed to not influence overall performance as it imposes further restrictions on the data, and the information on entity type is included in the features, i.e., a decision tree should be able to directly represent that a “version-relates-to” relation is only possible regarding a version.

4.4 Entity Disambiguation (ED)

In the given setting, ED has the objective of determining which software mentions refer to the same software entity. Software ED is essential for numerous analyses because, as outlined in Section 3.1.3, the same software is often referred to by different names due to use of abbreviations, geographical differences in software names, or naming conventions changing over time. The major challenge in software ED is that EL to external knowledge bases is not possible, as discussed in Section 3.5. Therefore, it is necessary to adapt methods of ED which work by clustering all mentions referring to the same software without explicitly linking to an external knowledge base. Instead, existing information from the mention context, such as orthography and contextual information, are utilized. Furthermore, run-time has to be taken into account when clustering a large-scale dataset, because the required calculations can quickly become infeasible when millions of mentions are examined, with distance calculations of all mention pairs requiring $\mathcal{O}(n^2)$ operations for n software mentions. Further ED methods for developers, licenses, or other metadata could also be developed but are not considered here, as analyses of software is the main objective of this work.

Two different approaches for ED based on clustering were implemented and compared in terms of performance and run-time. Both make the base assumption that mentions with the same name refer to the same software, which strongly reduces the number of required comparisons and improves run-time. On the other hand, this approach also introduces an error as it was found that software mentions with the same name can refer to different software during annotation (see Section 3.1.3). However, this case is rare and was only identified for one pair out of 884 distinct software within SoMeSci. Therefore, it is assumed that high ED performance can still be achieved under this assumption, which makes the trade-off between run-time and quality reasonable.

4.4.1 Rule-based approach

The rule-based approach employs manually created rules to directly cluster software mentions together. For this purpose the following rules are used:

1. Normalized string comparison, entailing: case-folding, replacing of special characters, removal of trailing numbers, removal of trailing syllable “pro”, removal of stopwords within the mention, and stemming.

```
Statistical Package for the Social Sciences → statist packag social scienc
SPSS                                        → spss
IBM-SPSS Statistics                         → ibm spss statist
SPSS16.0                                    → spss
Statistic Package for Social Study          → statist packag social studi
```

2. Acronym comparison, entailing: automatically generated acronyms on normalized strings based on leading characters.

```
statist packag social scienc                → spss
statist packag social studi                 → spss
```

3. URL comparison based on provided metadata URLs.

4. Comparison of known DBpedia alternative names, entailing
   - *rdfs:label*
   - *foaf:name*
   - *dbo:wikiPageDisambiguates*
   - *dbo:wikiPageRedirects*

The rules are sequentially applied and software that matches based on the rule is clustered. The clustered mentions are then used in aggregated form for the following rules. External knowledge from DBpedia is utilized in the last step. Even so a full EL is not possible based on external knowledge bases, the partial information on alternative names for specific software can be utilized. The corresponding alternatives name were obtained for all software concepts covered in DBpedia with the exception of *video games*. Further, data was gathered in multiple languages (English, Spanish, French, and German) to capture potential language based naming variations. The used query is listed in Listing A22, and was executed in April 2021.

4.4.2 Automatic weighting of rules

Same as the rule-based approach to ED, the automatic weighting of rules is also based on a set of manually engineered rules, but instead of directly applying them for clustering, they are processed

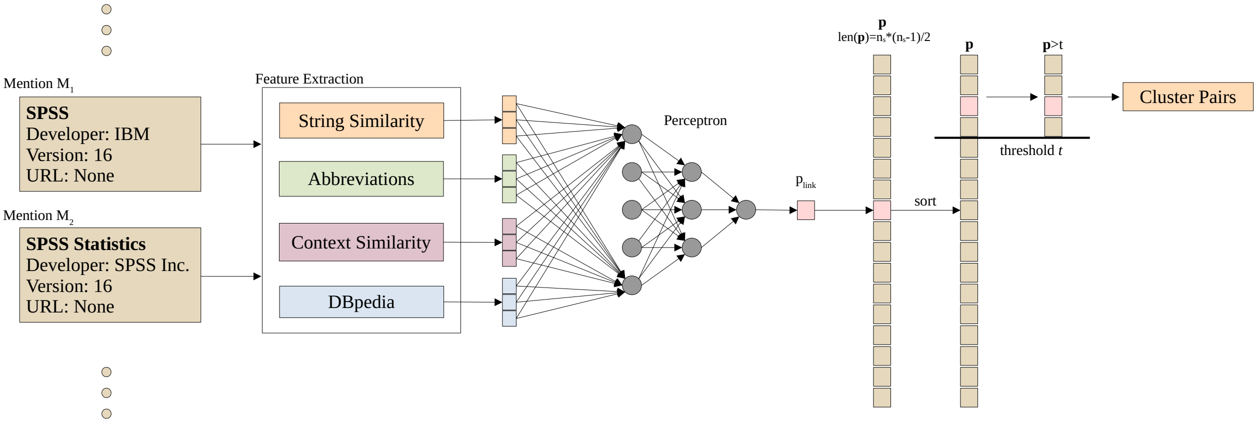


Figure 4.2: Overview of the software ED by automatic weighting of rules. For all pairs of extracted software mentions (m_1, m_2), features are extracted (*Feature Extraction*) and used to determine a probability between mentions indicating if they should be matched (*Perceptron*). Finally, agglomerative clustering is performed to cluster similar software names.

by a NN trained supervised on the task. This follows the general architecture of state of the art methods for NLP ED, with an overview of existing literature provided in Section 2.3. In difference to the most recent methods, however, manually generated rules were preferred over embeddings, since it is not clear how well scientific software is represented in these embeddings (see Section 2.3). An overview of the approach proposed in this work is given in Figure 4.2. For a given pair of software mentions (m_1, m_2) the objective is to determine whether they refer to the same software entity. For each pair (m_1, m_2) a feature vector $\mathbf{v}_{m_1,m_2}$ is established with features based on string similarity, similarity of extracted context, automatically generated abbreviations, and similarity of alternative names gathered from DBpedia. The vector is then mapped to a probability estimate p_{link} by a NN, which estimates whether the mentions refer to the same entity. A low complexity, multi-layer perceptron is used for this purpose made up of four fully-connected layers $(15 \times 10 \times 5 \times 1)$ in order to keep run-time complexity to a minimum. The model is trained supervised to predict whether a link exists based on the annotated ground-truth data based on a sigmoid output, which is then considered as a probability estimate. Due to the simplicity of the model, no extensive hyper-parameter optimization was performed.

In the final step, agglomerative clustering is performed based on the predicted probabilities using a single-linkage approach to merge the clusters. For that purpose, the two clusters containing the mention pair with the largest probability p_{link} are combined at each step until a threshold t is reached, indicating the minimal probability after which no further clustering is performed. The threshold t itself is determined based on the available gold standard labels, and set at the value for which they are optimally clustered. Here, the optimal threshold can vary depend on the underlying dataset, as the decision boundaries shift in a more densely populated feature space, as illustrated in Listing 18. To counteract it, the threshold t is optimized separately for prediction by determining it based on the merged set of gold standard data points and prediction data. Same as before, the threshold is set at the value where the gold standard data is best predicted, but this threshold shifts due to the additional data and the denser feature space. As before, the performance for this step is estimated in terms of Precision, Recall and FScore.

Single and average linkage were compared for clustering and found to perform almost identically for gold standard data alone. Single linkage was then chosen for all subsequent tests, as it offers strong benefits in run-time and space complexity compared to average linkage. When single linkage

```
1  PMC3629193:   PASW Statistics    17     SPSS, Inc.
2  PMC6910861:   SPSS               20.0   IBM
3  PMC6322718:   SPSS PASW          17     SPSS Inc.
```

Listing 18: Example illustrating the effect of adding more features in a clustering setting. Samples one (PMC3629193 [141]) and two (PMC6910861 [48]) have high distance in the feature space based on the provided information because all information is different and DBpedia does not reflect *PASW Statistics* as an alternative name for *SPSS*. However, both are close to sample three (PMC6322718 [277]) and will be added to the same cluster with a low threshold when three is included in the dataset. Software name (🟦), alternative name (🟪), developer (🟧), and version (🟩) are stated as provided in the original articles.

is applied the initially calculated similarities can be used for the entire agglomerative clustering, as it allows merging clusters based on the closest existing pair at every step. Average linkage, in contrast, would require additional computation for cluster distances based on average similarity after each new cluster is generated, which would lead to higher run-time requirements.

As described, the base assumption was made that all mentions with the same name refer to the same software. This was implemented as a pre-clustering step, but multiple contexts of each software were included during prediction, to increase the number of available contexts for each software. This step is important as clustering takes context information into account. However, a trade-off between run-time and sample number was required, and the number was therefore set at six contexts for each unique software name. To further improve run-time, symmetric feature vectors between mention pairs (m_1, m_2) were assumed, reducing the number of required comparisons to $\frac{n(n-1)}{2}$. In practice, this assumption is not strictly correct, as the entities' string lengths are included as features, which makes the feature vectors non-symmetrical. However, the NN is trained on both directions, and is assumed to make similar predictions between $\mathbf{v}_{m_1,m_2}$ and $\mathbf{v}_{m_2,m_1}$.

A problem that was encountered when working with this approach is the interaction between the sample size and the density of samples in the resulting feature space. In the manually annotated training set with a small number of data points the samples are less dense than in the larger inference set described in Section 4.8.1. Moreover, the large-scale set is based on automatic extraction instead of manual extraction which means that it is likely to have a higher number of false positive mentions with a resemblance to software mentions. This increases the difficulty of finding suited decision boundaries for the posed ML problem based on the training set alone. To counteract this problem, data augmentation was applied to add further entities resembling false positive extracted software names, which should not be linked to any other mentions. To simulate closeness to existing software names the new samples were generated by recombining sub-strings of existing samples, e.g., *ImageJ* and *SPSS Statistics* could be combined to *Image Statistics*. It was made sure to not re-create given software names or duplicates during augmentation. In total, $2n$ augmented samples were created from the n original software mentions and included with the training samples. Augmented samples were also added to the test set to estimate performance with false positive samples. This cannot lead to an overestimated evaluation as only negative samples, which should not be linked, are added, which do not affect the employed values of Recall, Precision, and FScore.

4.4.3 Evaluation

The clustering performance for ED is compared between rule-based clustering and automatic weighting of rules. Due to the nature of the clustering task, the performance can shift with the amount of input data, as new clustering links can occur with each added sample. Therefore, two evaluation settings are considered. In the first setting, only the annotated gold standard data is used, while in the second setting additional 100 k articles are included in the gold standard test set. For each article the final selected model of the described NER (see Section 4.2) and RE (see Section 4.3) steps were applied to extract software and related metadata. The software and related metadata is clustered together with the samples from the test dataset, with the performance estimation based on how well the test dataset is clustered in this setting. This approach increases data size and also adds noisy data due to prediction errors, which allows a further estimation of robustness of the ED step. Here, no generalization test is performed, because in small data samples the number of actual existing links between samples becomes too small to allow a meaningful evaluation, which would be the case in the generalization setting.

To enable the comparison described above, the ML component of the automatic weighting approach is initially trained as described in Section 4.4 to predict if two samples should be linked, and its performance assessed separately on the test set. Further, the required threshold t of the automatic weighting approach is determined based on clustering the training data. In the 100 k setting, the additional articles are also added to the training set when determining the threshold. The threshold setting is then used to estimate the performance on the development and test set.

4.5 Results

In the following, all results regarding IE hyper-parameter optimization and performance are summarized, split by the considered tasks of NER, RE, and ED. The NER results are further split between hyper-parameter optimization for Bi-LSTM-CRF, Transformer models, multi-task training parameters, and prediction performance estimation. Next, RE results including hyper-parameter optimization and final performance estimation are presented, and, lastly, the performance evaluation results for ED are compared between the manually engineered approach and the ML based weighting of rules.

4.5.1 Named Entity Recognition

Bi-LSTM-CRF hyper-parameter optimization

All Bi-LSTM-CRF hyper-parameters were tuned individually based on the default configuration summarized in Table 4.2, with a reasonable default setting based on existing literature that reported good performance on existing NER problems (see Section 4.2.2). An overview of the optimization process results on the development set is given in Figure 4.3. While performance of the default configuration achieves a value of P=80% (±1.6), R=77.4% (±1.1), and F=78.7% (±.4), it can be seen that the hyper-parameters do have varying influence on performance. The *Clipping Value* is found to have an influence, with a higher clipping threshold leading to better performance, with the effect dropping of at higher values, and the best performance of P=83.6% (±2.2), R=78.9% (±.8), and F=81.2% (±1.4) achieved without application of gradient clipping. For application of *Dropout* values no notable performance differences are observed. The same is also true for parameters influencing the model capacity in terms of trainable parameters, i.e., the *Char LSTM Size* and the *LSTM Hidden Size*. The number of applied *LSTM Layers*, retaining an approximately equal number of parameters but splitting them in multiple stacked LSTM layers, is also found to not influence performance. The *Initial Learning Rate* is observed to influence performance with a higher rate leading to better performance with the best performance achieved at the highest tested parameter setting of $lr = 0.05$, at values of P=83% (±1.2), R=77.4% (±.9), and F=80.1% (±.8). Different methods of *Learning Rate Decay*, on the other hand, were not found to influence performance.

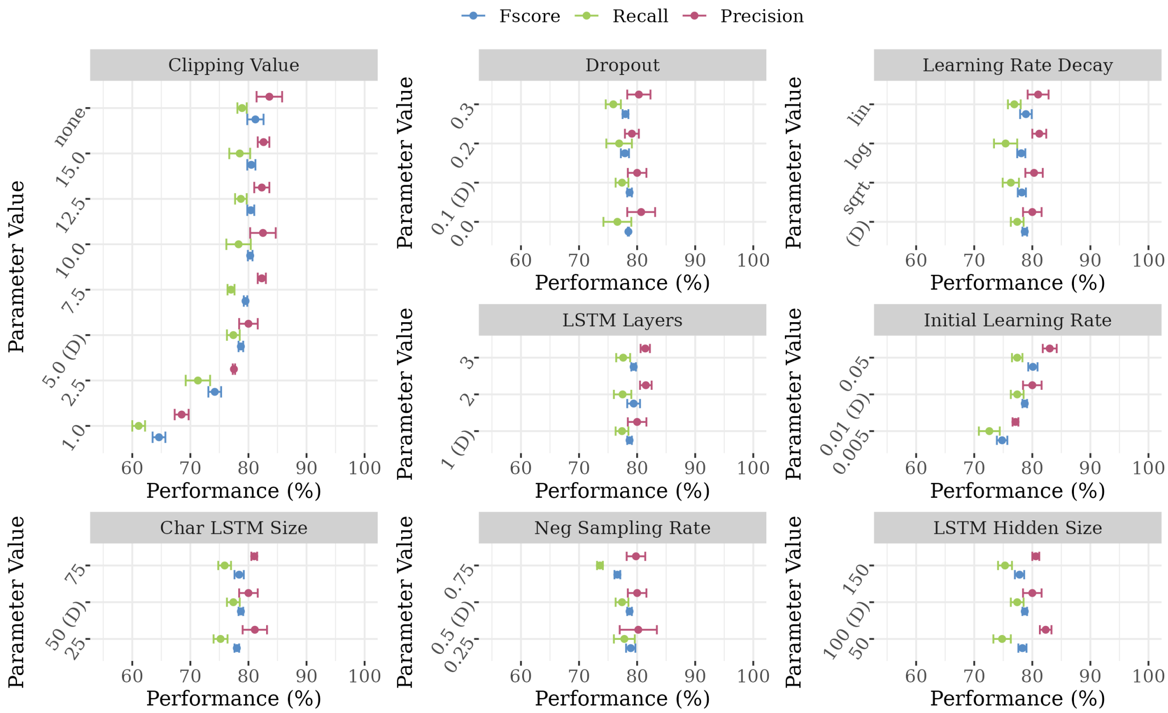


Figure 4.3: Hyper-parameter optimization results for the Bi-LSTM-CRF model on the development set. The default parameter setting is indicated by *(D)*. All plots show the mean performance value and one standard deviation of repeated tests to assess the influence of randomness in training. Note that the performance axis does not start at 0 but at 55 to allow a better comparison of small performance differences.

Lastly, the *Neg Sampling Rate*, performing an increasing oversampling of positive samples at lower values, is found to have an influence on performance, with higher oversampling leading to better performance. However, the benefit does not notably improve beyond the default value.

In summary, the results suggest that the performance of the default model can be further optimized by hyper-parameter optimization, particularly by optimizing the parameters controlling the training process. With respect to these parameters, further investigation of the learning rate would be required as the best performance was reached at the highest tested value. Moreover, a more exhaustive search strategy would have to be applied as multiple parameters were found to influence the performance, but their interaction requires further investigation. In the scope of this work, no further tests of the Bi-LSTM-CRF model were performed. This decision was made because the initial results for Transformer based models, summarized below, showed that they outperformed the Bi-LSTM-CRF model by a considerable margin, which led to the decision to not allocate further computational resources for the costly optimization of the Bi-LSTM-CRF.

Transformer hyper-parameter optimization

As before, all hyper-parameters for the Transformer based models were tuned individually with the remaining parameters set to the default configuration, selected based on the suggested parameters for fine-tuning in the literature (see Section 4.2.2). The optimization results on the development set are summarized in Figure 4.4 and contain only a limited parameter set because parameters such as the model architecture and its capacity are fixed by the pre-training. The default training configuration achieves a performance of P=86.3% (±1.1), R=89.6% (±1.2), and F=87.9% (±.3). The *Initial Learning Rate* is observed to influence performance, with high and low values showing perfor-

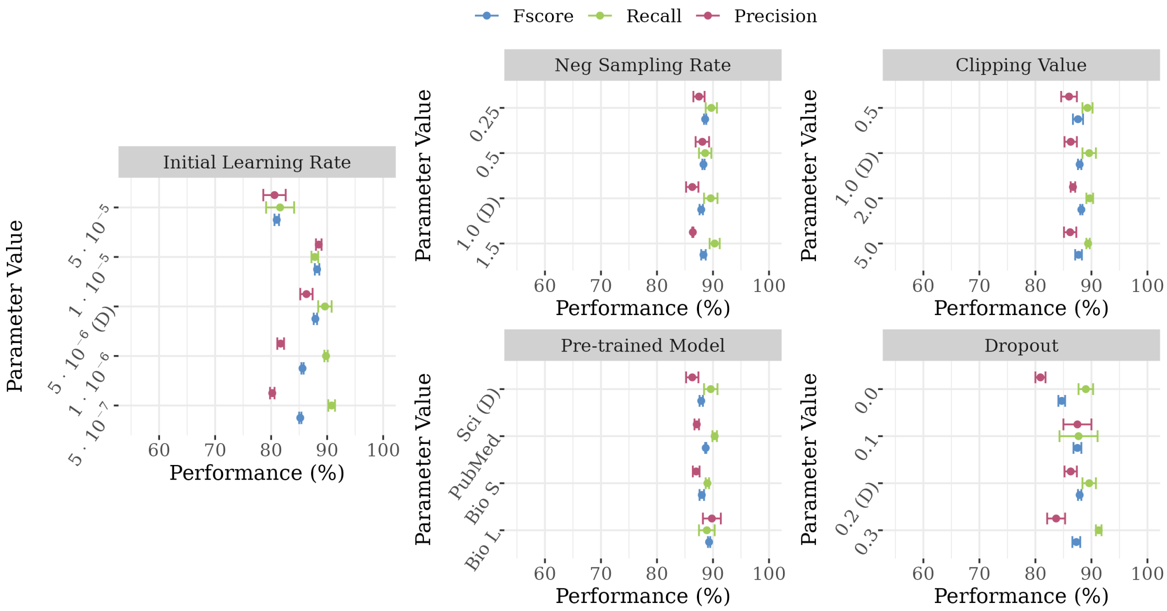


Figure 4.4: Hyper-parameter optimization results for Transformer based models on the development set. All plots show the mean performance value and one standard deviation of repeated tests to assess the influence of randomness in training. The default parameter setting is indicated by *(D)*. Note that the performance axis does not start at 0 but at 55 to allow a better comparison of small performance differences.

mance decreases, while the best performance is found at $lr{=}1{\cdot}10^{-5}$ with values of P=88.5% (±.5), R=87.8% (±.6), and F=88.2% (±.4), and at default value ($lr{=}5{\cdot}10^{-6}$). Moreover, with increasingly lower learning rate a performance gap between Precision and Recall can be observed, which does not exist at value $lr{=}1{\cdot}10^{-5}$. Regarding both the *Neg Sampling Rate* and the *Clipping Value*, no notable performance differences are observed. The *Dropout* value influence performance, with a higher value in the observed interval leading to better performance. In terms of FScore the performance stays at a comparable level for dropout values of .1, .2, and .3, while the performance in Precision and Recall varies with a lower Precision and higher Recall at higher values. Lastly, the selection of the *Pre-trained Model* does influences performance. The values for SciBERT are given as baseline, while PubMedBERT achieves a performance of P=87.1% (±.4), R=90.3% (±1.2), and F=88.7% (±.1), and the small and large version of BioBERT achieve respective performances of P=87% (±.6), R=89% (±.3), and F=88% (±.4), and P=89.8% (±1.6), R=88.9% (±1.4), and F=89.3% (±.2).

Compared to the prior described Bi-LSTM-CRF model, the results are considerably higher by ≈4 *pp* in Precision, ≈10 *pp* in Recall, and ≈8 *pp* in FScore, with the largest improvement in terms of Recall. Overall, the results confirm that the default fine-tuning parameter setup is reasonable for the problem and leads to good performance results, while a small adjustment to the learning rate can lead to a better Precision-Recall trade-off. The second aspect that has potential for optimization is the selection of the pre-trained model, with the models varying in model size regarding number of trainable parameters and underlying pre-training data. The comparison between BioBERT (S) and BioBERT (L) show a small performance improvement in the more complex, larger model by 2.8 *pp* in Precision and 1.3 *pp* in FScore, while the Recall is at a similar level. However, the larger model also has higher computational requirements which have to be taken into account when considering a large-scale application. Unfortunately, the remaining models are only available in a single configuration and do not allow further investigation of the model capacity on performance.

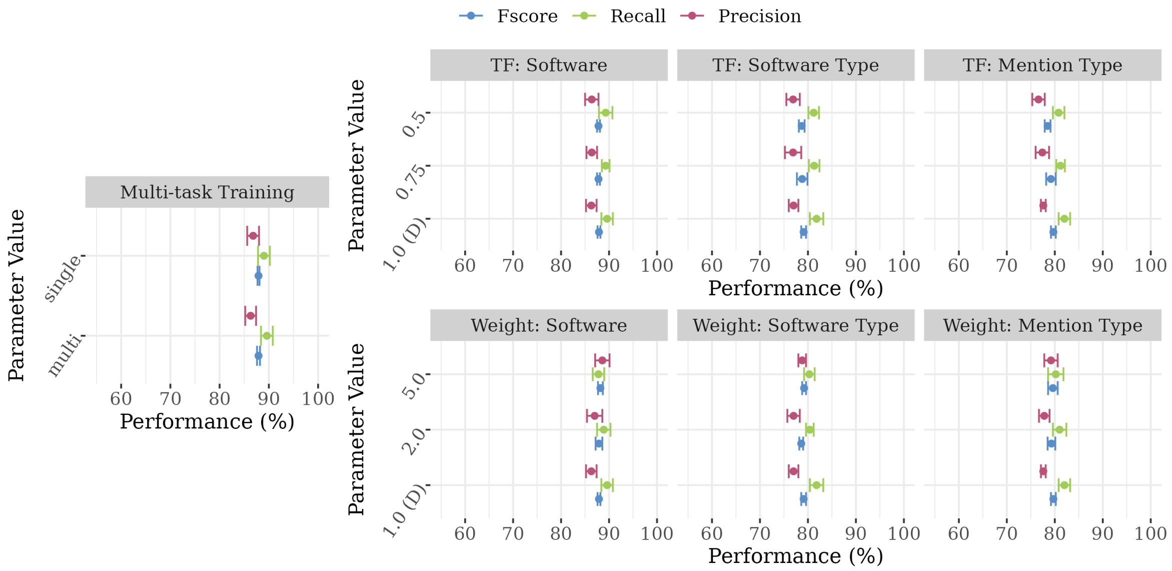


Figure 4.5: Hyper-parameter optimization results for multi-task learning on the development set. The default parameter setting is indicated by *(D)*. All plots show the mean performance value and one standard deviation of repeated tests to assess the influence of randomness in training. Note that the performance axis does not start at 0 but at 55 to allow a better comparison of small performance differences. TF: teacher forcing; Weight: task weight.

With respect to the underlying training data, it is found that SciBERT and BioBERT (S) perform equally, while PubMedBERT performs better by a small margin ($<1\,pp$). Finally, SciBERT is selected as the preferred model since it is not primarily trained on Life Science publications (see Section 2.4.3). While this might offer an advantage on the given dataset, it is argued that SciBERT might be better suited for an application in other scientific domains, i.e., Computer Science which is included in its training set. Therefore, the only change to the initially selected model is an adjustment of the learning rate to a value of $lr{=}1{\cdot}10^{-5}$.

Multi-task Learning

As described, the effects of multi-task learning were assessed on the development set based on the BERT default setup. They are summarized in Figure 4.5. On the left side, the difference in software recognition performance regarding *Multi-task Training* is depicted, for which no differences are found. Further, the influence of teacher forcing (*TF*) and sampling weight (*Weight*) are assessed. Here, results in Figure 4.5 are not only depicted for software recognition, but for all tasks, since the parameters are used to optimize performance between tasks. It is found that teacher-forcing does not influence performance of software recognition, which is expected since teacher-forcing only modifies the subsequent training process. Further, no notable influences on the tasks of software type and mention type classification is observed. The same is true for task weighting regarding all three tasks. Therefore, the model setup is retained as described in the default configuration, with multi-task learning applied since it is more run-time efficient, even so it does not influence performance.

Performance summary

The final hyper-parameter setup based on the optimization results described above is summarized in Table 4.5. Its final prediction performance on the test set is illustrated in Figure 4.6 and exact performance values for both test and development set are provided in Table 4.6. Software recognition performs well with values of P=87.9%, R=85.4%, and F=86.6%. Overall performance is also high at a value of P=86.2%, R=86.4%, and F=86.2%. However, when considering individual entities, performance differences can be observed. While developer (F=86.5%), version (F=91.8%), and URL are recognized with high performance rates, Release (F=78.7%), Citation (F=79.9%), and License (F=81.6%) are at a lower but good level. For the targets of Alternative Name (F=50%), Abbreviation (F=61.7%), and Extension (F=55.1%) notably lower performance values are found. Overall, performance is correlated with the number of available data samples, with particularly few samples available for Extension, Alternative Name, Release, License, and Abbreviation.

Considering the task of software type classification, the overall performance is lower than entity recognition but at a high level with values of P=74.4%, R=73.8%, and F=73.3%, considering that error propagation from software recognition is taken into account in the evaluation. It is found that the different types are recognized with varying performance, with *PEs* (F=92.3%) and *OSs* (F=83.7%) recognized with high performance, while *PlugIns* (F=40.1%) show the lowest performance due to confusion with *Applications* (F=77.4%). Lastly, the task of mention type classification also performs well considering that error propagation is taken into account with values of P=79.5%, R=78%, and F=78.4%. Here, the targets are also recognized with varying performance. *Creation* (F=90.3%) and *Deposition* (F=84.6%) are recognized with high performance, while *Allusion* (F=49%) proves to be the most challenging target due to confusion with *Usage* (F=81.2%).

Between the development and test set a performance decrease is observed concerning all tasks. Regarding software and general entity recognition the margin is small with 1.6 *pp* FScore (F=88.2% to F=86.6%) and 2.1 *pp* (F=88.3% to F=86.2%), re-

Parameter	Value
Model	SciBERT
Learning rate	$1{\cdot}10^{-5}$
Sampling rate	1.0
Dropout rate	.2
Gradient clipping	1.0
Multi-task	yes
Task weights	(1,1,1)
Teacher forcing	1.0

Table 4.5: Hyper-parameters setup of the final selected NER model, including fine-tuning and multi-task learning parameters.

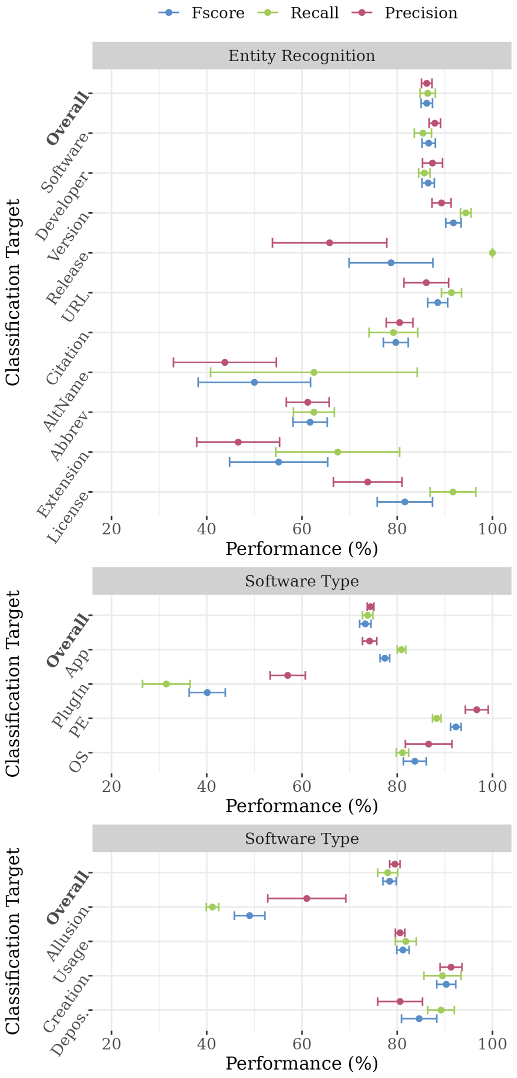


Figure 4.6: Final NER performance of the selected model. All plots show the mean performance value and one standard deviation of repeated tests. Note that the axis starts at 20.

spectively. With respect to software type classification the drop is larger with 5 *pp* (F=78.3% to F=73.3%), while the difference is also small for mention type classification with .7 *pp* (F=79.1% to F=78.4%). Overall, the tested model is strongly overfitted with an almost perfect performance on the train data, however, stronger regularization was not found to improve performance.

Target		Test Set	Devel Set
Software (n=511; 551)	P	87.9 (± 1.2)	88.5 (± 0.5)
	R	85.4 (± 1.8)	87.8 (± 0.6)
	F	86.6 (± 1.4)	88.2 (± 0.4)
Developer (n=105; 106)	P	87.4 (± 2.1)	89 (± 1.9)
	R	85.7 (± 1.2)	89.2 (± 0.8)
	F	86.5 (± 1.3)	89.1 (± 0.7)
Version (n=135; 171)	P	89.3 (± 2.0)	92.8 (± 1.2)
	R	94.4 (± 1.1)	97.5 (± 0.6)
	F	91.8 (± 1.6)	95.1 (± 0.7)
Release (n=7; 11)	P	65.8 (±12.0)	77 (± 5.3)
	R	100 (± 0.0)	81.8 (± 0.0)
	F	78.7 (± 8.8)	79.2 (± 2.8)
URL (n=61; 52)	P	86.1 (± 4.7)	85.2 (± 4.5)
	R	91.4 (± 2.1)	89.9 (± 1.6)
	F	88.5 (± 2.1)	87.4 (± 2.7)
Citation (n=66; 129)	P	80.5 (± 2.8)	81.8 (± 1.0)
	R	79.2 (± 5.1)	89 (± 3.4)
	F	79.7 (± 2.6)	85.2 (± 1.5)
AltName (n=2; 7)	P	43.8 (±10.8)	64.7 (± 3.9)
	R	62.5 (±21.7)	71.4 (± 0.0)
	F	50 (±11.8)	67.9 (± 2.1)
Abbrev (n=10; 12)	P	61.2 (± 4.5)	67.5 (± 3.9)
	R	62.5 (± 4.3)	93.8 (± 3.6)
	F	61.7 (± 3.6)	78.3 (± 2.1)
Extension (n=10; 5)	P	46.6 (± 8.7)	40.8 (±11.6)
	R	67.5 (±13.0)	55 (± 8.7)
	F	55.1 (±10.3)	45.7 (± 8.3)
License (n=9; 10)	P	73.8 (± 7.2)	62.2 (±15.1)
	R	91.7 (± 4.8)	77.5 (±17.9)
	F	81.6 (± 5.8)	68.9 (±16.2)
Overall (n=916; 1054)	P	86.2 (± 1.1)	87.3 (± 0.4)
	R	86.4 (± 1.6)	89.4 (± 0.8)
	F	86.2 (± 1.2)	88.3 (± 0.3)

Target		Test Set	Devel Set
Application (n=348; 401)	P	74.2 (± 1.5)	79.4 (± 1.1)
	R	80.9 (± 0.9)	83.9 (± 1.3)
	F	77.4 (± 1.0)	81.5 (± 0.9)
OS (n=33; 16)	P	86.6 (± 4.9)	88.4 (± 3.7)
	R	81.1 (± 1.3)	93.8 (± 0.0)
	F	83.7 (± 2.4)	91 (± 2.0)
PlugIn (n=81; 71)	P	57 (± 3.7)	53.3 (± 1.6)
	R	31.5 (± 5.0)	35.6 (± 4.7)
	F	40.1 (± 3.8)	42.4 (± 3.3)
PE (n=49; 63)	P	96.7 (± 2.4)	94.2 (± 2.7)
	R	88.3 (± 0.9)	95.2 (± 1.1)
	F	92.3 (± 1.1)	94.7 (± 1.1)
Overall (n=511; 551)	P	74.4 (± 0.7)	78 (± 0.9)
	R	73.8 (± 1.1)	79.2 (± 0.9)
	F	73.3 (± 1.2)	78.3 (± 1.0)
Creation (n=64; 45)	P	91.3 (± 2.3)	78 (± 6.3)
	R	89.5 (± 3.9)	78.9 (± 6.0)
	F	90.3 (± 2.0)	78 (± 1.6)
Deposition (n=30; 29)	P	80.6 (± 4.7)	77.1 (± 4.1)
	R	89.2 (± 2.8)	88.8 (± 2.9)
	F	84.6 (± 3.7)	82.4 (± 2.6)
Allusion (n=65; 71)	P	61 (± 8.2)	55.7 (± 6.4)
	R	41.2 (± 1.3)	48.6 (± 2.9)
	F	49 (± 3.2)	51.7 (± 3.5)
Usage (n=352; 406)	P	80.6 (± 1.0)	83.2 (± 1.0)
	R	81.8 (± 2.2)	84.4 (± 0.5)
	F	81.2 (± 1.3)	83.8 (± 0.4)
Overall (n=511; 551)	P	79.5 (± 1.1)	78.9 (± 0.8)
	R	78 (± 2.1)	79.5 (± 0.6)
	F	78.4 (± 1.4)	79.1 (± 0.7)

Table 4.6: Final NER performance with one standard deviation across the sub-tasks of software and metadata recognition, software type classification, and mention type classification. The error bars indicate the standard deviation of repeated training runs. P=Precision, R=Recall, F=FScore.

Generalization performance

Generalization performance was assessed on the default model in a five-fold cross-validation on unknown full-text articles. It has the goal to assess potential biases by underlying domain, journal, and restriction to methods sections. The result of the five-fold cross-validation are depicted in Figure 4.7. In summary, the average performance across all five folds is P=75.1%, R=83.1%, and F=77.9%, which is notable lower than the estimated prediction performance, with a large difference in Precision. In the results, strong differences in performance are observed between the individual folds, with values ranging from F=92.9% (±1.0) in fold two to F=66.7% (±4.3) in fold three. As for the overall, performance, a particularly low Precision of P=53.8% (±6.1) can be observed at fold three as compared to P=92.4% (±1.0) at fold two.

As the results suggest strong differences in performance between specific articles, the errors were further manually inspected. It was found that individual articles have a high impact on the error, and that confusion between method names and software names can lead to an accumulation of errors. It is often the case that the methodical approach or algorithm is named equally with the software, which is challenging for prediction. An example is provided in Listing 19 which shows excerpts from an article, where initially a methodical application of WGCNA is described in general, without connection to a software, while later in the text the implementation of the analyses with an equally named R package *WGCNA* is described[5].

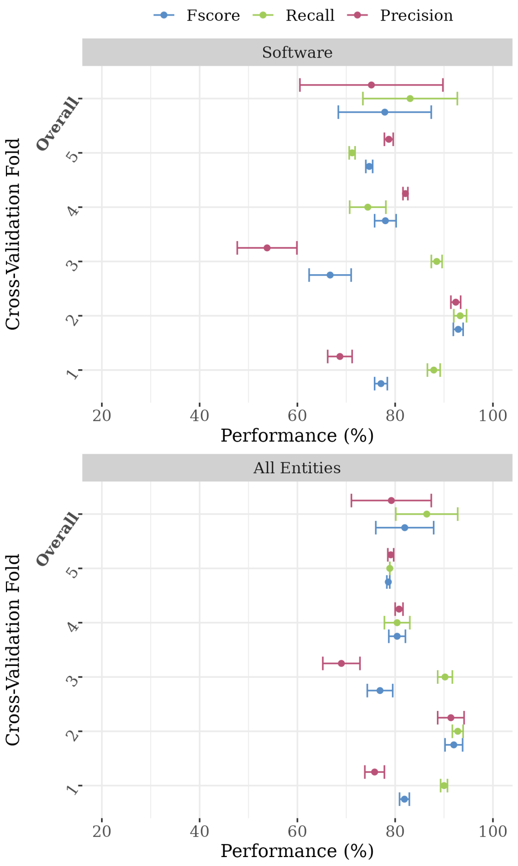


Figure 4.7: Generalization performance of the BERT default model toward PMC full-text articles in a five-fold cross-validation. The error bars indicate the standard deviation of repeated training runs, except for **Overall** where they indicate the standard deviation between the cross validation folds. Note that the axis starts at 20 in this plot.

```
Using a newly-available ovine array and weighted gene co-expression network analysis (WGCNA)
on differentially expressed genes (Additional file 1: Figure S1), we identified co-expressed
genes in different regions of the ovine fetal brain, from mid-gestation to one day of postnatal
life. WGCNA determines
pair-wise correlations between gene expression profiles to create modules
(clusters) of co-expressed genes (Additional file 2: Figure S2).
[...]
All the analyses were performed with the WGCNA package for R software.
```

Listing 19: Illustration of a challenging software prediction for *WGCNA*, highlighting potential confusion between algorithm or method name (🟦) and software implementation (🟩) [ID: PMC5574543, [223]]. Note that software R is also mentioned but not highlighted in the example.

[5] Note that this article is not actually part of the test set, but was chosen as suited illustration of the problem.

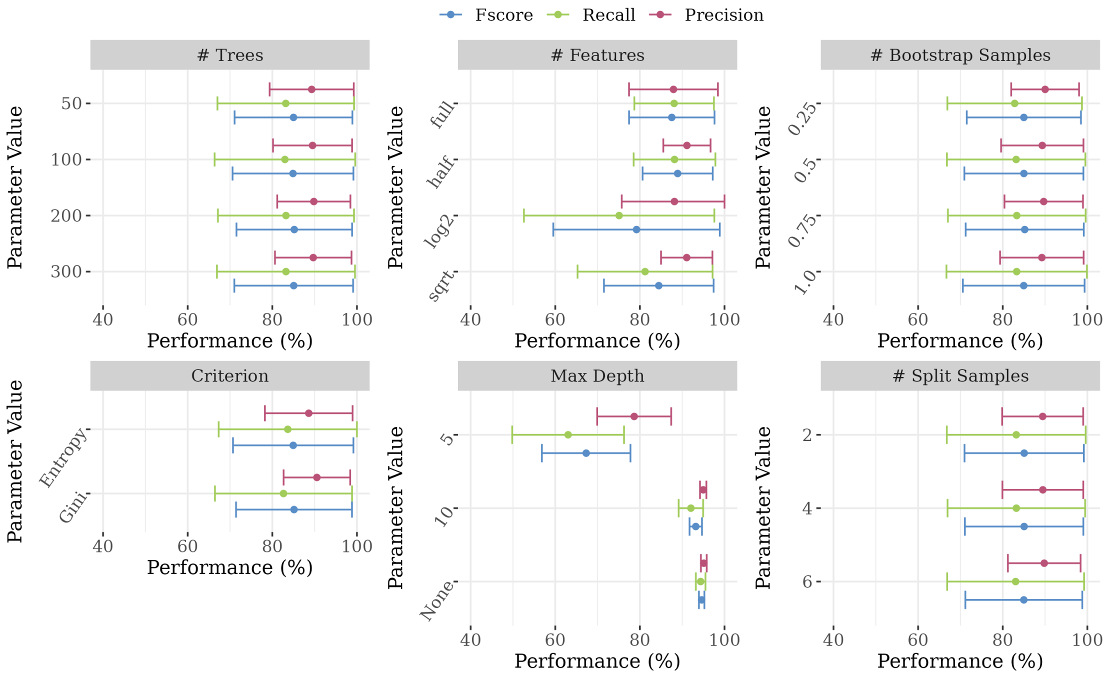


Figure 4.8: Hyper-parameter optimization results for Random Forest RE models on the development set. All plots show the mean performance value and one standard deviation of varying a specific parameter across all settings of the remaining parameters. Note that the performance axis starts at 40.

4.5.2 Relation Extraction

The hyper-parameter optimization results for the RE model are summarized in Figure 4.8. In difference to the NER models, the runtime is lower, and a full grid-search of parameters was performed. Therefore, the depicted values show the performance for a specific parameter across all possible settings of the remaining parameters to see if they have a general influence[6]. It is found that most parameters do not have a notable overall effect on the performance, which is true for *# Trees* (number of decision trees), the used impurity *Criterion*, the *# Bootstrapping Samples*, and the number of samples that are split at leaf nodes (*# Split Samples*). Only the *Max Depth* has an influence on performance with higher depth up to no restriction leading to better results, where a performance of P=78.7% (±8.8), R=63% (±13.2), and F=67.3% (±10.5%) is reached when the depth is restricted to d=5 and P=95.1% (±.7), R=94.4% (±1.1), and F=94.6% (±.6) for unrestricted growing. Another observable effect is that the variance decreases with respect to Recall and FScore when full or half of available *# Features* are used, while it is higher when the feature size is further reduced by using the logarithm or square-root.

Parameter	Value
# Trees	100
Criterion	Gini
# Features	half
# Bootstrapping	1.0
Max Depth	None
# Split Samples	4

Table 4.7: Final Random Forest RE model hyper-parameter selection.

The final model was then chosen based on the highest performance in five repeated runs based on the grid-search. The corresponding configuration is summarized in Table 4.7. The performance of the final RE model is illustrated in Figure 4.9 with detailed performance values provided in Table 4.8, including results for the development set. High recognition rates are achieved, with an overall FScore

[6] Varying parameters from the default values, as for NER models, shows the same influences.

of F=93% and good performance for a majority of recognition targets, i.e., $F \geq 90\%$ for all targets except Release (F=70.8%), PlugIn (F=77.6%) and Specification[7] (F=71.6%). This indicates that relations between software are the most challenging classification targets. In comparison to the development set, a small drop in performance is observed from F=95.5% to F=93%.

The generalization performance regarding PMC OA full-text articles was also assessed for RE. The corresponding five-fold cross-validation results are summarized in Figure 4.10. The overall RE performance is found to be at a generally high level with an average value of P=95.1%, R=95.7%, and F=95.1%. Some variation between F=92.8% (±0.5) and F=99.4% (±0.4) is found, but at a notably lower level than observed for NER.

Target		Test Set	Devel Set
Developer (n=105; 106)	P	97.2 (±0.8)	94.3 (±0.4)
	R	90.6 (±1.9)	97.7 (±0.9)
	F	93.7 (±0.9)	96.1 (±0.5)
Version (n=135; 171)	P	96 (±0.6)	98 (±0.3)
	R	96.1 (±0.3)	98.2 (±0.1)
	F	96.1 (±0.5)	98.1 (±0.2)
Release (n=7; 11)	P	100 (±0)	91.7 (±0.1)
	R	77.1 (±7.9)	100 (±0)
	F	86.9 (±5)	95.7 (±0.2)
URL (n=61; 52)	P	96.8 (±0.1)	93.5 (±0.9)
	R	98.3 (±0.2)	93.4 (±1.1)
	F	97.6 (±0.2)	93.4 (±0.5)
Citation (n=66; 129)	P	94.5 (±1.5)	97.4 (±0.5)
	R	86.7 (±1.6)	93.7 (±0.4)
	F	90.3 (±0.5)	95.5 (±0.4)
AltName (n=2; 7)	P	100 (±0)	100 (±0)
	R	100 (±0)	85.8 (±0.1)
	F	100 (±0)	92.2 (±0.1)
Abbrev (n=10; 12)	P	100 (±0)	100 (±0)
	R	90 (±0)	100 (±0)
	F	94.8 (±0.1)	100 (±0)
Extension (n=10; 5)	P	100 (±0)	100 (±0)
	R	81.9 (±0.1)	100 (±0)
	F	90 (±0)	100 (±0)
License (n=9; 10)	P	100 (±0)	100 (±0)
	R	100 (±0)	90 (±0)
	F	100 (±0)	94.8 (±0.1)
PlugIn (n=27; 27)	P	86.3 (±0.2)	80.6 (±1)
	R	70.3 (±0.2)	77.7 (±5.2)
	F	77.6 (±0.2)	79.1 (±3.2)
Specification (n=8; 7)	P	70.8 (±5.9)	85.8 (±0.1)
	R	72.5 (±5.6)	85.8 (±0.1)
	F	71.6 (±5.4)	85.8 (±0.1)
Overall (n=440; 537)	P	95.4 (±0.4)	95.7 (±0.3)
	R	90.8 (±0.7)	95.1 (±0.3)
	F	93 (±0.3)	95.5 (±0.4)

Table 4.8: Final RE performance with one standard deviation for repeated tests.

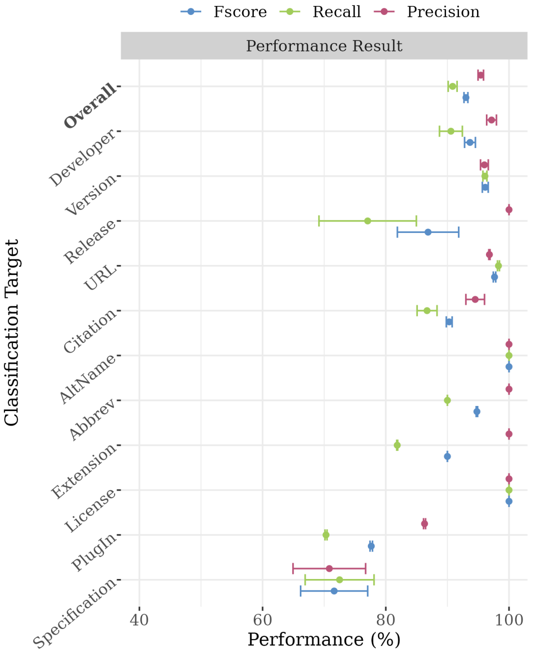


Figure 4.9: Final RE performance. The plot shows the mean performance value and one standard deviation of repeated tests. Note that the axis starts at 40.

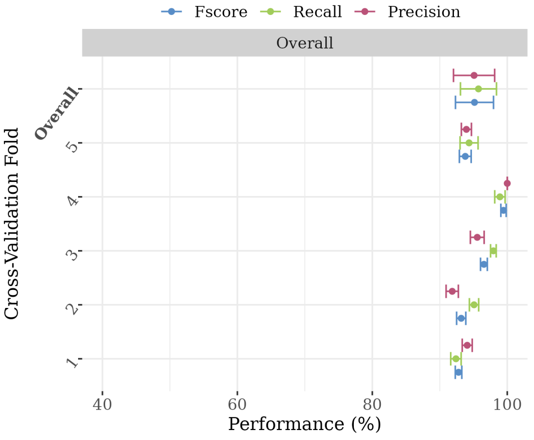


Figure 4.10: Generalization performance of the default Random Forest RE model toward PMC full-text articles in a five-fold cross-validation. The error bars indicate the standard deviation of repeated training runs, except for **Overall** where they indicate the standard deviation between the cross validation folds. Note that the axis starts at 40.

[7] A specification describes cases in which two mentions have the same identity.

		Test Set		Devel Set		Train Set	
Task		Rule	Auto	Rule	Auto	Rule	Auto
Plain	Precision	94.2	100	96.4	99.7	97.2	98.2
	Recall	88.8	93.1	89.2	94.5	91.4	94.9
	FScore	91.4	**96.4**	92.7	**97.1**	94.2	**96.5**
100k	Precision	92.4	94.1	96.3	98.7	94.0	97.9
	Recall	88.6	93.2	90.2	95.1	92.2	96.1
	FScore	90.4	**93.7**	93.1	**96.9**	93.1	**97.0**

Table 4.9: ED performance comparison between rule-based (*Rule*) and automatic (*Auto*) approach across all sets for plain setting considering only labeled data, and 100 k setting, including software mentions from 100 k articles on which NER and RE prediction was applied.

4.5.3 Entity Disambiguation

As described, ED is evaluated in two different setting. Once on annotated data only, and once including 100 k additional articles tagged by the NER and RE models to increase data size and to add noise labels. For the automatic weighting of rules, the perceptron performance is assessed first, separately from the actual clustering. Based on the training threshold of t=.5 and including augmented data in the test set, the employed perceptron model achieved a performance of P=96%, R=90% and F=93%. The thresholds for automatic weighting were then determined for both test settings at values of t_{plain}=.2270 and t_{100k}=.0378, with the threshold of the 100 k setting being notably lower, due to the additional links occurring through the larger data size, as illustrated in Listing 18.

The results of the ED clustering are summarized in Table 4.9. The development set performance is used to select the best of the two models while the test set allows a performance estimation for the selected model. Initially, the development set performance is used to select the approach, before the performance is estimated based on the test set. In the plain test setting the automatic weighting outperforms the rule-based approach with F=97.1% compared to F=92.7% on the development set. In the 100 k setting the automatic weighting also achieved higher performance with F=96.9% compared to F=93.1% on the development set. Further, no strong performance differences are observed between both evaluations on the development set (<0.5 pp). The prediction performance is, therefore, estimated for the automatic weighting approach on the test set with values of P=100%, R=93.1%, F=96.4% in the plain setting and P=94.1%, R=93.2%, F=93.7% for the 100 k setting. Here, a performance difference is observed between the two settings, with the Precision being 5.9 pp and FScore 2.7 pp lower in the 100 k setting, but remaining at a high level. Overall, both approaches were found to perform well, with the automatic approach consistently outperforming the manual approach by a small margin. Therefore, it is selected as the preferred approach. However, it should be noted that the manual approach is more run-time efficient and does not require an adjustment to new data. Therefore, it might be the better choice when available hardware is limited or the number of input data grows too large.

4.6 Limitations

The general limitation are the underlying data. A discussion of the corresponding biases towards Life Sciences and open-access publication is given in Section 3.4, while the potential bias towards an overrepresentation of software in training and test setting has been described in Sections 3.4 and 4.2.3. While the extent of a domain and open-access bias cannot be accessed based on the available data alone, a potential evaluation bias by the sampling method was systematically assessed by testing the generalization to unknown PMC OA full-text articles in the outlined cross-validation setting. It was found that the generalization test for NER exhibits a lower performance than the same model in the standard test setting by a margin of 11.2 *pp* in Precision, 6.5 *pp* in Recall, and 10 *pp* in FScore. This confirms that an evaluation bias does exist in the model. While the main bias can be attributed to the evaluation on the *Creation_s* set in which software is strongly overrepresented, part of this bias might result from a domain difference between the largest training set of *PLoS_m* which has an interdisciplinary focus, and the test set of *PubMed_f* which is sampled from Life Sciences in general. However, it is crucial to include *Creation_s* in the evaluation setting because otherwise the classification of software *Creation* and *Deposition* practices could not be evaluated. The a specific generalization test for RE exhibits no lower performance, while a test for ED could not be performed as described in Section 4.4.3.

Another limitation of the dataset is that some metadata, software types, and mention types are rare (see Section 3.4), making their extraction challenging for automatic approaches. It was observed that all classes were successfully learned by the model, including the ones with only ≈50 overall samples. It is argued that the ML models can represent these classes based on only few training samples because the extraction of metadata is a comparatively easy task. This is the case because metadata can only appear in the direct context of software mentions, and a sufficient number of training samples is available to train the software extraction itself. However, the results also show that, in general, metadata with fewer samples (e.g., Extension or Abbreviation) is extracted with lower performance than metadata with more samples (e.g., Developer and Version). Furthermore, the evaluation for these classes is less reliable as fewer test samples are available. In summary, it is necessary to increase the available ground-truth data in future work and to perform further evaluation regarding the extraction of rare metadata.

In general, high performance was observed for all ML methods employed for IE and ED. However, it is important to consider that error propagation exists between them. The given evaluation for software and mention type classification does take error propagation into account, but the results for RE and ED do not. Therefore, the performance of RE applied on noisy NER results is expected to be lower than the performance of F=.94% found when tested on gold standard NER results. For ED the effects of false positive entities were modeled by data augmentation, but it is not possible to estimate whether this completely suppresses their effect. Further, false negatives resulting from NER are inherently missing in the ED step. Moreover, evaluation for ED has proven to be challenging and the gold standard dataset alone is no good predictor for performance on large-scale ED. Therefore, a second, adjusted evaluation method was employed to take large-scale data into account, however, further systematic evaluation is required for software ED as this is the first work and dataset systematically assessing the problem. In summary, it can be expected that performance will be lower than the estimated performance for both RE and ED due to error propagation, however, both methods are expected to still perform well during prediction as their base performance is high.

4.7 Discussion

A pipeline for automatic software recognition was established. It splits the problem in three steps, starting with NER of software and its metadata and includes a classification of software and mention type, all integrated into an NN implemented with SciBERT. Then the relation between multiple software and metadata is assessed by RE implemented by a Random Forest classifier. Lastly, ED is performed based on agglomerative clustering of sample distances calculated by an automatic weighting of manually generated rules. All methods were found to perform well, while some limitations have been outlined above. All individual steps of the pipeline are discussed in the following.

Different state of the art NER models have been implemented and compared for the given problem. Regarding software recognition, a final recognition of F=86.6% FScore was estimated on the test set using a SciBERT model. In general, BERT-based models were found to outperform Bi-LSTM-CRF models by a considerable margin of ≈8 *pp* during model selection on the development set, illustrating the effectiveness of SciBERT for mining scholarly documents. This also represents an increase over the previous automatic approaches by Pan et al. [205] with F=58%, Duck et al. [78] with F=67%, and Lopez et al. [173] with F=71%. While a further generalization test shows a performance of F=77.9% and suggests that the performance might be overestimated, as discussed in Section 4.6. Based on the generalization test, the performance is closer to the results achieved by Lopez et al. [173] who use SciBERT in combination with a CRF, but still ≈7 *pp* higher. In subsequent work, higher performance values were reported, with F=92% in the work of Istrate et al. [128] using SciBERT trained on Softcite. However, it was argued that these results might overestimate performance since they are trained and evaluated with an overrepresentation of software because they considered positive paragraphs instead of full-text documents [24]. Most recently, a better performance was reported using SciBERT with a CRF on Softcite v2 with a performance of F=81%, which is a ≈3 *pp* improvement over the generalization results reported here. Overall, it should be noted that the results outlined above are based on different datasets and not directly comparable.

High performance is achieved regarding the identification of software metadata (F=86.2%), software type (F=73.3%), and mention type (F=78.4%), with the results taking error propagation into account. In the generalization test, software metadata is also recognized with lower performance but by a smaller margin of 4.2 *pp* (F=82%). For context classification, the generalization test cannot be reasonably performed because the mention types of *Creation* and *Deposition* are rare in all subsets of SoMeSci except for *Creation_s*. Only the prior work of Lopez et al. [173] considers software metadata in terms of version, developer, and URL, reporting an overall performance of F=74.6%, which is 5.4 *pp* lower than the generalization results achieved here. The subsequent work of Bassinet et al. [24] also extracts metadata in terms of version, developer and URL with an overall performance of F=81.6%, which is at a similar level as the performance reported here with a performance difference <0.5 *pp*[8].

This work is the first that considers automatic classification software mentions according to both, software and mention type. Subsequent work of Bassinet et al. [24] reports a performance of F=80.3% for software type classification, which is an 7 *pp* improvement. However, the considered types cannot be directly matched up due to annotation differences, hindering a comparison. The results for context classification reported here, show that software type *PlugIn* and mention type *Allusion* were extracted with lower performance as other types. In both cases the lower performance is mainly due to confusion with another class (*Application* and *Usage*) with corresponding higher prior probability but a difficult to distinguish context. This is consistent with the results of the manual annotation (see Section 3.4), where annotation IAA was also found to be lowest for these classes. *PlugIn* identification is particularly challenging when *PlugIns* are mentioned on their own without the host-software. During annotation the difference was established based on the use of external knowledge with potential errors discovered by the sequential annotation pipeline. Future methods for software type classification could utilize external databases (e.g., DBpedia) and package repositories (e.g., CRAN) to improve the classification.

An influence of the pre-trained model on performance was observed, offering potential for further improvement, particularly on the given data selection. PubMed based BERT models perform slightly better than SciBERT. PubMedBERT might be better suited for the given data because it

[8] The work of Istrate et al. [128] does not report results for metadata extraction.

was specifically trained on PubMed data, while SciBERT is set in other domains, which could lead to more general representations. Therefore, SciBERT was chosen for prediction to allow a potentially better generalization to other domains in future applications. However, it might make sense to perform further tests with respect to model initialization for pre-training based on the findings regarding PubMedBERT. Further, considering models with higher capacity might lead to better results, as it was found that larger model size could have an influence on performance in the comparison between BioBERT (S) and BioBERT (L). However, implementation of these comparisons would require considerable additional computational resources and is out of the scope in this work, because each of the models should be individually fine-tuned to achieve the best comparison.

It was found that the proposed multi-task learning setup does not influence the overall performance. However, the approach still offers a benefit in terms of run-time and is, therefore, utilized in the proposed model. The corresponding hyper-parameters also do not have an influence on performance. Regarding the weighting of individual tasks, this supports the hypothesis that further optimization in this regard is not required because all tasks are optimized based on the same loss metric at a comparable scale.

RE for software mentions has, to the best of my knowledge, not been evaluated as part of any scientific investigation regarding software mentions because corresponding ground-truth data is only available in the established corpus. The RE performance observed for the used Random Forest classifier is high with a value of F=94%, with a better Precision of P=95.4% and lower Recall of R=90.8%. In general, RE performs well for additional information related to software, but the prediction of relations between software, e.g., *PlugIn-of*, was found to be more challenging (F=71.6–77.6%). This was expected since, by definition, additional information is always related to another entity, while two software entities are not necessarily related to each other making their prediction more difficult.

Two different algorithms were proposed to solve the problem of software ED, evaluated in two different test settings. High performances were found for both methods with the automatic weighting of manually generated rules performing better and achieving a performance of F=96.4% on ground-truth data, and F=93.7% in the 100 k large-scale test setting including noisy data. While the prior work of Lopez et al. [173] and the subsequent work of Istrate et al. [128] considered EL of software (see Section 2.3), no work has considered a full ED of software mentions, beyond matching of equal names. Therefore, a comparison to state of the art results is not possible.

ED was found to be a challenging problem because the density of the feature space changes with the number of samples, influencing the distance based clustering. In small datasets, only few spelling variations (and other features used for ED) of software exist; this number increases with the size of the dataset. This means that finding reliable boundaries between different software gets more difficult with increasing dataset size because rare spelling variations (and other features) of software with similar names tend to overlap stronger. In the given case, the entire SoMeSci corpus contains 3718 software mentions, when during prediction millions of software mentions have to be considered. To recreate this effect for the training data, a large set of augmented, fictional software names was included when training the perceptron for the automatic weighting approach. With respect to evaluation, the negative effect of increasing sample size on the ability of finding reliable boundaries between different software prevents the transfer of quality statement from training to inference dataset. To counteract this effect, the 100 k test setting was developed by combining the manually disambiguated ground-truth data with part of the inference dataset for which NER and RE have been applied. The threshold for clustering is then determined on the combined data, and the quality is evaluated by clustering the data together. This approach leads to a larger sample size, includes real false positives, but does not shift the objective of clustering the labeled data.

The actual clustering step during ED is based on single linkage. It was selected due to computational and space complexity because it enables an efficient implementation when distance between all pairs is pre-computed and sorted up to a given threshold (see Section 4.4.2), but is known for semantic drift away from cluster means. However, an initial evaluation showed only marginal differences between single and average linkage based clustering for ED, which seemed sufficient for the task at hand.

Reliable Method for Software Mention Extraction

Overall, the outlined IE pipeline achieved superior recognition rates compared to previous, automatic methods for software mentions in scholarly publications, making large-scale analyses of software mentions in scientific literature more reliable. It is, therefore, considered as suited to perform automatic large-scale analyses. However, errors are still contained in the results of the IE pipeline. Therefore, large-scale analyses should be designed to not concern software on mention basis, but software mentions on article level. This is assumed to make the results more robust due to aggregation over mentions. The aggregation also partially alleviates the confusion between algorithm and implementation, as illustrated in Listing 19, by disregarding false positives in the aggregation step. Further, the Recall is expected to improve on article level as one positive mention is sufficient to identify a software. Lopez et al. [173] explicitly assess the benefit of aggregated analyses and find that the performance results improve by a margin of 5.4 *pp* from F=71% to F=76.4% when switching from mention to article level.

4.8 Application

To perform large-scale analyses, the IE pipeline, consisting of NER, RE, and ED, was applied on the article set described below with additional information on articles gathered to enable further analyses. Since the work outlined in this thesis was performed iteratively, the large-scale pipeline was not applied based on the entire results outlined in Chapter 3 and 4. Specifically, minor changes have been made to the data annotation in the scope of iterative reviewing, and the model was applied as described in Schindler et al. [245], with a different initial learning rate for fine-tuning. Since the made improvements are small they do not justify a re-execution of the entire pipeline in the context of sustainability, because it would require significant computational resources and high calculation time. Instead, it is argued that the data should be upgraded in a future extension of the large-scale analyses covering a broader article range and updating the analyses with recent publications.

4.8.1 Dataset and Information Enrichment

The PMC OA subset was used as a data basis for the large-scale analysis same as for the initial data annotation (see Section 3.2.1) due to the advantage of providing articles as JATS XML documents with licenses allowing a full republication of labeled data. Here, all available data as of January 22, 2021 was used in the analyses, which amounts to a sufficient dataset for large-scale analyses with a total of 3,215,396 articles, obtained via bulk download from PMC[9]. All files were preprocessed equally to the training data, however, some files from the set had to be removed because the full-texts were not available as JATS but only as PDF documents. As argued in Section 3.2.1, PDFs are not considered here, because text extraction from PDFs introduces artifacts in scientific articles caused by elements such as headers, footers, page numbering, or other layout elements. Therefore, a total of 3,036,913 (94.4%) articles is utilized.

[9] https://www.ncbi.nlm.nih.gov/pmc/tools/ftp/, accessed 16 March 2024.

Additionally to the article full-text, metadata on articles was gathered to enable analyses dependent on publication time, scientific domains, and bibliometrics. Information on the publication date and the publishing journal were directly extracted from the JATS document. Information on article references could also be extracted from the JATS document as done in Section 3.2.3, but were omitted as they were not included in the IE pipeline as argued in Section 3.5. To further cover bibliometric and domain information, data from PubMedKG [313] was integrated (PKG2020S4 (1781-Dec. 2020), Version 4 [314]). It includes Scimago data on the Scimago Journal Rank (SJR) and related journal domains. Moreover, it includes citation information for articles originating from PubMed and Web of Science, which were used to generate citation counts. For integration of PubMedKG, PMC identifiers were matched against the PubMed identifier used in PubMedKG based on PMC's mapping service[10].

Lastly, information on software availability was added to enable analyses on the state of free and open source software in science, as defined in Section 1.1. The information on both aspects was manually annotated including, if applicable, the corresponding *open source license*. The manual annotation was necessary since the information cannot be systematically obtained from available sources. As described in Section 3.3, the coverage on software in existing knowledge bases is sparse, and the specific aspect of availability can additionally be missing. Since the manual annotation requires considerable effort, only a subset of all software could be annotated. To make reliable statements the most common software was selected with the goal of achieving an annotation coverage of 50% of extracted software mentions, resulting in a total of 428 software. The effort of the annotation strongly varied between the specific software. While details for availability information and license can easily be obtained for software published in repositories such as Github, information on software published in custom repositories or by commercial publishers is often difficult to obtain, particularly for older software without active maintainance. The overall annotation effort is estimated with two minutes per software, which leads to a total annotation effort of $\approx 14\,h$.

[10] https://www.ncbi.nlm.nih.gov/pmc/pmctopmid/, accessed 16 March 2024.

5 | Data Model for Software Mentions in Scientific Publications

The content of this section is based on the following publications:

> David Schindler et al. "Investigating Software Usage in the Social Sciences: A Knowledge Graph Approach". In: *The Semantic Web – ESWC 2020*. 2020, pp. 271–286. DOI: 10.1007/978-3-030-49461-2_16 [251], establishes an initial data model for software in scientific publications.

> David Schindler et al. "SoMeSci- A 5 Star Open Data Gold Standard Knowledge Graph of Software Mentions in Scientific Articles". In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. CIKM '21. Virtual Event, Queensland, Australia, 2021, pp. 4574–4583. DOI: 10.1145/3459637.3482017 [244], establishes a data model for textual annotation of software in science.

> David Schindler et al. "The role of software in science: a knowledge graph-based analysis of software mentions in PubMed Central". In: *PeerJ Computer Science* 8 (2022), e835. DOI: 10.7717/peerj-cs.835 [245], establishes a data model for large-scale analyses of software in science.

The previous chapters described how information on software in science was systematically gathered by manual annotation of a ground-truth dataset and by automatic extraction on a large-scale dataset. This chapter describes how the corresponding information can be semantically modeled as a Knowledge Graph (KG) to allow a FAIR publication, and enhance aspects such as findability, accessibility, and interoperability, particularly with existing research KGs, described in Section 2.2.1. Therefore, this chapter describes a data model for software mentions in scientific publications, its implementation with suited RDF/S vocabulary, and the KG that was built on it and the large-scale extraction described in Section 4.8

The data model was built to represent informal and formal software citation in scientific literature. Therefore, it needs to represent all aspects of software citation in scientific publications introduced in Section 3.1, and requires a concept to model software identities. Particularly, it needs a mechanism to express uncertainty for knowledge about software that was aggregated over different scientific publications, after the software identity was resolved. Moreover, the model needs to represent scientific articles, their corresponding metadata, and their formal references associated with software. Lastly, it needs to consider information provenance and the associated data quality.

In the following, it is first outlined which existing ontologies and vocabularies were chosen to represent the given problem. Then, the data model and its RDF/S lifting is described, including a detailed description of the information aggregation. Next, it is described how the data model was implemented in terms of SHACL, to allows a formal validation of data corresponding to the data model. Lastly, the resulting KG is summarized and the data model is critically reflected.

5.1 Vocabulary Selection

A prerequisite for establishing the data model is to define the semantic representation of the covered concepts. For this purpose, the data model is built upon existing ontologies and vocabularies to improve interoperability with other data sources. Several different ontologies with varying scopes are incorporated in the data model because it covers a wide range of concepts related to scholarly articles, textual data, and software metadata. More general concepts can be covered by widely used ontolgies while specific concepts, such as the representation of software, require explicit search for suited ontologies. In these cases, ontologies were manually identified in the existing literature and compared to determine which is best suited. Additionally, new terms were defined to represent concepts that are not covered in any of the existing vocabularies with respect to the required semantics. When possible they are defined as *rdfs:subClassOf* or *rdfs:subPropertyOf* of existing terms in the introduced vocabularies.

The *schema.org* [99] ontology was included to cover general terms such as URLs. It was selected because it is a widely used vocabulary for general terms, that is also used in other ontologies, for instance, by the *Software Description Ontology (SDO)* (see below), which bases its definition of organization and developers on it. The *Dublin Core Metadata Initiative Terms (DCT)* [63] defines terms for the general description of metadata and is applied several times in the data model to represent metadata and relations between resources. Further, the *Simple Knowledge Organization System (SKOS) [185]* provides terms to model information about scientific domains and their relations, and is used for this purpose in the data model, while *The PROV Ontology (PROV-O)* [157] is used to model provenance of information from different data sources. All three were selected because they are W3C recommendations and cover concepts central to the data model.

The *Bibliographic Ontology (BIBO)* [60] is used to represent scholarly publications and their metadata. However, it does not introduce a semantic distinction between the bibliographic references of an article and a general bibliographic record that describes any given resource and is complete in terms of the covered information (further details on the distinction are provided in Section 5.2.4). Therefore, the *Bibliographic Reference Ontology (BiRO)* [70] is included to specifically represent this difference. Both ontologies were selected based on the review of Ruiz-Iniesta and Corcho [234] comparing existing ontologies for representation of bibliographic information. Further, the *NLP Interchange Format (NIF)* [108] is included to represent text and annotation information in the data model and to integrate statements regarding annotation confidence. In general, *NIF* was developed as an interchange format for NLP tools, language resources, and annotations, making it suited for the representation of texts, its semantic structures and the information contained within it. Furthermore, *Internationalization Tag Set 2.0 (ITS)*[1] [85] allows to model the connection from human language to real world entities. Therefore, it is included in the data model to bridge the gap between NIF and the other vocabularies by modeling entity identities, as described in Section 3.1.3, i.e., it establishes the link between the concept of mention and the concept of entities, such as software.

Software and its metadata also needs to be semantically described in the data model. In the literature, there are several ontologies specifically dealing with the representation of scientific software, namely the *Software Ontology (SWO)* [178], *CodeMeta* [132], *OntoSoft* [91], and *SDO* [89]. SWO [178] is an ontology developed for the description of software that is used to store, manage, and analyze data. It originates from the field of Biomedicine, and focuses on software users rather than describing the software itself, i.e., it models the information processing of software, the data inputs and outputs, and used data formats. CodeMeta [132] is built as a minimal metadata scheme

[1] `http://www.w3.org/2005/11/its/rdf#`, accessed 16 March 2024.

specialized for scientific software and code, and intended for standardized representation of metadata across software publications. It covers a broad range of terms related to software, covering aspects of software development, funding, but also metadata such as version. OntoSoft [91] is also concerned with covering software metadata, with the focus on scientists sharing and using software. Similar to CodeMeta, it covers a wide range of information, aspects of software identity, background, execution, and updates, including metadata such as version. SDO [89] is an extension of OntoSoft, that simplifies the base model, extends the semantic by introducing concepts where OntoSoft used textual descriptions, and adapts terms from CodeMeta. SDO was selected in favor of the other ontologies because it offers the most complete representation in terms of semantics. Particularly, SWO does not reflect software developer information in a suited way, CodeMeta does not introduce a type for software versions but only a property relating it to textual information, and OntoSoft is a direct prior work to SDO, that uses textual descriptions instead of semantic concepts. The aspect not covered by any existing vocabulary is the mention of software within scientific publications, for which terms are newly defined in the scope of the data model.

5.2 Data Model Implementation and RDF/S Lifting

This section introduces the formal data model and describes how all information described in Section 3 and further metadata necessary for large-scale analyses of software in science are integrated in a KG scheme and how the data is lifted into a RDF representation. For this purpose, the covered information itself is described and the terms used to represent them based on the vocabularies introduced above. Since the overall model is complex, it is separated into four semantic parts in the description below: 1. academic publications, 2. bibliographic references, 3. software mentions, and 4. software entities. In the following, the used vocabulary namespaces are abbreviated when describing RDF/S lifting, with abbreviations summarized in Figures 5.1–5.5, and the namespace *http://data.gesis.org/softwarekg/vocab/* with the prefix *skg* covering newly introduced terms. Furthermore, it is divided between optional and required properties, where required indicates that objects must possess the property.

The intention of the data model is to fully cover the practice of software citation in scientific publication, all associated metadata, and the corresponding software entities and their metadata. However, the model is not intended to be exhaustive in its representation of scientific software and does not consider all aspects of software, e.g., SWO [178] does cover further information such as software functions, dependencies or parameters that are not included here. Furthermore, only information necessary for large-scale analyses outlined in this work is reflected for scientific articles and bibliographic references, while external links are used to connect the given KG to other databases to enable further integration in future work.

5.2.1 Modeling Uncertainty

The data model needs to distinguish between the concept of software, an existing entity that scientists employ in their work, and its citation within scientific literature. From here on, the representation of software is referred to as *entity level* and the description of its citation as *mention level.* While the mention of software allows to track where and how software is applied in scientific investigation, knowledge about the identity of software is crucial to asses aspect such as the impact of software. In the scope of this work the available information is gathered on the mention level by text mining scientific publications. This means, all knowledge regarding the entity level needs to be inferred from information on mention level. Note, that the entity level represents the concept of software identity outlined in Section 3.1.3 that was also included in the annotation.

On mention level, software has several metadata associated with it, which was summarized in Section 3.1. In general, software entities have the same metadata with some semantic differences, e.g., while a software mention has a specific version, a software entity has a release history covering all versions throughout the development of the software. Since only the mention level is observable, the information on entity level needs to be inferred by aggregating over all mentions of a specific software. Moreover, the aggregation method needs to account for uncertainty, as information on mention level is subject to uncertainty and can contain errors or contradicting metadata, e.g., the developer of a software can change over time (see Section 3.1) and the software is subject to spelling variations (see Table 3.4). To model the inherit uncertainty on entity level its metadata is modified by a numerical value expressing the certainty of the information. The value is always calculated in the same way, as described below, but based on semantics the modifier can take the meaning of (1) proportion, or (2) confidence. A proportion expresses the case where multiple attribute expressions can be true at the same time, e.g., a software can have multiple valid versions or associated publications, while confidence expresses metadata where only one instance is true, e.g., a software can only have one name or a single license. The details of which metadata is semantically associated with a proportion or confidence is described in Section 5.2.5.

The result of the aggregation is a mapping of existing software, its usage, and metadata, where a certain level of validity on entity level metadata can be expected when working on a large-scale dataset. In general, high validity is expected for commonly mentioned software, while the representation of highly specialized, rarely used software might still be subject to uncertainty. It should be noted that this definition might in some cases not be reasonable, which leads to an extension of the discussion in Section 3.1, where it was argued that it is often difficult to identify the developer of software due to dynamic changes [263, 135]. Due to these reasons, directly modeling a software developer by a confidence value might not be adequate for all software. Here, it is argued that based on the available data it is the best approximation, while further enhancement would require external data. One approach to further improve the representation without external knowledge would be to infer information (i.e., developer or name) on software version basis, making the representation robust to changes over time. However, this would require a higher number of data samples to make reliable statements. Therefore, it is not performed in the scope of this work because the dataset contains many specialized software for which data is sparse.

Implementation

The aggregation of metadata is performed across all mentions of a specific software. To achieve a balanced result, the information is first aggregated within each individual document and then between all documents containing the metadata of interest, assigning equal weight to each document. Hence, the number of times a document contains specific information does not increase its weight in the aggregation. Formally, let $I_{r,x}$ be the set of all forms of a piece of information for a given relation r (e.g., Version) and software x (e.g., *SPSS*). Further, let D be the set of all articles and $m_{r,a,x}$ the mapping of a piece of information $a {\in} I_{r,x}$ (e.g., version *"v20"*) to x under the relation r. The confidence or proportion $c_{m_{r,a,x}}$ is then defined as:

$$c_{m_{r,a,x}} = \frac{1}{\mid \{d \in D \mid m_{r,b,x} \in d, b \in I_{r,x}\} \mid} \cdot \sum_{d \in D} \frac{\mid \{m_{r,a,x} \in d\} \mid}{\sum_{b \in I_{r,x}} \mid \{m_{r,b,x} \in d\} \mid}, \ a \equiv b, \tag{5.1}$$

where $a \equiv b$ signals that both, a and b represent the same type of information, e.g., name, version, or developer. This way a ratio based fair weighting is achieved on mention level and on document level. All values range from 0 to 1 and also add up to 1. Note that the definition of confidences and proportions requires a recalculation when the dataset is extended.

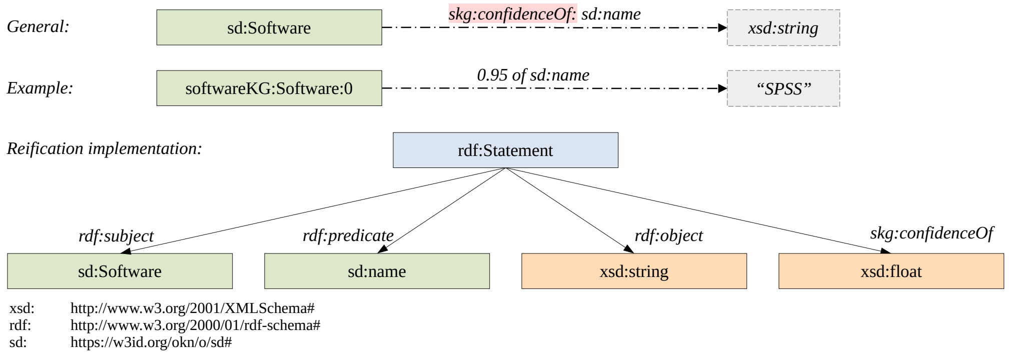


Figure 5.1: Illustration of how edges with a modifier can be represented through reification statements in RDF on the example of software names.

RDF/S Lifting

Metadata on entity level resulting from the aggregation cannot directly be represented in RDF triples because RDF does not allow modified edges. Therefore, they are implemented through *reification* statements, illustrated in Figure 5.1. Instead of representing the relation between software and its metadata directly, a *rdf:Statement* is introduced that relates to the software by *rdf:subject*, the metadata by *rdf:object*, and represents the relation between both by a *rdf:predicate* relation. The modifying information is then also added to the statement as a further property either as *skg:confidenceOf* to express a confidence or *skg:proportionOf* to express a proportion, dependent on the semantics described above. In the given example the software (*sd:Software*) is the subject, the metadata is represented as a string (*xsd:string*), and their relation is by predicate name (*sd:name*). The modifier is provided as a floating point value (*xsd:float*) and as a confidence (*skg:confidenceOf*) to express that a software can only have one name. The modifier properties *skg:confidenceOf* and *skg:proportionOf* are newly defined to represent confidence and proportion of an aggregation statement as described above.

5.2.2 Article perspective

The data model needs to represent scientific articles as they are the main source for software mentions and formal citations. There is some metadata associated with articles that needs to be represented to enable large-scale analyses of software mentions within them. An overview of all types and properties connected to the representation of scholarly articles is given in Figure 5.2. Below, all covered classes, their properties, and relations are introduced.

Articles

The source for software mentions in science are scholarly articles. They are represented by the two different types *bibo:AcademicArticle* and *nif:Context*, with the former semantically capturing the scientific publication and the latter the textual document as natural language data. Several metadata of articles is relevant for unique identification and systematic analyses. The properties *dct:title* and *dct:created* are required metadata and form the minimal requirement for identification of an article by representing its name and publication data. Furthermore, articles have optional identifier properties *bibo:doi*, *bibo:pmid*, *skg:pmcID*, *skg:semanticscholarID*, and *skg:crossrefID* capturing various different identifiers for scientific publications and establishing links to other databases. The most common unique identifier for scholarly articles are DOIs. They are widely adapted by

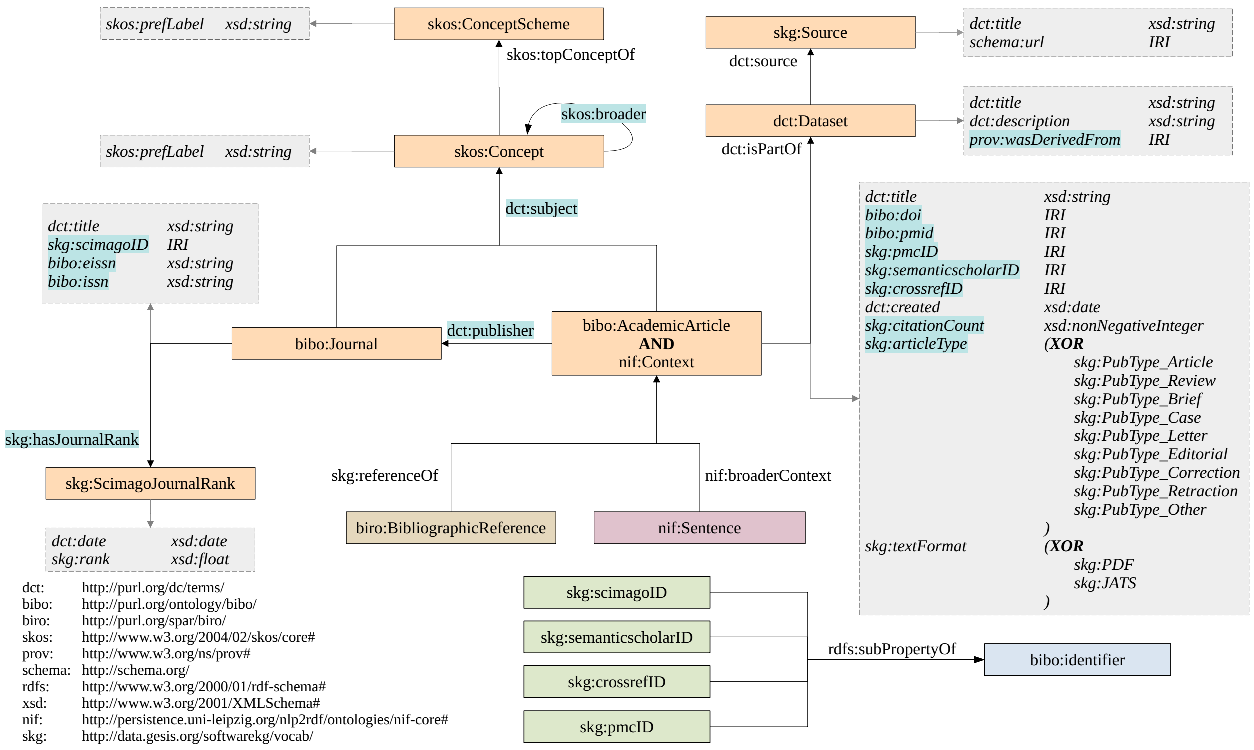


Figure 5.2: Excerpt from the overall data model covering the representation of scholarly publications, journals, research domains, datasets and their associated properties. The connection to other parts of the data model is given by *biro:BibliographicReference* and *nif:Sentence*. Further, the representation of newly introduced properties as *rdfs:subPropertyOf bibo:identifier* is illustrated. Optional properties are highlighted (▨).

scientific publishers and identify articles globally unique. However, older articles or specific article types such as letters or editorials are sometimes not assigned a DOI. In those cases they might be identifiable by specific database identifiers such as PubMed identifiers (PMID), but also in knowledge bases such as Crossref or Semantic Scholar (introduced in Section 3.2.3). The properties *skg:pmcID*, *skg:semanticscholarID*, and *skg:crossrefID* were newly defined as sub-properties (*rdfs:subPropertyOf*) of *bibo:identifier*, to specifically represent PMC, Semantic Scholar, and Crossref IDs, to establish links to external knowledge bases.

Articles can optionally have a citation count, captured by the newly introduced property *skg:citationCount*, that describes the number of citations an article has received, to allow analyses with respect to article impact. Further, they can optionally have an article type (*skg:articleType*) that closer defines the type of article, that might be of interest in specific large-scale analyses. The expected value for this property is one of defined types: *skg:PubType_Article* for scholarly articles, *skg:PubType_Review* for review articles, *skg:PubType_Brief* for brief reports, *skg:PubType_Case* for case reports, *skg:PubType_Letter* for letters, *skg:PubType_Editoral* for editorials, *skg:PubType_Correction* for corrections, *skg:PubType_Retraction* for retractions, and *skg:PubType_Other* summarizing all remaining types. Lastly, articles have the property *skg:textFormat* that captures the format from which their full-text document was obtained. The corresponding values are restricted to only allow specifically defined types *skg:JATS* and *skg:PDF* to capture the JATS and PDF format, respectively (see Section 3.2.1 for background on the formats).

Further metadata for articles is omitted because it is not required for software specific analyses in the scope of this work, e.g., authors, affiliations, or countries. Information with respect to this metadata can be gathered from external sources based on the included identifiers if required in future analyses, e.g., from OpenALEX [220] or other research KGs described in Section 2.2.1. Articles are further related to the representations of sentences (*nif:Sentence*) and bibliographic references (*biro:BibliographicReference*), with respective details described in Section 5.2.4 and 5.2.3.

Journals

Most academic articles, especially in the domain of Life Sciences, are published in scientific journals[2] In general, journals differ in several aspects, e.g., in their publication policies and review requirements. Modeling journals allows targeted analyses towards software mention in specific journals, for instance, to uncover how journal policies impact software citation completeness. In the data model, journals are represented by type *bibo:Journal* and articles are connected to them by the optional property *dct:publisher*. Journals themselves have a title, captured by the property *dct:title* and further optional identifying properties capturing their International Standard Serial Number (ISSN) (*bibo:issn*) and eISSN (*bibo:eissn*), used for electronic publications. Furthermore, a link to Scimago[3] is added through the newly defined property *skg:scimagoID*, which is modeled as a sub-property (*rdfs:subPropertyOf*) of *bibo:identifier*, same as the previously introduced identifiers.

Journals have an optional property *skg:hasJournalRank* relating them to journal rank information represented by type *skg:ScimagoJournalRank*. The rank itself is based on the SJR. Modeling this information allows to incorporate systematic analyses of software citation with respect to journal ranking. The *skg:ScimagoJournalRank* itself is calculated on a yearly basis and, therefore, has the required properties *dct:date* to capture its time frame and *skg:rank* that describes the actual rank obtained from Scimago, represented as a floating point value as in the original Scimago data. Same as for articles, only information immediately required for analyses is included, e.g., the journals publisher is omitted, with further information being identifiable through unique identifiers.

Domains

Both, articles and journals, are set in specific scientific domains, which are known to influence software usage, as outlined in Section 2.1. Domains are represented as type *skos:Concept* to which articles and journals are related by the optional property *dct:subject*. Since domains can be defined with different levels of specificity a *skos:Concept* can optionally be related to another *skos:Concept* by property *skos:broader* to indicate that the second is a top-level domain of the first. Every domain has a required property *skos:prefLabel* to provide a human readable label describing the domain. Furthermore, following SKOS best practices, an overarching *skos:ConceptScheme* is introduced to which each *skos:Concept* is related by required property *skos:topConceptOf*, indicating that all introduced domains belong to the same domain classification scheme used specifically within this data model. It is also assigned a human readable name by required property *skos:prefLabel*.

Dataset and Source

The data model further represents the data source from which scientific articles were obtained. This can be any existing collection of articles, e.g., an existing dataset, a database, or a publisher's collection. Adding information on source allows to capture the provenance of the data creation. Explicit modeling of the source further allows to integrate data of different origin, while ensuring reliability of analyses based on the data.

[2] Conference publications could be modeled similarly, but were not considered here.

[3] https://www.scimagojr.com/, accessed 16 March 2024.

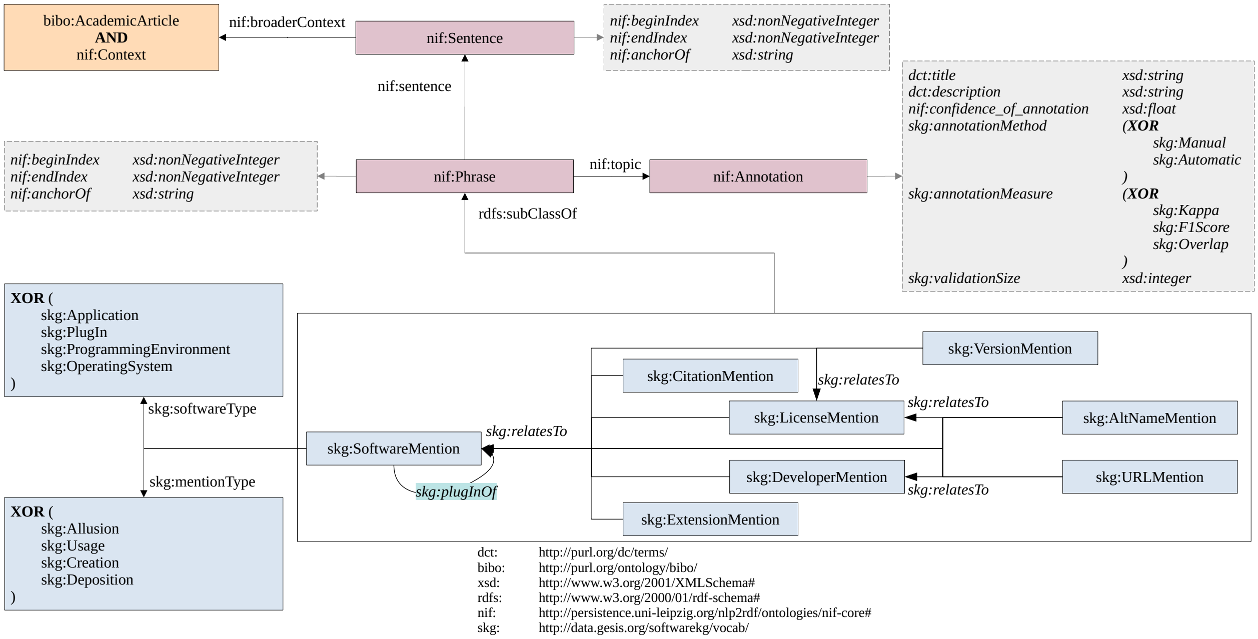


Figure 5.3: Excerpt of the data model covering the representation of textual information for software mentions. The connection to other parts of the data model is given by *bibo:AcademicArticle*. The connection to the software entity representation is omitted due to complexity, but is included in Figure 5.5. Optional properties are highlighted (▨).

Each article is modeled as part of a dataset by the required property *dct:isPartOf* relating an article to a dataset (*dct:Dataset*). The dataset itself has the required properties of *dct:title* and *dct:description* that provide a human readable identifier and description for the dataset. Moreover, it is modeled whether the dataset was based on an existing dataset by the optional property *prov:wasDerivedFrom* indicating the potential source of the original dataset as an Internationalized Resource Identifier (IRI). The source of a dataset is then represented by type *skg:Source* to which datasets relate by the required property *dct:source*. Same as the dataset, it has a human readable title through required property *dct:title*, and a location at which the source can be accessed by property *schema:url* relating it to an IRI.

5.2.3 Annotation perspective

Articles are not only viewed as scholarly publications but as text sources containing information of interest. Therefore, information on software mentions in full-text documents are modeled as NLP data, with the text being split into sentences and phrases, defined as continuous strings in a sentence. Phrases are only represented if they refer to a textual entity of interested as introduced in Section 3.1. Moreover, a concept for annotation quality is included to allow the integration of data from different annotation sources with varying annotation or extraction confidence. The representation of textual information in the data model is illustrated in Figure 5.3.

Sentences and Phrases

An article (*nif:Context*) consists of sentences represented as type *nif:Sentence* and related to their context by required property *nif:broaderContext*. Phrases are continuous strings within one sentence and represent either software or associated metadata. They are modeled by type *nif:Phrase* and relate to the sentence containing them by required property *nif:sentence*. Both, phrases and sentences, have the required properties *nif:beginIndex*, *nif:endIndex*, and *nif:anchorOf*. The former two describe the position of the sentence or the phrase within the respective larger context, while the latter represents the actual string content.

To represent the mention of software metadata in textual documents, all different metadata types described in Section 3.1 are represented by a newly defined type to semantically separate them. All defined types are sub-classes (*rdfs:subClassOf*) of *nif:Phrase*, as described above, to capture their semantics as phrases appearing in a document, and to inherit the corresponding properties of phrases in the data model. In detail, the new terms *skg:SoftwareMention* for mention of software names, *CitationMention* for formal citations, *skg:AltNameMention* for explicit mention of a second alternative name, *skg:URLMention* for URLs, *skg:VersionMention* for versions, *skg:ExtentionMention* for meta-versions, *skg:LicenseMention* for licenses, and *skg:DeveloperMention* for developers, are defined. Note that releases are not explicitly modeled and instead represented as versions in this context. This decision was made because all informally provided release dates identified in SoMeSci are date based versions, e.g., "Excel 2003" or "Matlab R2020a", instead of full dates. All mention types except the name mention (*skg:SoftwareMention*) have a required property *skg:relatesTo* that relates them to another mention to which they refer. This is semantically modeled by defining both, the domain (subject) and range (object), of *skg:relatesTo* as *nif:Phrase* through *rdfs:domain* and *rdfs:range*. Moreover, software mentions can be related to each other with the optional property *skg:plugInOf* describing that one software is dependent on the other software to run, making them related as *PlugIn* and host-software. As before, this is modeled by defining domain (*rdfs:domain*) and range (*rdfs:range*) of *skg:plugInOf* as *skg:SoftwareMention*.

Furthermore, software names (*skg:SoftwareMentions*) have the required properties of *skg:softwareType* and *skg:mentionType* describing the software and mention types introduced in Section 3.1.1. The values for both properties are restricted to newly defined types capturing the exact semantics of the allowed software and mention types. In detail, the software type *Application* is modeled by *skg:Application*, *PlugIn* by *skg:PlugIn*, *PE* by *skg:ProgrammingEnvironment*, and *OS* by *skg:OperatingSystem*. Similarly, the mention type *Allusion* is represented by *skg:Allusion*, *Usage* by *skg:Usage*, *Creation* by *skg:Creation*, and *Deposition* by *skg:Deposition*. Further, the mention representation is connected to the citation perspective through citation mentions (*skg:CitationMention*), described in Section 5.2.4, and connections to the software entity representation through all other introduced mentions (e.g., *skg:SoftwareMention*), described in Section 5.2.5[4]

Source and Quality

Phrases have metadata further describing how they were annotated or extracted. This information is crucial as it determines the annotation quality of all extracted information, because it is subject to errors, regardless if annotated by human or extracted by machine. Detailed modeling of the annotation process can help users to filter information with a sufficient level of confidence for specific analyses. It also allows to integrate data from different sources with different levels of trust into one model.

The annotation information is modeled by type *nif:Annotation* to which phrases are related by required property *nif:topic*. The annotation source has a human readable title and description modeled by required properties *dct:title* and *dct:description*, and a property *nif:confidence_of_annotation* that reflects the specific quality for the given phrase annotation, e.g., a specific IAA value reported in Section 3.2.3. Moreover, it represents the used annotation method by required prop-

[4] These connections are not included in Figure 5.3 as they would make the illustration too complex and difficult to comprehend.

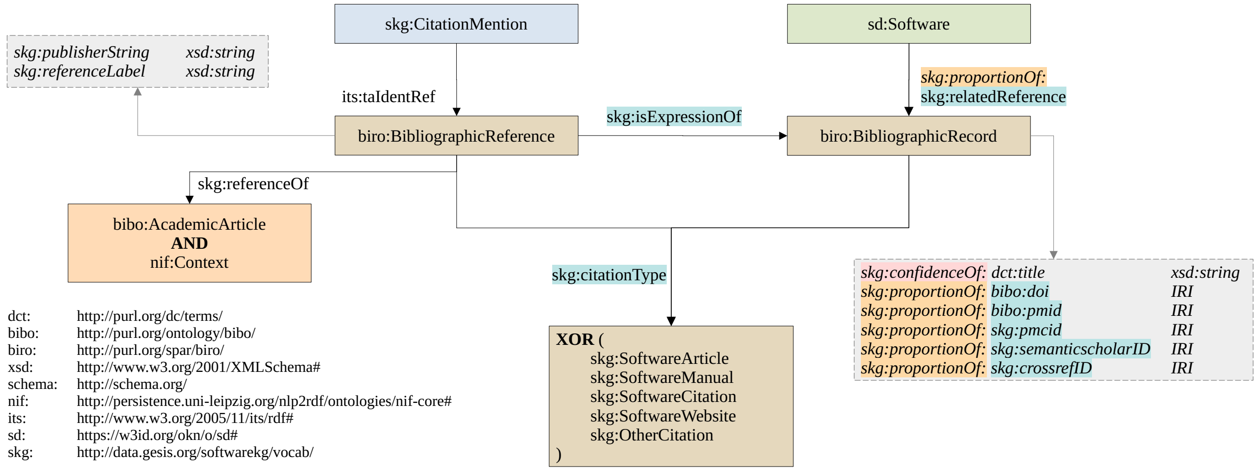

skg:CitationMention
sd:Software
skg:publisherString xsd:string
skg:referenceLabel xsd:string
its:taIdentRef
skg:proportionOf:
skg:relatedReference
skg:isExpressionOf
biro:BibliographicReference
biro:BibliographicRecord
skg:referenceOf
bibo:AcademicArticle
AND
nif:Context
skg:citationType
skg:confidenceOf: dct:title xsd:string
skg:proportionOf: bibo:doi IRI
skg:proportionOf: bibo:pmid IRI
skg:proportionOf: skg:pmcid IRI
skg:proportionOf: skg:semanticscholarID IRI
skg:proportionOf: skg:crossrefID IRI
XOR (
skg:SoftwareArticle
skg:SoftwareManual
skg:SoftwareCitation
skg:SoftwareWebsite
skg:OtherCitation
)
dct: http://purl.org/dc/terms/
bibo: http://purl.org/ontology/bibo/
biro: http://purl.org/spar/biro/
xsd: http://www.w3.org/2001/XMLSchema#
schema: http://schema.org/
nif: http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core#
its: http://www.w3.org/2005/11/its/rdf#
sd: https://w3id.org/okn/o/sd#
skg: http://data.gesis.org/softwarekg/vocab/

Figure 5.4: Excerpt of the data model covering the representation of bibliographic information and the associated properties. The connection to other parts of the data model is given with *bibo:AcademicArticle*, *skg:CitationMention*, and *sd:Software*. Optional (🟦) properties and reification statement confidence (🟥) and proportions (🟧) are highlighted (see Section 5.2.1).

erty *skg:annotationMethod* which is restricted to the specific introduced types *skg:Manual* and *skg:Automatic* representing manual human annotation and automatic extraction by a machine, respectively. The quality metric for the annotation is indicated by property *skg:annotationMetric* restricted to the specific introduced types of *skg:Kappa* representing Cohen's κ, *skg:F1Score* representing the FScore value, and *skg:Overlap* representing simple percentage overlap (see Section 2.4.2 for details on annotation metrics). Lastly, it has the required properties of *skg:validationSize*, representing the number of data points on which the agreement was calculated as an integer value, to allow a better estimation of trust in the validation process.

5.2.4 Citation perspective

The data model needs to represent the bibliographic references of scientific articles because they contribute to software citation practices (see Section 3.3.2). Here, a single entry within the bibliography of an article is referred to as a *bibliographic reference*, and the in-text reference pointing towards a bibliographic reference is called a *citation*. Lastly, the term *bibliographic record* refers to a general record describing a specific resource. This means a *bibliographic reference* and *bibliographic record* can refer to the same object, while the former is bound to an article and is an expression of the latter. This distinction is introduced to model multiple bibliographic records that are related to one software, e.g., a manual, a direct citation, or software articles, and individually cited in one or multiple articles. An illustration of how the model represents bibliographic information is given in Figure 5.4.

An article contains a list of bibliographic references, represented by *biro:BibliographicReference*, where each entry uniquely identifies a scientific resource. Since the reference information is human provided information it is subject to uncertainty, which means that it can be incomplete or even wrong in rare instances. Bibliographic references are related to articles (*bibo:AcademicArticle*) by the required property *skg:referenceOf*. They further have two required properties *skg:publisherString* describing the content of the reference as a string exactly as provided by the publisher, and *skg:referenceLabel* describing the unique citation identifier, referring to the string label used in publications to connect in-text citation and bibliographic references. Moreover, bibliographic ref-

```
PMC3293852:     Boersma P, Weenink D. Praat: doing phonetics by computer (Version 5.1.04). 2009

PMC4726623:     Boersma P, Weenink D. Praat: Doing phonetics by computer; 2015. Available from:
                http://www.praat.org.
```

Listing 20: Examples for two bibliographic references from different articles (PMC3293852 [226], PMC4726623 [300]) referring to the same bibliographic record, corresponding to the representation of software *Praat* (▨). Note that references for a bibliographic record are aggregated over considered samples. In the given example, information from external sources could be integrated because *Praat* is represented in Wikidata[6], however, for software this is in general not possible.

erences have a required property *skg:citationType* that is introduced to distinguish the different citation types explained in Section 3.1.4. Each of the citation types is semantically represented by a newly introduced type: direct software citations by *skg:SoftwareReference*, software articles by *skg:SoftwareArticle*, manuals by *skg:SoftwareManual*, websites by *skg:SoftwareWebsite*, and others by *skg:OtherCitation*. Since citation mentions (*skg:CitationMention*) refer to bibliographic references of an article, their representation is modeled by required property *its:taIdentRef*, representing that the mention is a citation of the bibliographic reference.

A bibliographic record, represented by *biro:BibliographicRecord*, is the global concept of a bibliographic entry that identifies a specific scientific resource. Here, only bibliographic records referring to software as a resource are considered[5]. Bibliographic records are a result of aggregating bibliographic references for a specific software as described in Section 5.2.1 taking into account the citation type of the reference. An example for two different bibliographic references referring to the same bibliographic record is given in Listing 20. Records have a required property *dct:title* capturing a human readable title modified by a confidence (*skg:confidenceOf*) to express the uncertainty about the real title. They further have optional identifiers associated with them, modified by proportions (*skg:proportionOf*), namely *bibo:doi*, *bibo:pmid*, *skg:pmcID*, *skg:semanticscholarID*, and *skg:crossrefID* (see Section 5.2.2). The new optional property *skg:isExpressionOf* is defined to relate bibliographic references to bibliographic records and indicates that the reference is an expression of a record. The new term had to be introduced to cover this connection as BIRO only represents the connection through the common resource of reference and record, which is unsuited as multiple records can exist for one software as described above. Moreover, bibliographic records have a citation type, same as references, which is not modified by uncertainty because it is taken into account during aggregation. Lastly, bibliographic records are connected to the representation of software entities (*sd:software*, see Section 5.2.5) by property *skg:relatedReference*, which models that a record refers to the software. It is modified by a proportion (*skg:proportionOf*) as this connection expresses how often the record is referenced for the software.

5.2.5 Software Entity Perspective

The entity perspective models the knowledge about software and its related entities, where a software corresponds to a real world entity. As outlined in Section 5.2.1 this information is generated by an aggregation process implemented on top of extracted software mentions, and it is, therefore, necessary to model uncertainties through reification statements. The data model for software entities resulting from inference based on software mentions in scientific literature is illustrated in Figure 5.5.

[5] Other references could be modeled in the same way but are not relevant for the conducted analysis of software in science.

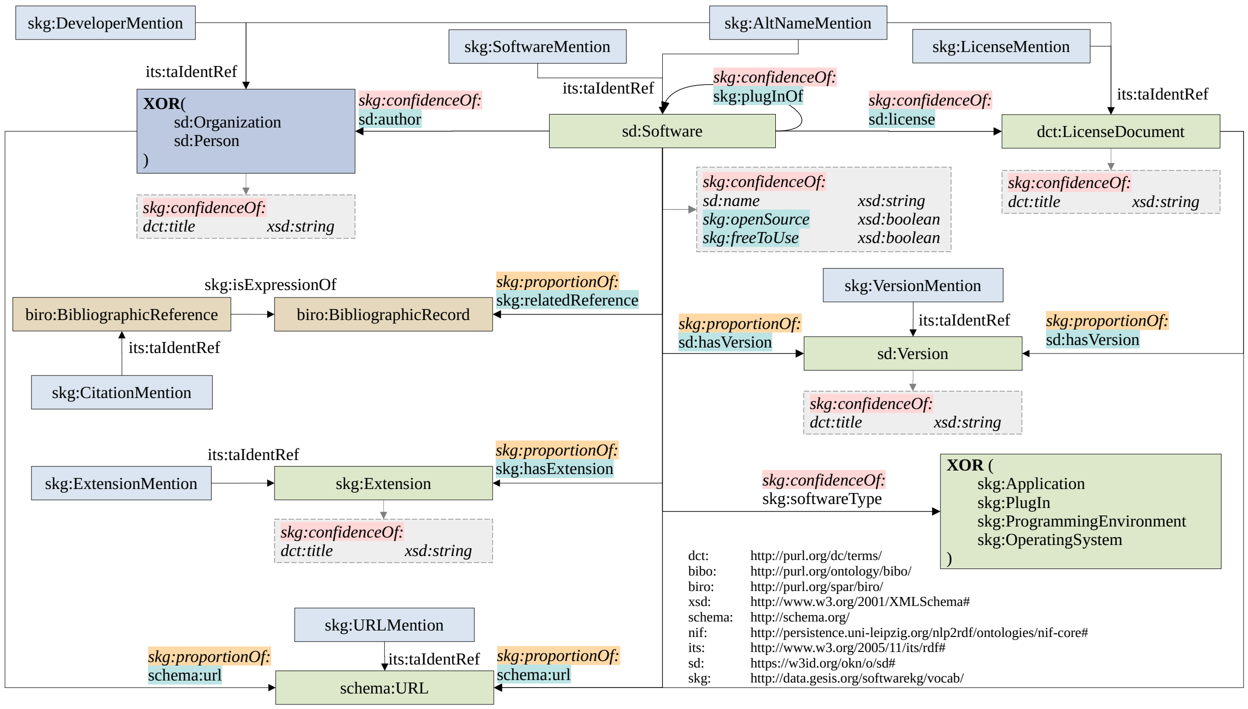


Figure 5.5: Excerpt from the overall data model covering the representation of software entities, their metadata, and the associated properties. The connection to other parts of the data model is given through all types of mentions, *biro:BibliographicReference* and *biro:BibliographicRecord*. Optional (🟦) properties and reification statement confidence (🟥) and proportions (🟧) are highlighted (see Section 5.2.1).

Software

Software entities are represented by type *sd:Software* and are referred to by in-text software mentions (*skg:SoftwareMention* and *skg:AltNameMention*), represented by property *its:taIdentRef* relating a mention to a specific entity. One software can have multiple names referring to it, but it usually has one developer assigned name that is best suited to identify it. Note that this name can be subject to change, for instance, through changes in software developer. The name of a software is represented by required property *sd:name* and the respective uncertainty regarding it is modeled by reification with modifier *skg:confidenceOf*. Moreover, the optional property *skg:plugInOf* (introduced in Section 5.2.3) relates a software to its host-software, another *sd:Software*, and is modeled by a reification with *skg:confidenceOf* because it is a binary relation between two software entities that can either be true or false.

Software has two optional properties *skg:openSource* and *skg:freeToUse* that capture whether the source code of a software is published and whether the software is free to use for academic purposes. These aspects have not been discussed before, as they generally cannot be inferred from software mentions in articles. The information is, however, of interest for analyses and has been included in prior studies, see Section 2.1. It might be possible to infer the knowledge from stated licenses, but theses are rarely present in practice, and usually only stated in cases where software is newly developed. Therefore, it is planned that this information is gathered later on for analyses either by automatic search and linking to other knowledge bases or through manual efforts.

The type of a software is defined in the same way as for mentions through required property *skg:softwareType* with allowed values of *skg:Application*, *skg:PlugIn*, *skg:ProgrammingEnvironment*, and *skg:OperatingSystem*, representing the defined types. Same as other information, the type information is aggregated from the mention level information and is, therefore, represented in a reification statement modified by a *skg:confidenceOf* as only one type can be true for a software.

Metadata

The developer of a software is represented by *sd:Organization*, while SDO intends to describe the software developer as either *sd:Organization* or *sd:Person* depending on the specific software. However, this distinction cannot be made based on the available data because it is not implemented in the IE pipeline. Licenses are represented by *dct:LicenseDocument*. Software (*sd:Software*) is related to its developer by optional property *sd:author* and to license by *sd:license*. The property is optional because it is possible that no knowledge on the developer or license is given, while both are modified by a confidence. The developer and license both have a required property *dct:title* capturing their names, modified by a confidence, in the same way as the name for a software. Also, developer mentions (*skg:DeveloperMention*) and alternative names (*skg:AltNameMention*) relate to a developer by *its:taIdentRef*, and license mentions (*skg:LicenseMention*) and alternative names (*skg:AltNameMention*) relate to a license by *its:taIdentRef* as described for software above. For the remainder of the metadata, their *its:taIdentRef* properties are no longer explicitly described as all function in the same way and are illustrated in Figure 5.5.

While a specific version of a software is used on mention level, a pool of versions, potentially covering the entire release history of a software exists on entity level. A specific version is represented by type *sd:Version* and a software is related to it by optional property *sd:hasVersion*. Since multiple versions of the software can exist the edge is modified by a proportion. The same is the case for extensions, which describe different meta-versions of a software, for instance, a standard and professional version. They are represented by type *skg:Extension* and optional property *skg:hasExtension* modified by a proportion. Same as license and developer, versions and extensions also have a title (*dct:title*) modified by a confidence to express their human readable name as a string value. Further, URLs are represented by type *schema:URL*, where a pool of URLs can be used to identify a software, license, or a software developer. Therefore, the three entities software (*sd:Software*), developer (*sd:Organization*), and license (*dct:LicenseDocument*) are related to URLs by optional property *schema:url* modified by a proportion. Note that formal citations, illustrated in Figure 5.5, have already been described in Section 5.2.4.

5.3 Formal Validation Model

In the previous section the data model for software in scientific publications was described based on suited RDF/S vocabulary. However, that does not allow to represent imposed constraints on the data, concerning optional and required properties, allowed values for properties, and their cardinality. Therefore, a formal validation model is established in the Shapes Constraint Language (SHACL), to explicitly describe these constraints and enable systematic tests of whether data conforms with them. The W3C has two recommendations that allow to define restrictions on RDF graphs, the Web Ontolology Language (OWL) and SHACL. However, OWL is intended for inference extending RDF/S, while SHACL is intended for validation. As the application purpose here, is explicitly a formal validation of the model, SHACL is selected. Note that Shape Expressions (ShEx)[7] can also be used as a validation language for RDF graphs, but is not a W3C recommen-

[7] https://shex.io/, accessed 25 January 2024.

```
skg:SoftwareMentionShape
    a sh:NodeShape ;
    sh:targetClass skg:SoftwareMention ;

    sh:property [
        sh:path rdf:type ;
        sh:hasValue skg:SoftwareMention ;
    ] ;

    sh:property [
        sh:path skg:mentionType ;
        sh:in (
            skg:Allusion
            skg:Usage
            skg:Creation
            skg:Deposition
        ) ; sh:minCount 1 ; sh:maxCount 1 ;
    ] ;

    sh:property [
        sh:path skg:softwareType ;
        sh:in (
            skg:Application
            skg:PlugIn
            skg:ProgrammingEnvironment
            skg:OperatingSystem
        ) ; sh:minCount 1 ; sh:maxCount 1 ;
    ] ;

    sh:property [
        sh:path skg:plugInOf ;
        sh:hasValue skg:SoftwareMention ; sh:maxCount 1 ;
        sh:severity sh:Warning ;
    ] ;

    sh:property [
        sh:path its:taIdentRef ;
        sh:hasValue sd:Software ; sh:minCount 1 ; sh:maxCount 1 ;
    ] .
```

Listing 21: Definition of the SHACL *sh:SoftwareMentionShape* which defines the expected properties for type *skg:SoftwareMention*, their allowed scopes, and cardinality.

dation. The defined validation model is formulated as a closed model, not allowing data entries beyond the explicitly made descriptions. In future extension of this dataset and for integrating more data the definition can either be extended or set to an open model. Since the model includes RDF/S definitions, it is important to consider its interaction with the SHACL validation. Here, the defined SHACL shapes consider the data without triples added by the RDF/S inference step. The main principles of the SHACL model are illustrated in Listing 21 based on the example of the formally defined shape of *skg:SoftwareMentions*, which allows to illustrate several core concepts of the validation model. The entire validation model is made available with the data publication as described in Section 7.

The SHACL model excerpt illustrates how required properties can be defined based on the *rdf:type*, which directly states that a software mention has to be of type *skg:SoftwareMention*. The definition of the required property is made through determining the cardinality at a fixed value of 1 (*sh:minCount 1* and *sh:maxCount 1*). The example further illustrates constraints on property objects by *sh:in*, which restricts the scope of the properties mention type (skg:mentionType) and software type (skg:softwareType) to the newly defined types *skg:Application, skg:Usage, skg:Creation, skg:Deposition* and *skg:Application, skg:PlugIn, skg:ProgrammingEnvironment, skg:OperatingSystem*, respectively. Property *skg:plugInOf* illustrates an optional property as there is no required *sh:minCount*, while it is restricted in cardinality by *sh:maxCount* because a software can only be a *PlugIn* to one other software in the scope of a single mention. Furthermore, the severity of this constraint is set to *sh:Warning* by *sh:severity* because the cardinality of this constraint can be violated by errors in automatically extracted information. Lastly, the definition defines that a software mention has to be related to exactly one software entity by property *its:taIdentRef*, defining the relation between mention and entity.

5.4 Knowledge Graph Summary

The manually annotated gold standard dataset of SoMeSci described in Section 3.3 and the extracted large-scale dataset described in Section 4.8 were integrated into the described data model. The established KG—named SoftwareKG—represents the largest KG of software mentions and related metadata in scholarly publications covering over 300 M triples, and was upon its publication the largest dataset of software mentions in scholarly publications[8]. A summary of the main resources covered by SoftwareKG is given in Table 5.1. It contains 11.8 M software mentions of over 606 k different software automatically extracted from more than 3 M open-access articles from PMC OA. Moreover, information from PubMedKG was integrated to allow bibliometric analyses, as described in Section 4.8.1.

As described, SoftwareKG represents information about software on the mention and entity level. On the mention level, it contains information on >9 M metadata mentions aside from software, e.g., 3, 4 M version mentions. Furthermore, it contains 9, 8 M properties (e.g., skg:relatesTo) which relate different mentions. On the entity level, it contains the same metadata disambiguated through the name, except for software for which the ED approach is described in Section 4.4. This is further illustrated in Table 5.1, where corresponding types on mention and entity level are listed next to each other. It should be noted that the manually annotated data of SoMeSci does cover more information than the automatically extracted data, especially with respect to formal citations, which were not included in the large-scale analyses as argued in Section 3.5. This is also apparent in Table 5.1 where the low number of *biro:BibliographicRecord* results only from manual annotations.

Some information contained in SoftwareKG was omitted in Table 5.1 to achieve better readability by limiting the table to the most relevant information for subsequent analyses. Aside the listed information, SoftwareKG does contain 16 *nif:Annotation* objects representing annotation quality for data on software mentions (as described above). Moreover, the data is structured into five datasets (*dct:Dataset*) capturing the data selection described in Section 3.2.1 and the automatically extracted data. Currently, there is only one *skg:Source* for all included datasets which represents PMC OA. Lastly, it should be noted that only sentences with software mentions were included in SoftwareKG. This decision was made as negative sentences by automatic extraction do not offer much value for future investigations, but strongly inflate the size of the KG. Manually annotated negative sentences from SoMeSci were included, as they can be utilized as negative examples for training ML models.

[8] Recently, Istrate et al. [128] published a larger dataset which is described in Section 2.1

Type	Frequency	Type	Frequency
General		**Properties**	
bibo:AcademicArticle	3,215,396	skg:relatesTo	9,410,482
nif:Sentence	9,011,987	skg:plugInOf	416,120
bibo:Journal	11,075		
skg:JournalRank	132,439	**Reification Statement**	3,031,272
skos:Concept	338		
Mentions		**Entities**	
skg:SoftwareMention	11,773,930	sd:Software	606,247
skg:VersionMention	3,437,368	sd:Version	422,114
skg:ExtensionMention	115,167	skg:Extension	22,932
skg:DeveloperMention	2,377,921	sd:Organization	113,230
skg:AltNameMention	320,196		
skg:CitationMention	2,180,563	biro:BibliographicReference	1,714,573
		biro:BibliographicRecord	415
skg:URLMention	616,445	schema:URL	173,568
skg:LicenseMention	10,104	dct:LicenseDocument	1148
Metadata Overall	9,057,764	Metadata Overall	3,054,227

Table 5.1: Overview of the main resources covered by SoftwareKG, integrating SoMeSci and automatically extracted data. Reification statements cover both confidences and proportions; only positive sentences are included except for SoMeSci; the low amount of *biro:BibliographicRecord* results only from SoMeSci as this aspect was not considered in automatic analyses (see Section 3.5).

5.5 Limitations

As described, the KG does only include information from scientific publications, making the only observable source for information on software the description of authors. This makes the information subject to uncertainty, which is directly represented by the model. However, especially for rarely mentioned software inaccuracies might still be present due to the sparse number of extracted mention contexts. Users of the KG should further be aware that the data model is not built to be exhaustive regarding software or other covered aspects. Instead it is built to specifically represent how software is mentioned in scientific articles and omits information about software that cannot be inferred from this context. Furthermore, it includes only metadata with respect to scientific publications required for immediate analysis of software in science, but does include identifiers to extend available data through external sources.

As outlined in Section 5.2.1, the modeling of specific information, e.g., developer, might not adequately reflect reality for all software. In this regard the data model definition is driven by the available data because the data is too sparse for more fine-grained modeling. However, it can be expected that data will always be sparse for mentions of highly specialized scientific software. Therefore, the defined model might be an overall suited representation. Another approach could be to model the information in more detail only for software with sufficient sample size, however, this would lead to inconsistent representations and might lead to problems in large-scale analyses.

6 | Large-Scale Analysis

The content of this section is based on the following publications:

> David Schindler et al. "The role of software in science: a knowledge graph-based analysis of software mentions in PubMed Central". In: *PeerJ Computer Science* 8 (2022), e835. DOI: 10.7717/peerj-cs.835 [245], covers a large-scale analyses of software mentions in 3.2 M articles of PMC OA.

> David Schindler et al. "Software use in retracted papers". In: *Proceedings of ISSI 2023 — the 19th International Conference of the International Society for Scientometrics and Informetrics.* 2023, pp. 411–414. DOI: 10.5281/zenodo.8428921 [249], covers a contribution describing the ongoing work on investigating the relationship between software mentions and article retractions.

> David Schindler et al. "Retracted articles use less free and open-source software and cite it worse". In: *Quantitative Science Studies* (2023), pp. 1–23. DOI: 10.1162/qss_a_00275 [248], covers an extended analyses of the relationship between software mentions and article retraction.

As described in Chapter 4 and Chapter 5, an automatic extraction pipeline for mentions of software in scientific publications was developed and applied for IE on >3 M publications, with the extracted information being modeled in the research KG SoftwareKG. Based on this information a large-scale analysis of software in science was performed, informed by prior analyses (see Section 2.1.3) and an initial analysis performed on the manually annotated gold standard dataset SoMeSci, described in Chapter 3. Specifically, the use of software in science, its citation quality, and the amount of utilized open source software were analyzed, with all aspects being further investigated regarding changes over time, differences between research domains, and the impact of the articles and the publishing journals. Moreover, the outlined IE pipeline was used in a specific case study investigating the relationship between software and article retraction. This study was based on different underlying data, and serves as further demonstration of how the methods established in this work can be applied to gain a better understanding on the impact of software on science. In the following, some base statistics on the publications contained in SoftwareKG[1] are introduced before the results of the analysis and the case study are described.

6.1 Publication Statistics

As described in Section 4.8.1, information from PubMedKG was included in SoftwareKG to integrate information on research domains. Overall, 303 sub-domains corresponding to 27 top-level domains are distinguished. While all information is represented in SoftwareKG, only top-level domains are utilized in the subsequent analysis to facilitate comparisons and to ensure a sufficiently

[1] Only automatically extracted information contained in SoftwareKG is considered in this chapter.

Main research domain	Research subcategories (excerpt)
Physics and Astronomy	Acoustics and Ultrasonics, Astronomy and Astrophysics, Atomic and Molecular Physics, and Optics
Chemistry	Analytical Chemistry, Chemistry (miscellaneous), Electrochemistry
Social Sciences	Anthropology, Archeology, Communication
Materials Science	Biomaterials, Ceramics and Composites, Electronic
Engineering	Aerospace Engineering, Architecture, Automotive Engineering
Economics, Econometrics and Finance	Economics and Econometrics, Economics, Econometrics and Finance (miscellaneous)
Multidisciplinary	Multidisciplinary
Energy	Energy (miscellaneous), Energy Engineering and Power Technology, Fuel Technology
Agricultural and Biological Sciences	Agricultural and Biological Sciences (miscellaneous), Agronomy and Crop Science, Animal Science and Zoology
Environmental Science	Ecological Modeling, Ecology, Environmental Chemistry
Veterinary	Equine, Food Animals, Small Animals
Nursing	Advanced and Specialized Nursing, Assessment and Diagnosis, Care Planning
Decision Sciences	Statistics, Probability and Uncertainty, Information Systems and Management
Earth and Planetary Sciences	Atmospheric Science, Computers in Earth Sciences, Earth and Planetary Sciences (miscellaneous)
Pharmacology, Toxicology and Pharmaceutics	Drug Discovery, Pharmaceutical Science, Pharmacology
Mathematics	Algebra and Number Theory, Analysis, Applied Mathematics
Computer Science	Artificial Intelligence, Computational Theory and Mathematics, Computer Graphics and Computer-Aided Design
Biochemistry, Genetics and Molecular Biology	Aging, Biochemistry, Biochemistry
Dentistry	Dentistry (miscellaneous), Oral Surgery, Orthodontics
Neuroscience	Behavioral Neuroscience, Biological Psychiatry, Cellular and Molecular Neuroscience
Arts and Humanities	Archeology (arts and humanities), Arts and Humanities (miscellaneous), Conservation
Psychology	Applied Psychology, Clinical Psychology, Developmental and Educational Psychology
Business, Management and Accounting	Accounting, Business and International Management, Business
Medicine	Anatomy, Anesthesiology and Pain Medicine, Biochemistry (medical)
Immunology and Microbiology	Applied Microbiology and Biotechnology, Immunology, Immunology and Microbiology (miscellaneous)
Health Professions	Chiropractics, Complementary and Manual Therapy, Health Information Management
Chemical Engineering	Bioengineering, Catalysis, Chemical Engineering (miscellaneous)

Table 6.1: Overview of the 27 main research domains and 3 of their sub categories that were used to group journals. Bold font highlights the abbreviation of the respective research domain used here.

large number of data samples for each domain. An excerpt of how top-level domains relate to sub-level domains is given in Table 6.1, with the full summary provided in supplementary Table A3. The table also indicates how domain names were abbreviated in all following analyses.

The distribution of the publication year of all articles contained in SoftwareKG is illustrated in Figure 6.1. In the illustration and for all subsequent analyses, articles published before 1990 were excluded due to low sample number and because they are also less relevant with respect of utilized scientific software. Moreover, publications from the year 2021 were excluded because only partial information on this year is available as data was gathered in January. This reduces the number of articles analyzed in subsequent analyses to 2,929,567 (96%).

Information on the underlying scientific domain of publications is available for 2,697,820 (92.1%) articles. The distribution of articles in the 27 top-level domains described in Table 6.1 is illustrated in Figure 6.2. As expected from a repository of open-access articles from Life Sciences, the distribution is skewed with ≈1.9 M articles relating to Medicine, while only ≈2 k articles are related to Economics. However, there is a high relative amount of articles not directly related to Medicine (819,643, >30%). This includes disciplines such as Computer Science (≈39 k articles) and Mathematics (≈76 k articles), but also Business (≈3 k articles) and Arts and Humanities (≈8 k articles).

The articles were published in a total of 11,605 different journals. The distribution of publications per journal is highly skewed with *PLoS ONE* containing 239,962 (8.2%) publications and *Scientific Reports* containing 125,673 (4.3%) publications, while other journals only contribute a single article, e.g., *Zygon*[2]. Information of the underlying journal rank is available for 1,936,845 (66.1%) of the considered publications. Moreover, 2,421,571 (82.7%) of the considered articles have a citation count larger than zero. Articles with a citation value of zero, on the other hand, were either not cited yet, or data is missing for them.

Different publication types are represented in the KG as described in Section 5.2.2. Their distribution over the largest groups of article types is illustrated in Figure 6.3, with the remaining types summarized under *Other*. In general, it has to be assumed that theses types strongly differ in their purpose, structure, and in the role of software within them, e.g., *Research Articles*, *Letters*, and *Case Studies* are assumed to differ strongly in these regards. While all types were added to the KG, peer reviewed articles are the main target of interest and only these are considered for the subsequent analyses. Specifically, the types *Research Article* and *Review Article* are considered, which rules out unwanted influences from strongly differing publication types with a different prior probability for software usage, e.g., *Letters*. Overall, research (76.8%) and review (8.3%) articles account for 2,493,869 (85%) articles covered in the KG. However, these two groups account for >95% of overall identified software mentions. For this group, domain information is available for 93.3% of articles, journal rank for 67.8%, and 85.9% have a citation count larger than zero.

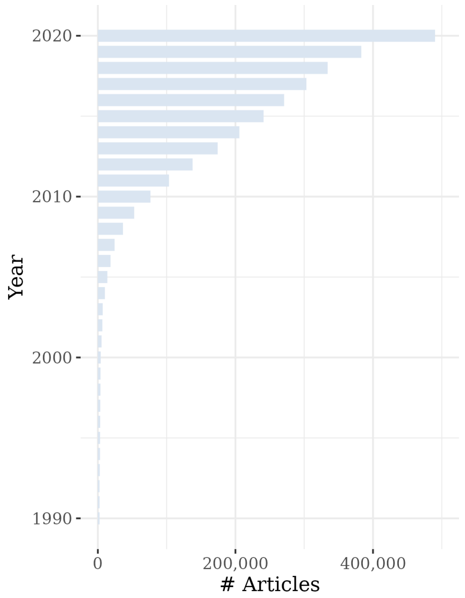


Figure 6.1: Distribution of publication year over all analyzed articles.

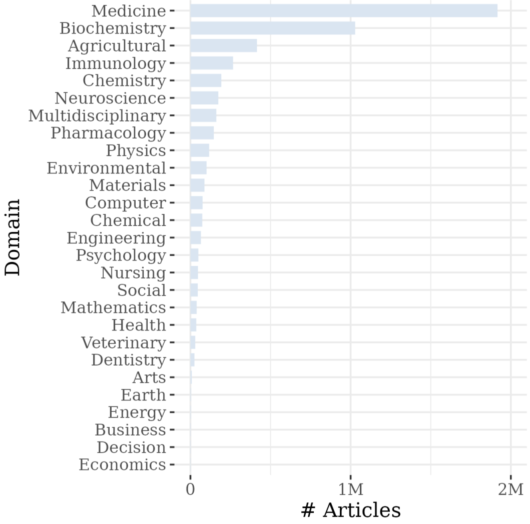


Figure 6.2: Relative amount of software mentions per research domain.

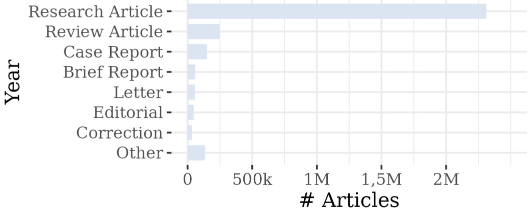


Figure 6.3: Relative amount of article types represented in SoftwareKG.

[2] https://onlinelibrary.wiley.com/journal/14679744, accessed 3 February 2024.

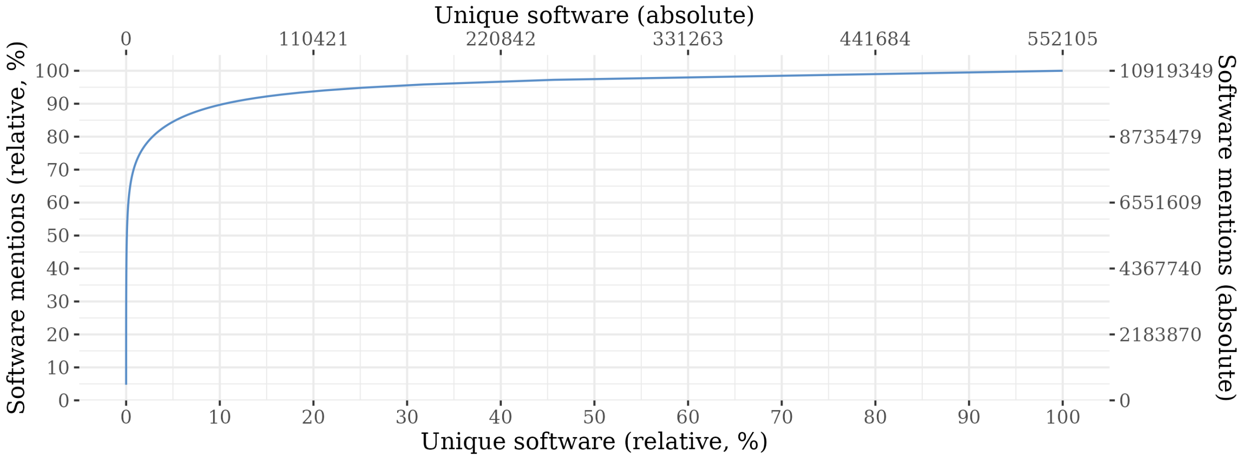


Figure 6.4: Cumulative distribution of software mentions per unique software. Left (bottom) scale gives the relative values, whereas right (top) scale provides the absolute numbers.

6.2 Results: Analysis of Software Mentions

In this section the main analyses regarding the role of software in science are summarized. First, aggregated results regarding software mentions in science are presented, investigating the software distribution, the most used software, and other base statistics. Then the number of software used and the usage of open source software is investigated along the dimensions of time, article discipline, journal impact, and article impact. Subsequently, the software citation quality is analyzed regarding the same variables, before additional aspects of software in science are explored. All analyses described below only consider the set of 2,493,869 research and review articles published between 1990 and 2020.

6.2.1 Software Mentions

Overall, 10,919,349 mentions of software were identified in the considered set, corresponding to 552,105 unique software entities with 1.25 different spellings and 19.8 mentions on average, after performing ED. A highly skewed distribution of mentions per software was observed, where about 10% of the software account for about 90% of the software mentions across all articles. Figure 6.4 illustrates this distribution graphically. Table 6.2 provides an overview of the ten most frequent software, including their absolute and relative number of mentions across all articles. With 517,262 respectively 446,877 mentions, *SPSS* and *R* are mentioned most frequently across all articles, where 749 different spellings were observed for *SPSS* and 69 for *R*[3]. The different spellings for *SPSS* include common names such as “SPSS” (80.3%), “SPSS Statistics” (10.7%), and “Statistical Package for the Social Sciences” (3.1%), but also spelling mistakes such as “Statistical Package for the Spcial [*sic*] Sciences”. *SPSS* is applied in twice as many articles as *R*, which is a notably higher difference as in absolute mention numbers. Such a difference can also be found between *Prism* and *ImageJ*, with *Prism* being applied in more articles, while *ImageJ* has a higher number of overall mentions.

The top ten most used software per research domain are further illustrated in Figure 6.5, from which domain-specific differences can be observed. Each domain is characterized by a different distribution of the top ten software, and no domain is consistent with the domain-independent view (see Table 6.2). *SPSS* (top software for 12/27) and *R* (7/27) together represent the top

[3] Spelling variations include *Alternative Names* identified by the IE pipeline.

Software	Abs. # Mentions	Rel. # Mentions	Abs. # Articles	Rel. # Articles	# Spellings
SPSS	517,262	.047	449,806	.180	749
R	446,877	.041	225,207	.090	69
Prism	214,225	.020	184,526	.074	23
ImageJ	222,779	.020	141,515	.057	397
Windows	134,267	.012	122,672	.049	14
Stata	139,889	.013	112,511	.045	267
Excel	141,638	.013	111,237	.045	216
SAS	134,045	.012	107,754	.043	390
BLAST	255,865	.023	97,423	.039	943
MATLAB	152,450	.014	85,743	.034	47

Table 6.2: Information about the ten most frequent software mentions across all disciplines together with their absolute and relative number of mentions, the number of articles that contain at least one mention and the number of spelling variation that were disambiguated. Note that the number of identified names includes all literature in SoftwareKG and not just *Research* and *Review Articles*.

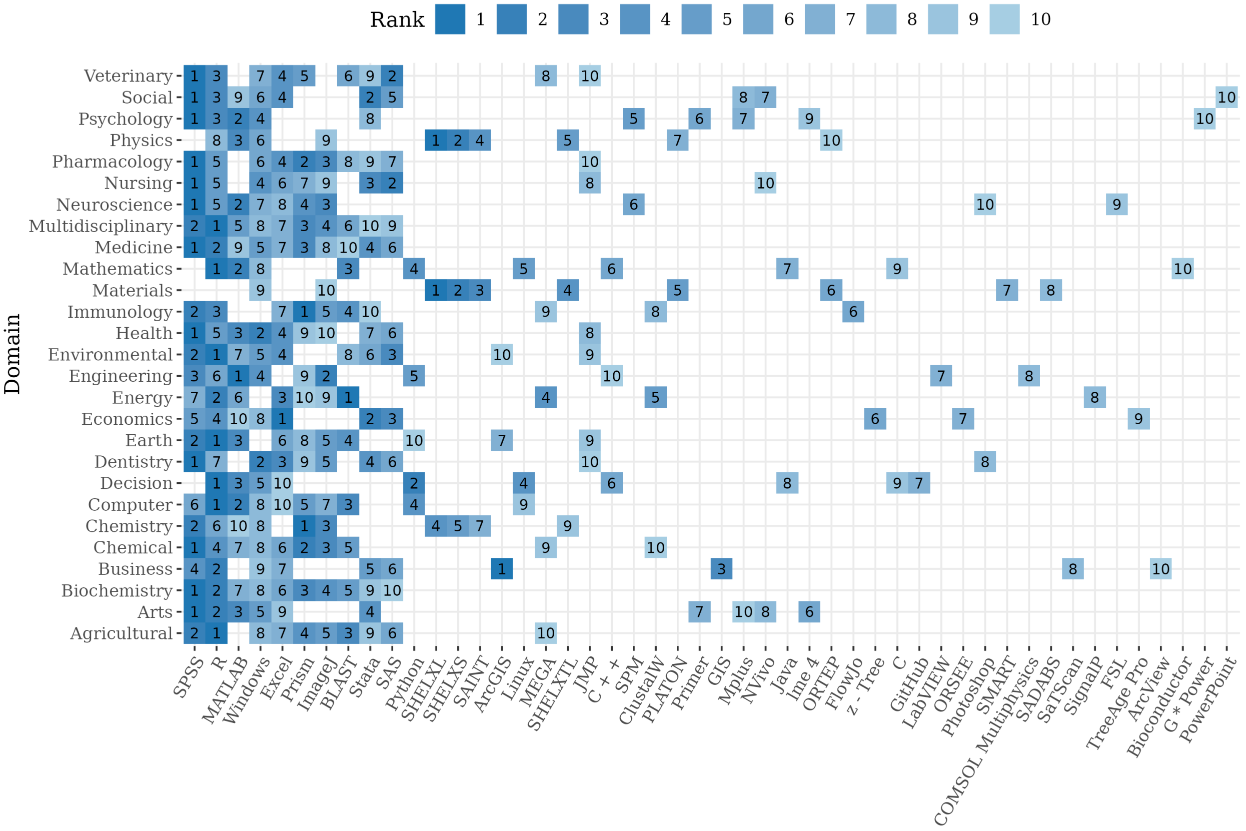


Figure 6.5: Top ten software per domain. Higher rank within the domain is represented by darker color. The number on the tile gives the rank within the domain. Software with rank higher than ten, are excluded from the plot to improve readability. Software are ordered by rank over all domains left to right.

mentioned software in more than 70% of the domains. *Prism* (2/27) is the top software in Chemistry and Immunology, while *BLAST*, *Excel*, and *ArcGIS* (1/27, each) are the top software in Energy, Economics, and Business, respectively. The software *SHELXL*, *SHELXS*, *SAINT*, and *SHELXTL* play a mayor role in Materials, and Physics, taking ranks among 1–5 in both domains. They are also present in Chemistry with a lower rank, but are not among the top ten software in any other field. Several *PEs* are listed among the top ten software, including *R*, *Python*, *Java*, *C*, and *C++*, most prominent in Mathematics, Engineering, and Decision Sciences. Material Science shows the least overlap with the general top ten software sharing only the *OS Windows* and the application *ImageJ*. Regarding *OSs*, *Windows* is frequently mentioned in all research domains, except for Earth and Planetary Sciences, Energy, and Immunology. The *Linux OS*, in contrast, is ranked 5th in Mathematics, 4th in Decision Sciences, and 9th in Computer Science, respectively.

The usage of specialized software between domains is further illustrated based on statistical software, which is the most common type of software applied in scientific investigations. A summary of the most mentioned statistical software in the time from 2008 to 2020 is given in Figure 6.6. Since the number of mentions increases across all software in absolute numbers, due to the large increase in covered publications illustrated in Figure 6.1, the relative number of mentions by the number of analyzed articles is considered. With respect to all articles, it can be observed that there are different trends between software, with the number of *SPSS* steadily increasing from 2008 to 2016, but reaching a plateau after 2016. The mention of *R* and *Prism*, on the other hand, has steadily increased throughout the observed time frame. The plot for Multidisciplinary articles, which involves more than one specific research domain, varies from the overall trend, particularly with respect to *SPSS* and *R*. While *SPSS* is used less frequently in comparison, as already observed in Figure 6.5, the use of R has strongly increased since 2013 with R being the most used software since 2017.

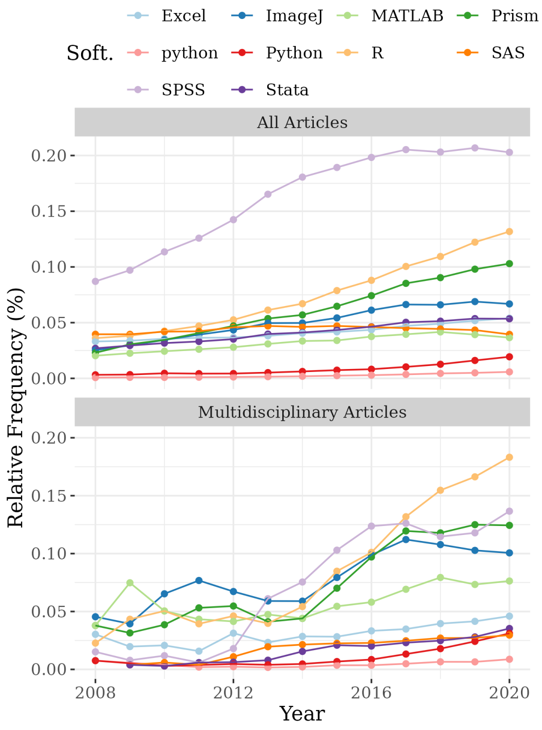


Figure 6.6: Relative number of articles mentioning the most frequently used statistical software, illustrated across all articles and for the domain of Multidisciplinary research.

It was further found that 48.9% (CI_{95}=[48.8, 48.9]) of distinct software used within articles is available free for research purposes and 39.3% (CI_{95}=[39.2, 39.3]) is published under open source licenses, estimated based on the 428 distinct software annotated for information enrichment as described in Section 4.8.1. There is also a strong relation between free and open source software, with all of open source software being also free, and 71.2% of free software being open source.

6.2.2 Article Level Statistics

On the article level, it was found that 72% (CI_{95}=[72,0, 72,1]) of articles mention software, with 3.93 (CI_{95}=[3.93, 3.93]) distinct software being mentioned on average within them. The number ranges from 1 software mention in 564,387 articles to a maximum of 225 distinct software mentioned within a single article. In this particular case, the article performed a review of existing software tools for mass spectral plant metabolomics [213].

The relative number of articles mentioning software over time is found to increase as illustrated in Figure 6.7 (top, blue line), with only a small fraction of articles mentioning software before 1997.

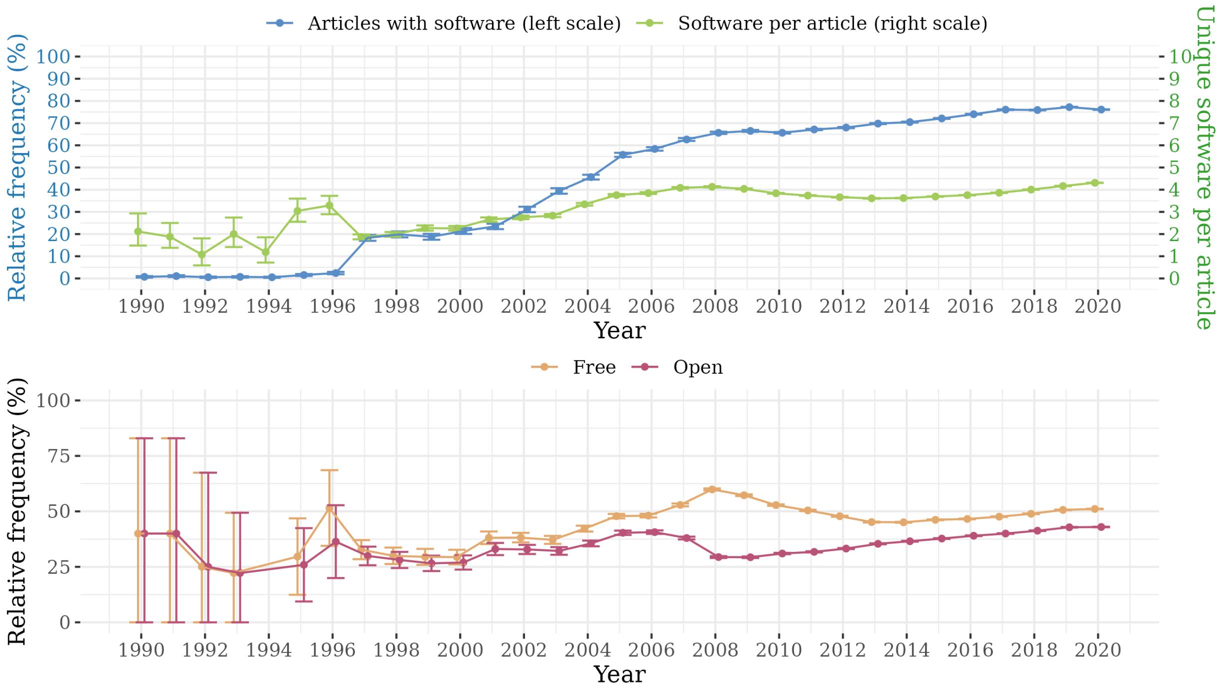


Figure 6.7: Top: *Blue*: Relative frequency of articles with at least one software mention per year; *Green*: Absolute mean frequency of distinct software mentioned per article with at least one software mention per year. **Bottom**: *Orange*: Relative amount of mentioned software that is available free for research purposes per year; *Red*: Relative amount of mentioned software with open source code per year. 95% CIs are provided in all plots.

In the year 1997, a steep increase in the number of articles with software can be observed from 2.4% (CI_{95}=[1.88, 2.96]) to 18.3% (CI_{95}=[17.0, 19.6]). From 2001 to 2008 another strong increase can be observed from 23.4% (CI_{95}=[22.2, 24.7]) to 65.6% (CI_{95}=[65.1, 66.2]), which is followed by a less steep but consistent increase over the remaining time frame up to 76.1% (CI_{95}=[76.0, 76.2]) in 2020. From 2005 more than 50% of the articles contain at least one software. The average number of distinct software within these articles is further illustrated in Figure 6.7 (top, green line). Due to the low sample number before 1997, the estimation of mean software frequencies per article is less reliable in this time, which is reflected by the provided CIs. In the time from 1997 to 2007 the number of software used per article doubles from a value of 1.87 (CI_{95}=[1.76, 1.98]) up to 4.08 (CI_{95}=[4.05, 4.12]), with the frequency remaining at a similar level of $\approx$4 till 2020. Figure 6.7 (bottom) further illustrates the amount of mentioned software available for free for research purposes (orange) and published open source (red). As above, it can be observed from the CIs that there is a high uncertainty in results before 1997. Afterwards, an increase is observed for the use of free software (orange) from 29.4% (CI_{95}=[26.1, 32.7]) in 2000 to 59.9% (CI_{95}=[59.4, 60.3]) in 2008, followed by a decline down to 45.1% (CI_{95}=[44.9, 45.3]) in 2014 and a slight, continuous increase since then back up to 51.1% (CI_{95}=[51.0, 51.2]) in 2020. Regarding open source software (red), an increase can be observed from 26.6% (CI_{95}=[23.1, 30.1]) to 40.6% (CI_{95}=[39.8, 41.4]) between 1999 to 2006, also followed by a decline to 29.3% (CI_{95}=[29.0, 29.8]) till 2008. Since then, a steady increase can be observed for the remaining time frame up to 43% (CI_{95}=[42.8, 43.1]) in 2020. In the time from 2008 to 2011 a notable difference can be observed between the number of software available for free and published open source. This effect can be attributed to a high number of software mentions for the software toolchain around *SHELX*, identified in Figure 6.5, which was

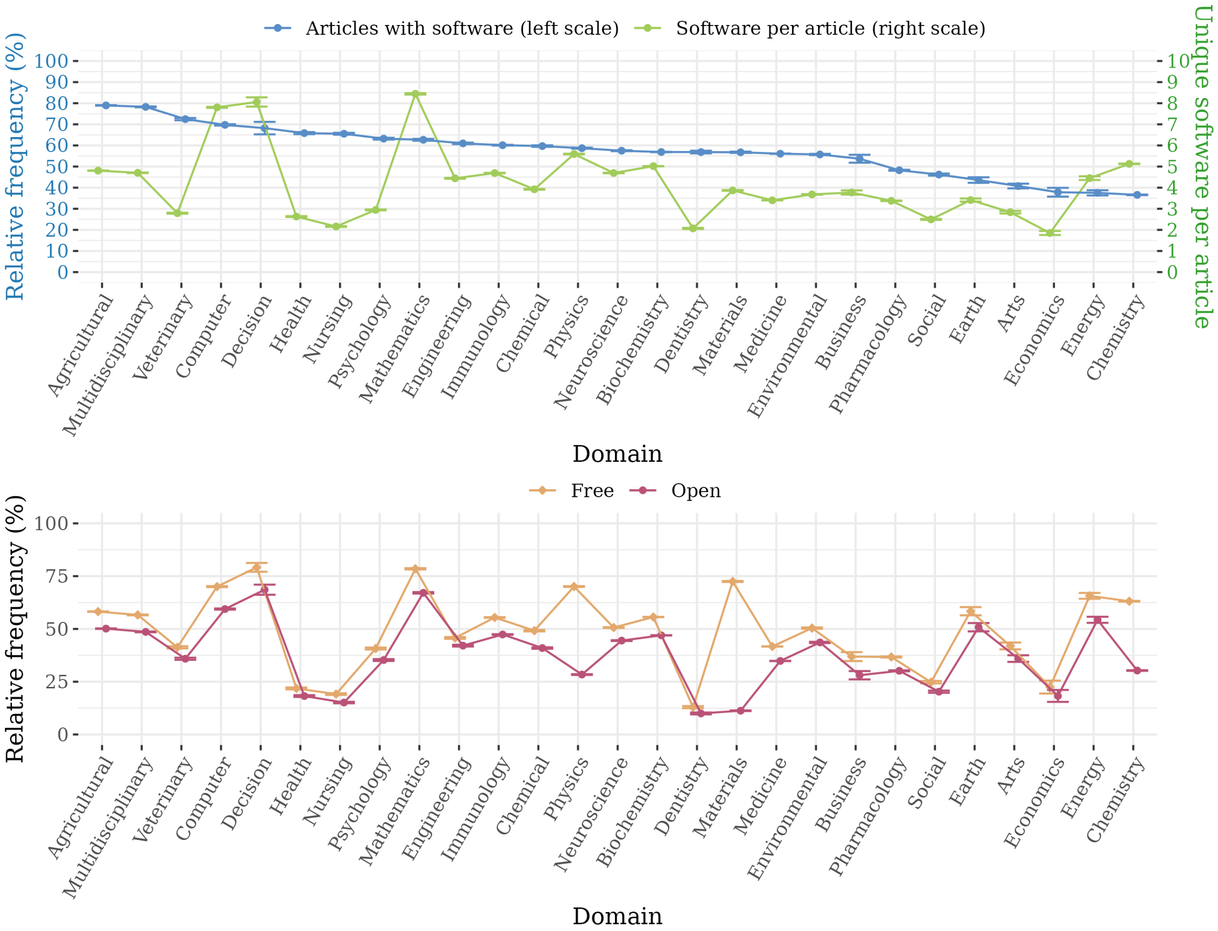


Figure 6.8: Top: *Blue*: Relative frequency of articles with at least one software mention per research domain; *Green*: Absolute mean frequency of distinct software mentioned per article with at least one software mention per research domain. **Bottom**: *Orange*: Relative amount of mentioned software that is available free for research purposes per research domain; *Red*: Relative amount of mentioned software with open source code per research domain. 95% CIs are provided in all plots.

most popular in that time frame, because all of these tools are free but their source code is not publicly available. Similar effects were also found in the following analyses regarding disciplines and journal impact.

When looking at the relative amount of articles with software per research domain, notable differences between the individual domains can be observed. Figure 6.8 (top, blue line) illustrates these differences graphically. While in Chemistry and Energy only respective 36.6% (CI_{95}=[36.4, 36.7]) and 37.5% (CI_{95}=[36.3, 38.8]) of articles contain software mentions, 78.3% (CI_{95}=[78.1, 78.5]) and 79% (CI_{95}=[78.8, 79.1]) of articles mention software in the domains of Agricultural and Biological Sciences and Multidisciplinary Science, respectively. The number of different software per article shows a strong discrepancy for some domains (Figure 6.8, top, green line). The domains of Computer Science, Decision Sciences, and Mathematics have a notably higher number of distinct software mentions per article compared to all other domains with respective values of 7.8 (CI_{95}=[7.78, 7.83]), 8.06 (CI_{95}=[7.84, 8.28]), and 8.44 (CI_{95}=[8.41, 8.48]). The lowest value on the other hand is observed in Economics with 1.85 (CI_{95}=[1.76, 1.95]) software per article. Furthermore, Figure 6.8 (bottom) illustrates differences in the use of free and open source software between domains. No-

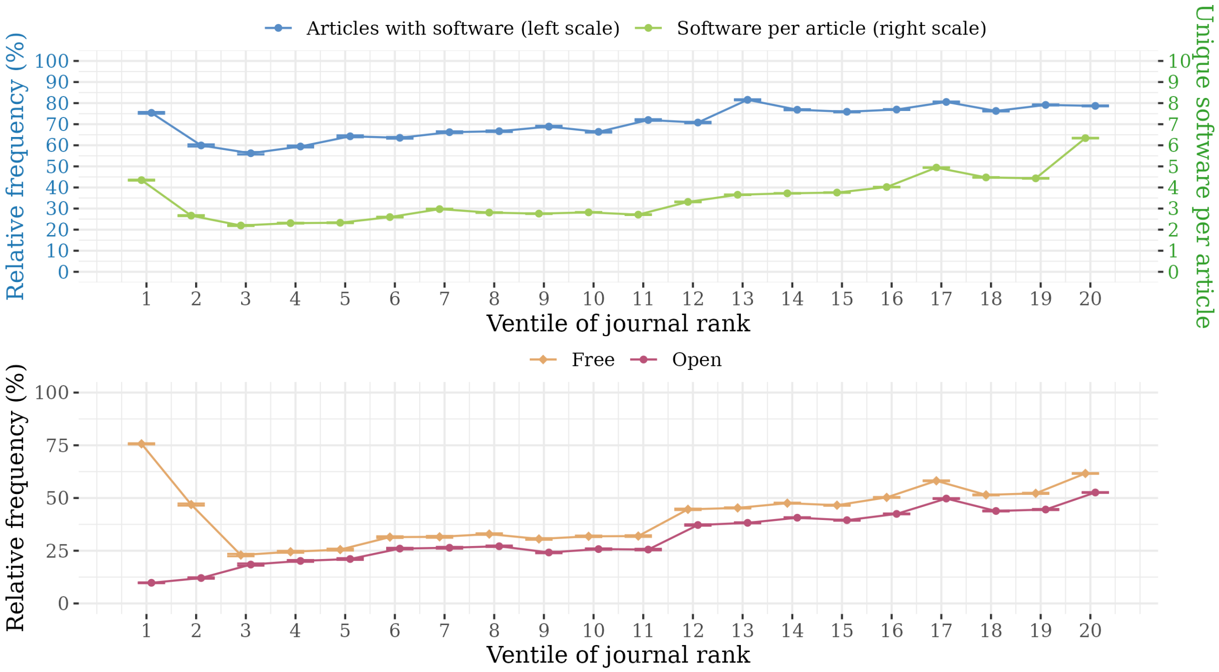


Figure 6.9: Top: *Blue*: Relative frequency of articles with at least one software mention per ventile of journal rank; *Green*: Absolute mean frequency of distinct software mentioned per article with at least one software mention per ventile of journal rank. **Bottom**: *Orange*: Relative amount of mentioned software that is available free for research purposes per ventile of journal rank; *Red*: Relative amount of mentioned software with open source code per ventile of journal rank. 95% CIs are provided in all plots.

tably, the domains with the highest number of software per article, also show a high use in both free and open source software, e.g., Mathematics with ratios of 78.5% (CI_{95}=[78.2, 78.8]) and 67.1% (CI_{95}=[66.8, 67.5]), respectively. Another notable effect regards the domains of Physics, Materials Science, and Chemistry, where a particularly high difference is present between the number of free and open source software, with values of 70.1% (CI_{95}=[69.9, 70.3]) to 28.4% (CI_{95}=[28.2, 28.6]) in Physics, 72.5% (CI_{95}=[72.3, 72.7]) to 11.3% (CI_{95}=[11.1, 11.4]) in Material Science, and 63.1% (CI_{95}=[62.9, 63.3]) to 30.3% (CI_{95}=[30.1, 30.5]) in Chemistry. This discrepancy is due to the software toolchain around *SHELX*, which is mainly applied in these domains as illustrated in Figure 6.5.

Further the relation between the amount of software mentions and the bibliographic measures of journal rank and citation count per year were investigated. Figure 6.9 (top) illustrates the ventiles (20-quantiles) of the journal rank, grouped by domain to prevent domain specific biases due to higher journal ranks. In detail, the journals were distributed according to their rank ventiles for each domain and the resulting ventiles were then merged across all domains. Further, the ventiles are calculated on a yearly basis because journal ranks are also calculated on a yearly basis. Ventiles are chosen because they allow an assessment of fine-grained trends, while also achieving an aggregation over multiple journals, which is argued to lead to more reliable results. Following this approach *PLoS ONE* is ranked in the 13th ventile from 2017 to 2020, while *Nature* is ranked in the highest ventile (20th) since 2012, with an example of a journal in the lowest ventile discussed below. A similar approach was chosen for summarizing the articles via citation count ventiles on a yearly basis, illustrated in Figure 6.10.

When considering the journal rank, it was found that 75.4% (CI_{95}=[75.0, 75.8]) of the articles on the lowest rank contain software mentions, followed by a strong decrease for the next two

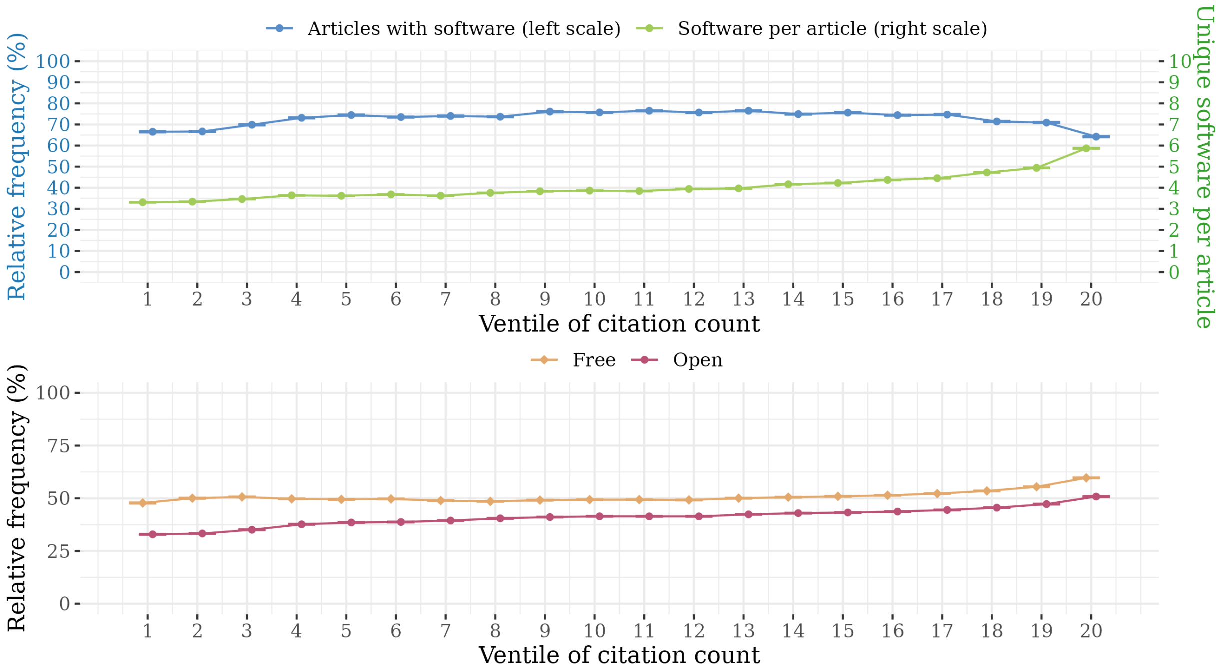


Figure 6.10: Top: *Blue*: Relative frequency of articles with at least one software mention per ventile of citation count; *Green*: Absolute mean frequency of distinct software mentioned per article with at least one software mention per ventile of citation count. **Bottom**: *Orange*: Relative amount of mentioned software that is available free for research purposes per ventile of citation count; *Red*: Relative amount of mentioned software with open source code per ventile of citation count. 95% CIs are provided in all plots.

ventiles to a value of 56.2% (CI_{95}=$[55.7, 56.8]$) (Figure 6.9, top, blue line). For the remaining ventiles an increasing trend up to ≈80% for higher journal ranks could be observed with the highest value at ventile 13 with a value of 81.6% (CI_{95}=$[81.3, 81.8]$). The number of software per articles is overall similar, consisting of an initial high point of 4.35 (CI_{95}=$[4.33, 4.37]$), a short decline to 2.2 (CI_{95}=$[2.17, 2.22]$) at ventile three, followed by an increase as the rank rises with a final high point at the highest ventile with a value of 6.34 (CI_{95}=$[6.33, 6.35]$) (Figure 6.9, top, green line). Regarding the relative amount of free and open source software the results are illustrated in Figure 6.9 (bottom). For both free and open source software, a steady increase in mention frequency can be observed as the journal rank increases. For free software the value increases from 12.1% (CI_{95}=$[11.7, 12.4]$) to 61.6% (CI_{95}=$[61.5, 61.8]$) from ventile three to 20 and for open source from 9.8% (CI_{95}=$[9.6, 10.0]$) to 52.6% (CI_{95}=$[52.5, 52.8]$) from ventile one to 20. Notably, in the lowest two ventiles there is a strong discrepancy between free and open source software, e.g., at ventile one 75.7% (CI_{95}=$[75.4, 75.9]$) of software are found to be free in comparison to the 9.8% of aforementioned open source software. Same as above, this difference is caused by the software toolchain around *SHELX*. The lowest ventile, for instance, comprises the journal *Acta Crystallographica Section E* with almost 20 *k* (62.3% of the 1st ventile) publications set in the domains of Chemistry, Materials Science, and Physics and Astronomy, in which a high number of publications utilize the *SHELX* toolchain.

With respect to the citation count (Figure 6.10) only small differences in the amount of articles using software can be observed between ventiles (top, blue line). A small drop is present at both edges at the lowest and highest ventiles with values of 64.2% (CI_{95}=$[63.9, 64.5]$) and 66.5% (CI_{95}=$[66.2, 66.8]$) for the 1st ventile and the 20th ventile, respectively, while the maximum is reached at the 11th ventile with a value of 76.5% (CI_{95}=[76.2, 76.8]). The number of distinct software per article, however, increases with citation count from a minimum of 3.31 (CI_{95}=[3.29, 3.32]) at the 1st ventile up to the maximum of 5.87 (CI_{95}=[5.85, 5.89]) at the highest ventile. Regarding free and open source software (bottom), there is a slight change in the ratio of free software (orange) from 47.8% (CI_{95}=[47.5, 48.0]) at the lowest ventile to 59.7% (CI_{95}=[59.5, 59.9]) at the highest, with the strongest increase in the highest three ventiles. For open source software (red) a similar but trend can be observed, particularly at the edges with a higher ratio of open source software at higher journal ranks from 32.9% (CI_{95}=[32.7, 33.1]) at the 1st ventile to 50.8% (CI_{95}=[50.6, 51.0]) at the highest ventile.

Since the results presented in Figure 6.8 indicated a relation between the number of software mentioned in an article and the ratio of free and open source software used, this relationship is further illustrated in Figure 6.11. The results show that there is strong relation between the number of software mentioned within an article and the ratio of both free and open source software used. While on average 19.9% (CI_{95}=[19.8, 20.0]) of software mentioned in articles with one software are free and 17.4% (CI_{95}=[17.3, 17.5]) are open source, the values increase to 67.9% (CI_{95}=[67.7, 68.0]) and 49.8% (CI_{95}=[49.6, 50.0]) for five used software, and further up to 87.1% (CI_{95}=[86.5, 87.6]) and 76% (CI_{95}=[75.3, 76.6]) for 15 mentioned software, while the values start approaching a plateau beyond this point.

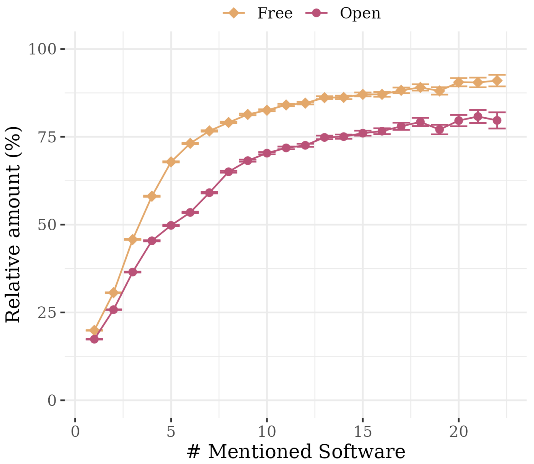


Figure 6.11: *Orange*: Relative amount of mentioned software that is available free for research purposes by number of software mentioned per article; *Red*: Relative amount of mentioned software with open source code by number of software mentioned per article. 95% CIs are provided.

Finally, four regression models were used to further assess and quantify the relation between software mentions and the article metadata examined above. The first model is a logistic regression fitted to predict the binary target of whether an article is mentioning software based on the covariates of publication year, scientific domain, journal rank, and citation count. As before, journal rank and citation count are modeled in ventiles, while individual domains are provided as binary features, and the year as a discrete number. The full results for the fitted regression are available in supplementary Table A4, with effect sizes provided as odds-ratios (ORs). The results show that the year has significant positive influence with OR=1.091 (CI_{95}=[1.090, 1.092]) for each year[4]. Further, almost all domains were found to have a significant influence, e.g., Computer Science articles were found to be more likely to mention software with OR=1.729 (CI_{95}=[1.672, 1.788]). The journal rank also has a positive influence with OR=1.048 (CI_{95}=[1.047, 1.048]) per additional ventile. The citation rank, on the other hand, has a small negative impact with OR=.985 (CI_{95}=[.985, .986]) per ventile.

The second model was used to predict the number of software mentioned per article containing software. Due to the count based target variable, a Poisson regression is used for this model. Based on the methodology described by Gelman and Hill [90] it was initially tested whether the variance equals the mean, to assess if the underlying assumption for the Poisson regression holds true [292]. Since overdispersion was detected, i.e, a higher variance as assumed by the model, a quasi-likelihood

[4] The provided OR values are significant, when the provided confidence interval does not include 1.

estimation was performed, which includes an estimation of a dispersion parameter. The full results are available in supplementary Table A5. The results also show a significant positive influence of the publication year, with OR=1.027 (CI_{95}=[1.026, 1.028]) for the number of software with each year. Moreover, all disciplines have significant influence, e.g., Computer Science with OR=2.394 (CI_{95}=[2.362, 2.428]). The journal rank also has a significant positive influence with OR=1.034 (CI_{95}=[1.034, 1.035]) per ventile, while the citation count also has a positive but small influence with OR=1.009 (CI_{95}=[1.009, 1.010]) per ventile.

Furthermore, two logistic regression models were used to predict the binary targets of whether software is freely available and open source. The same covariates as before were considered, with the addition of the number of software contained in an article as a discrete numerical input. The full results are available in supplementary Table A6 for free software and in Table A7 for open source software. In both cases the number of software mentioned in an article has an influence with strong effect sizes of OR=1.347 (CI_{95}=[1.345, 1.349]) (free) and OR=1.223 (CI_{95}=[1.222, 1.225]) (open source) with every additional software mentioned. The influence of the year is small for free software with OR=1.008 (CI_{95}=[1.007, 1.008]), and slightly higher for open source software with OR=1.023 (CI_{95}=[1.022, 1.023]) per year. Further, almost all domains have significant influence on both free and open source software. The journal rank has a positive effect in both cases with OR=1.03 (CI_{95}=[1.030, 1.031]) (free) and OR=1.047 (CI_{95}=[1.046, 1.048]) (open source) per ventile, while the citation count has a small negative influence with OR=.992 (CI_{95}=[.991, .992]) (free) and OR=.995 (CI_{95}=[.995, .996]) (open source) per ventile.

6.2.3 Software Citation Completeness

Considering articles containing software mentions, it was analyzed whether the metadata necessary to identify particular software was provided within an article. For each distinct software per an article, which might be mentioned multiple times, it was analyzed whether the used version, its developer, a corresponding URL, and a formal citation were provided. In total, a version was provided for 39.5% (CI_{95}=[39.5, 39.6]) of software and the developer for 28.5% (CI_{95}=[28.5, 28.5]). Further, software was formally cited in 24.8% (CI_{95}=[24.8, 24.9]) of cases and a URL provided for 7.4% (CI_{95}=[7.4, 7.4]).

Same as in Section 3.3, the results are further summarized regarding identifiability of the software itself, developer attribution, and the identification of the used codebase. In the following, software is considered as precisely identifiable when the developer is provided additionally to the name, or when it is formally cited, while proper attribution is considered as given in the same cases. These definitions are made based on the findings of Section 3.3, which showed that developer information is commonly present in formal citations, while attribution is also provided through the citation. Further, software is additionally considered as uncertain identifiable when only an URL is provided as argued in Section 3.3, because URLs are often not persistent and might not identify the correct target. Lastly, the codebase is considered as identifiable when the version (or release) was provided and the software itself is identifiable. Here, formal citations are not considered as sufficient because the results of Section 3.3 showed that a majority of formal citations points to references that do not include version numbers (69.6% software articles). Moreover, version numbers are also not consistently present in the remaining citation formats (64.9% in direct software citations). Overall, precise identification and attribution is possible in 51.3% (CI_{95}=[51.3, 51.3]) of software mentions, while an uncertain identification is given in 56.8% (CI_{95}=[56.8, 56.8]), and the codebase can be identified in 25.4% (CI_{95}=[25.4, 25.5]) of mentions. As discussed in Section 3.3, these are theoretic criteria and more software might be identifiable based on the name and context alone, however, the information is in general required for identification due to known ambiguities and legacy software

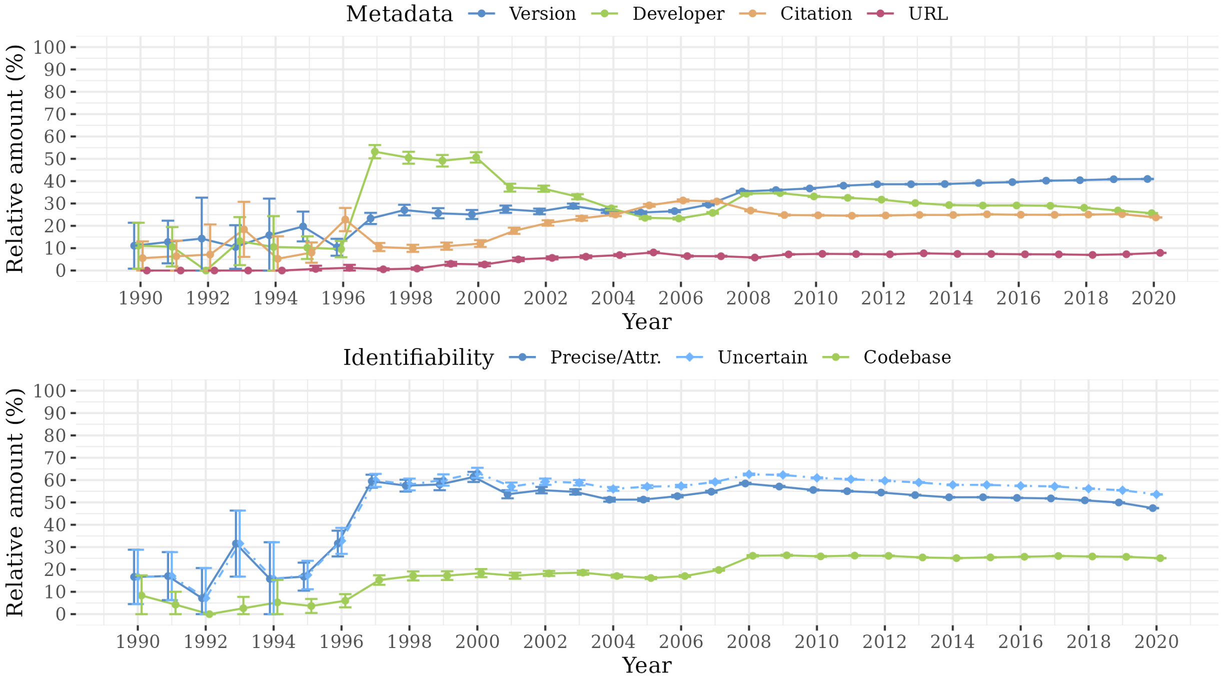


Figure 6.12: Top: relative amount of provided metadata (version, developer, formal citation, and URL) for software mentioned in articles per year. **Bottom**: relative amount of identifiable software in terms of software itself (and developer attribution) and codebase identification per year. Software identification is split between precise identification and uncertain identification, which does consider identification by only a provided URL. 95% CIs are provided in both plots.

for which limited information is available.

Figure 6.12 depicts the completeness of software mentions per year. As in previous results, a high uncertainty in metadata completeness (Figure 6.12, top) can be observed before 1997, and all metadata is found to be at a low level in this period, with URLs being almost never provided. The amount of provided versions stays consistent at a value of around 25% in the interval between 1997 and 2006, after which a positive trend from 26.6% (CI_{95}=[25.8, 27.3]) in 2006 up to 41% (CI_{95}=[40.9, 41.1]) in 2020 is observed. From 1997 to 2000 there is a peak in mention of developers with values of more than 50%, e.g., 53.2% (CI_{95}=[50.2, 56.2]) in 1997, followed by a decline down to 23.3% (CI_{95}=[22.8, 23.7]) in 2006. From 2006 there is a brief increase of provided developers up to 34.4% (CI_{95}=[34.1, 34.7]) in 2008, followed by a steady decline for the remaining time down to 25.7% (CI_{95}=[25.6, 25.8]) in 2020. For formal citations a rise can be observed from 10.5% (CI_{95}=[8.7, 12.3]) in 1997 up to 31.4% (CI_{95}=[30.9, 31.8]) in 2006, which is followed by a brief decline down to 24.8% (CI_{95}=[24.6, 25.1]) in 2009. For the remaining time the amount of formal citations stays at a plateau at about 25%. The amount of URLs is generally at a low level, with an initial increase between 1997 and 2005 from .5% (CI_{95}=[.1, 1.0]) to the overall maximum value of 8.1% (CI_{95}=[7.8, 8.4]), after which the values stay at a plateau below 8%. Regarding identifiability (Figure 6.12, bottom), the values are also at a low level before 1997. From 1996 to 1997 the values increase from 31.6% (CI_{95}=[25.8, 37.4])[5] to 59.5% (CI_{95}=[56.6, 62.4]) due to the peak in developer mentions, and stay at a high level of 58.5% (CI_{95}=[58.2, 58.8]) till 2008 due to the increased number of formal citations. From 2009, there is a decline for the remaining time frame

[5] Precise values are provided, while uncertain values show similar results.

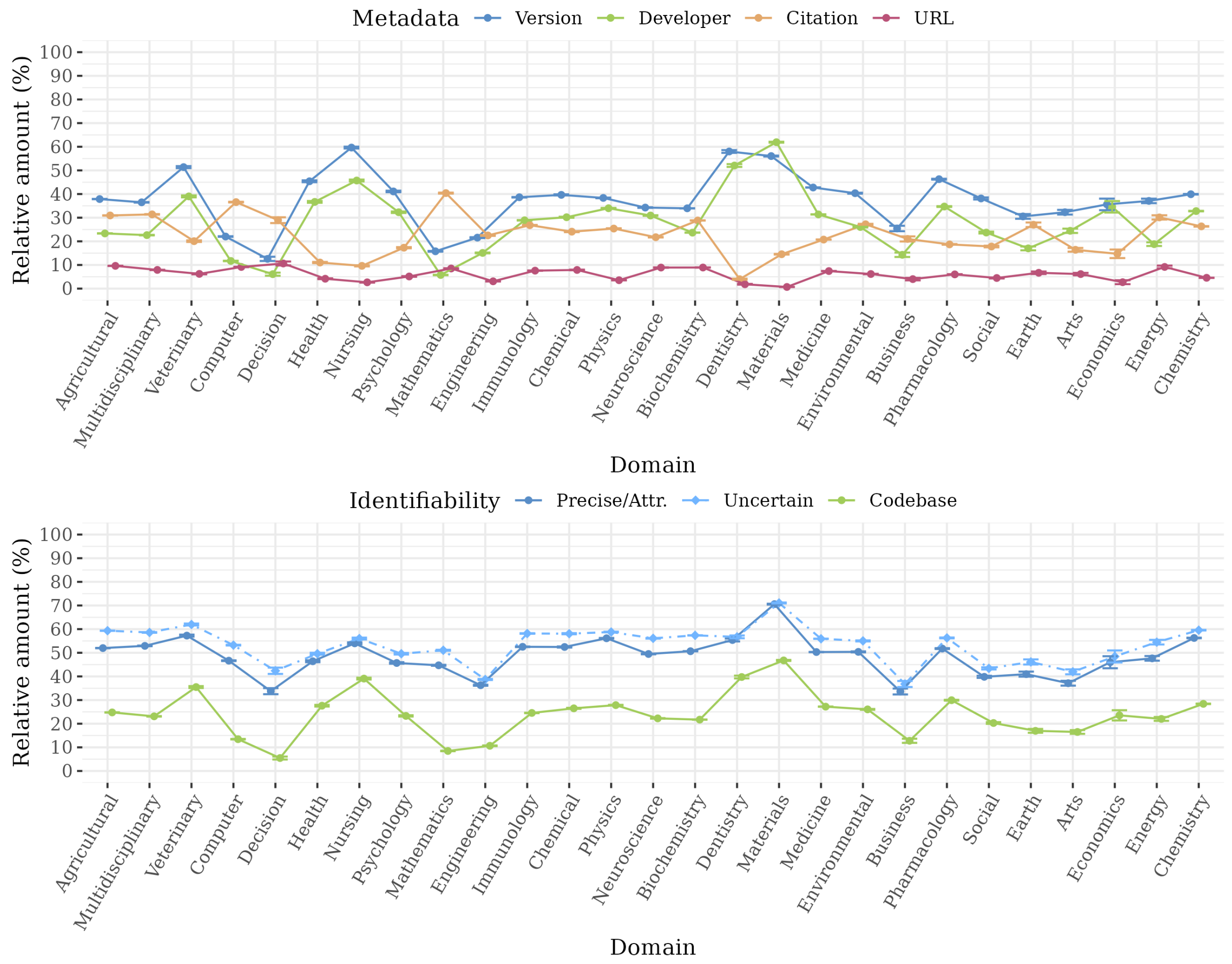


Figure 6.13: Top: relative amount of provided metadata (version, developer, formal citation, and URL) for software mentioned in articles per domain. **Bottom**: relative amount of identifiable software in terms of software itself (and developer attribution) and codebase identification per domain. Software identification is split between precise identification and uncertain identification, which does consider identification by only a provided URL. 95% CIs are provided in both plots.

down to 47.5% (CI_{95}=[$47.4, 47.5$]) in 2020. For codebase identification there is an initial increase from 6% (CI_{95}=[$3.1, 8.9$]) to 15.3% (CI_{95}=[$13.1, 17.4$]) in 1997, followed by a plateau around 17% up to 2006. Then a brief increase occurs to 26.1% (CI_{95}=[$25.8, 26.4$]) in 2008, followed by another plateau around 26% for the remaining time frame. Even so version completeness rises after 2008, the decreasing software identification leads to the overall plateau in codebase identification.

Figure 6.13 illustrates the amount of information provided per software over different research domains. Overall, there are strong differences in provided metadata between domains (Figure 6.13, top), and two inverse trends can be observed. Some domains such as Health Science, Nursing, Dentistry, and Materials Science provide comparatively high numbers of developers and versions, but low numbers of formal citations. In Nursing, for instance, versions are provided with an amount of 59.7% (CI_{95}=[$59.3, 60.0$]) and developers with 45.7% (CI_{95}=[$45.3, 46.1$]) but formal citations with only 9.6% (CI_{95}=[$9.4, 9.9$]). On the other hand, there are domains with a comparatively high number of formal citations which provide notably fewer developer and version metadata such as Computer Science, Decision Science, and Mathematics, e.g., in Mathematics 15.8% (CI_{95}=[$15.6, 15.9$]) of software

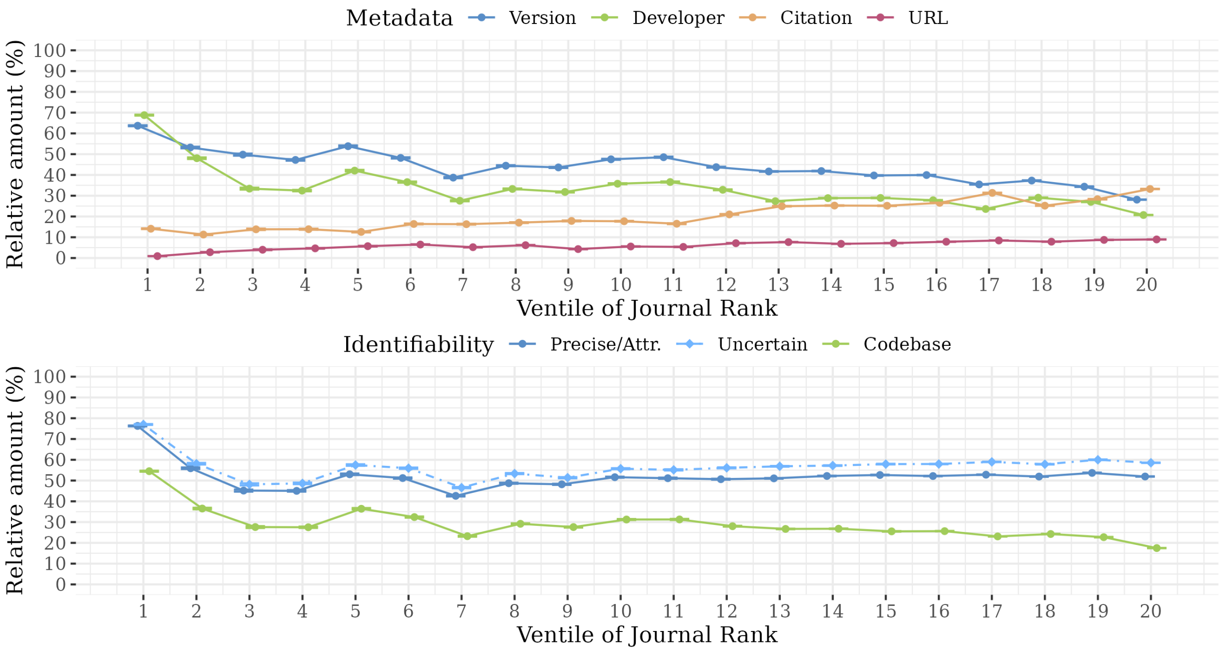


Figure 6.14: Top: relative amount of provided metadata (version, developer, formal citation, and URL) for software mentioned in articles per journal rank ventile. **Bottom**: relative amount of identifiable software in terms of software itself (and developer attribution) and codebase identification per journal rank ventile. Software identification is split between precise identification and uncertain identification, which does consider identification by only a provided URL. 95% CIs are provided in both plots.

are mentioned with a version, 5.8% (CI_{95}=[5.7, 5.9]) with developer, and 40.4% (CI_{95}=[40.2, 40.6]) are formally cited. Interestingly, this overlaps with domains mentioning a high number of software as observed from Figure 6.7. The same trends also influence the identifiability. Where citation and developer mention practice partially balance each other out with respect to overall identifiability and attribution, the differences are high regarding codebase identification, e.g., in Decision Sciences only 5.5% (CI_{95}=[4.9, 6.1]) of codebases are identifiable, while 46.8% (CI_{95}=[46.6, 47.0]) are identifiable in Materials Sciences. Regarding URLs there is some variation between domains, but the values are generally at a low level.

With respect to journal rank, a slight negative correlation between the rank ventile and the amount of provided versions and developer metadata can be observed, with maximum values of 63.7% (CI_{95}=[63.4, 63.9]) for version and 68.8% (CI_{95}=[68.5, 69.0]) for developer at the 1st ventile and minimum values of 28.1% (CI_{95}=[28.0, 28.2]) and 20.7% (CI_{95}=[20.6, 20.8]) at the highest (20th) ventile. Figure 6.14 (top) illustrates these relations graphically. In turn, a small positive correlation is found between the ventile and the provided formal citations, with the minimum value of 11.3% (CI_{95}=[11.0, 11.6]) at the 2nd ventile and the maximum of 33.2% (CI_{95}=[33.1, 33.3]) at the 20th ventile. In low ventiles there is also a small increase in provided URL which stays at a plateau below 10% beyond the 5th ventile. With respect to identifiability (Figure 6.14, bottom) an initial decrease is observed in low journal ranks with values plateauing from the 5th to 20th ventile around a value of 50% and 55% for precise and uncertain identification. Codebase identification, behaves similarly in low ventiles, but notably decreases beyond the 12th ventile from 28% (CI_{95}=[27.8, 28.2]) to 17.5% (CI_{95}=[17.4, 17.6]) at the highest ventile.

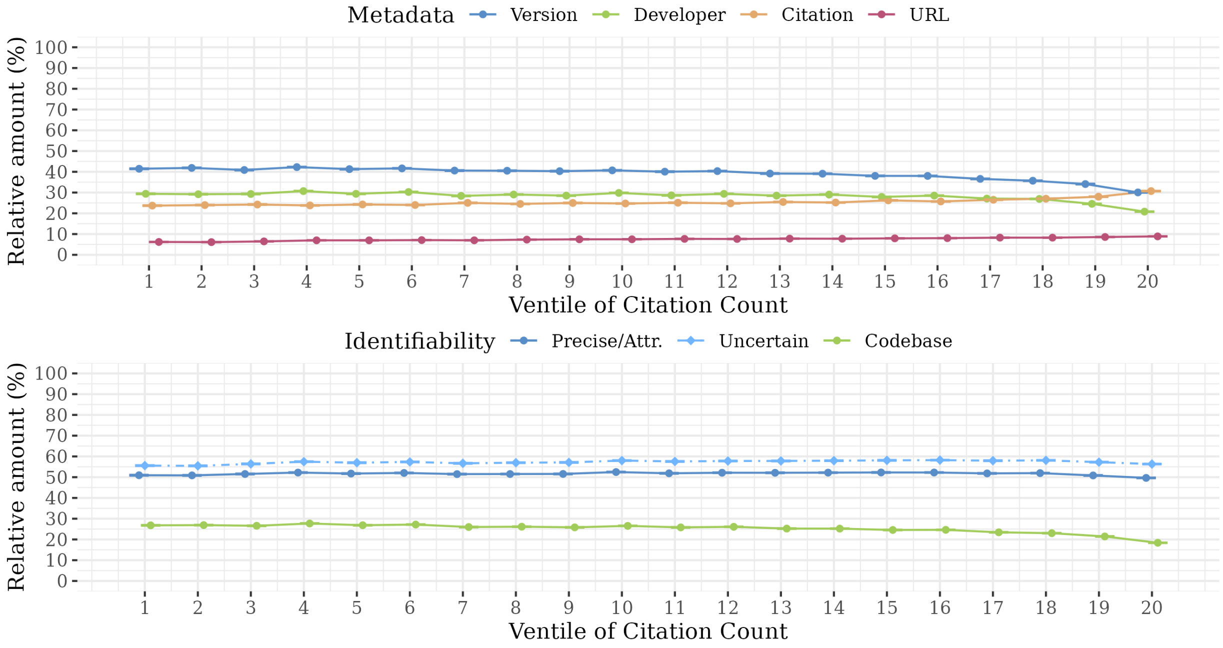


Figure 6.15: Top: relative amount of provided metadata (version, developer, formal citation, and URL) for software mentioned in articles per citation count ventile. **Bottom**: relative amount of identifiable software in terms of software itself (and developer attribution) and codebase identification per citation count ventile. Software identification is split between precise identification and uncertain identification, which does consider identification by only a provided URL. 95% CIs are provided in both plots.

Considering the citation rank depicted in Figure 6.15 (top), differences in provided metadata can mostly be observed in higher ventiles. Regarding version and developer the numbers plateaus for the first 14 ventiles, and start to drop afterwards from values of 39.1% (CI_{95}=[38.9, 39.2]) and 29% (CI_{95}=[28.9, 29.2]) down to 30.1% (CI_{95}=[29.9, 30.2]) and 20.8% (CI_{95}=[20.7, 20.9]) at the last ventile. An inverse trend is present for formal citations from 25.2% (CI_{95}=[25.1, 25.3]) at the 13th ventile up to 30.7% (CI_{95}=[30.6, 30.9]) at the 20th. For URLs there is almost no difference with all values in a range between 6.2% (CI_{95}=[6.1, 6.3]) and 8.9% (CI_{95}=[8.8, 9.0]) at the 1st and 20th ventiles, respectively. Regarding identification (Figure 6.15, bottom), no difference in software identification and developer attribution is present throughout the ventiles, while codebase identification follows version completeness, dropping from 25.2% (CI_{95}=[25.1, 25.4]) at the 14th ventile to 18.4% (CI_{95}=[18.3, 18.5]) at the 20th ventile.

The relation between software availability and provided metadata is illustrated in Figure 6.16, with software divided in three categories based on availability. The set *comm-closed* entails commercial software that is not freely available with closed source code; the set *free-closed* entails software that is freely available but has closed source code; and *free-open* describes freely available software with open source code. It is found that provided metadata strongly differs between these groups. The mention of both versions and developers is most common in set *comm-closed* with values of 64.8% (CI_{95}=[64.8, 64.9]) and 59.1% (CI_{95}=[59.0, 59.2]) for version and developer, respectively. The values decrease to 48.4% (CI_{95}=[48.2, 48.6]) and 30.5% (CI_{95}=[30.4, 30.7]) for *free-closed*, and even further down to 37.1% (CI_{95}=[37.0, 37.2]) and 14.5% (CI_{95}=[14.4, 14.5]) for *free-open*. An inverse trend is present for formal citations where the values increase from 4.8% (CI_{95}=[4.8, 4.9]) in *comm-closed* to 32.4% (CI_{95}=[32.3, 32.6]) in *free-closed*, and 35.1% (CI_{95}=[35.0, 35.2]) in *free-open*.

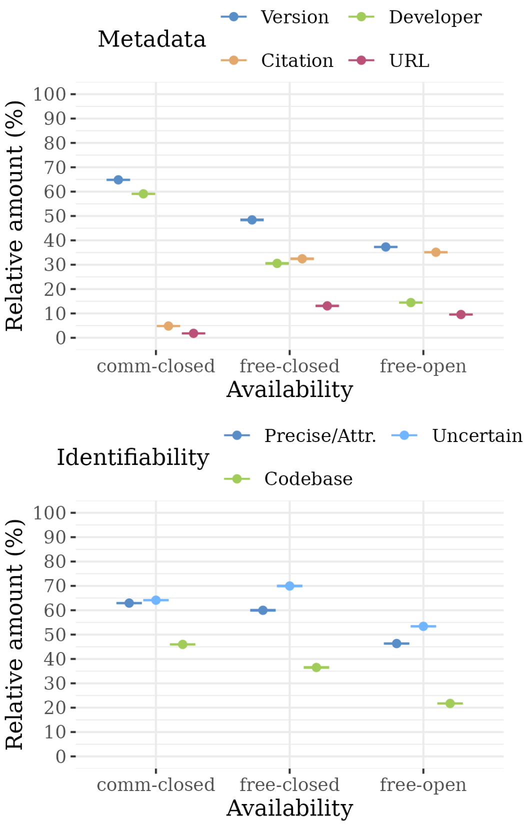


Figure 6.16: Top: relative amount of provided metadata (version, developer, formal citation, and URL) for software mentioned in articles in dependence of software availability. **Bottom**: relative amount of identifiable software in terms of software itself (and developer attribution) and codebase identification in dependence of software availability. Software identification is split between precise identification and uncertain identification, which does consider identification by only a provided URL. 95% CIs are provided in both plots.

URLs are also more commonly provided for free software with 1.8% ($CI_{95}=[1.8, 1.8]$) in *comm-closed* compared to 13.1% ($CI_{95}=[13.0, 13.2]$) in *free-closed* and 9.6% ($CI_{95}=[9.5, 9.6]$) in *free-open*. Regarding identification, *comm-closed* software allows for the best identification of 62.9% ($CI_{95}=[62.9, 63.0]$) compared to 60% ($CI_{95}=[59.8, 60.1]$) for *free-closed* and 46.3% ($CI_{95}=[46.2, 46.4]$) for *free-open*. This is due to the high number of developers provided in *comm-closed*, while the margin is reduced by the higher number of formal citation in free software. However, codebase identification notably decreases from 46% ($CI_{95}=[45.9, 46.0]$) in *comm-closed* over 36.5% ($CI_{95}=[36.3, 36.7]$) in *free-closed* to 21.7% ($CI_{95}=[21.7, 21.8]$) in *free-open*.

Finally, four equal regression models were used to further assess and quantify the relation between the available prior knowledge on scientific articles examined above, and the amount of provided software metadata of version, developer, formal citation, and URL. All models are fitted as logistic regressions to predict the binary target of whether a specific software metadata item was provided in an article. The considered covariates include the article's publication year, its scientific domain, journal rank, and citation count. As before, journal rank and citation count are modeled in ventiles, while individual domains are provided as binary features, and the year as a discrete number. Furthermore, the number of software contained in the article is included as a numerical feature, and the information whether a software is freely available, and whether its source code is available are coded as two separate binary features. The full results for the fitted regression models are available in the Appendix, while effect sizes are provided as odds-ratios (ORs) below. Regarding the version (see supplementary Table A8), the results show a significant and considerable positive influence of the year with OR=1.047 ($CI_{95}=[1.046, 1.048]$) per year.

Further, the publishing domains were found to have a significant influence with strong variations between domains. The journal rank has a significant negative influence with OR=.967 ($CI_{95}=[.967, .968]$) per ventile, while another negative influence of the citation count is found but at a negligible effect size of OR=.997 ($CI_{95}=[.997, .998]$) per ventile. The availability of the software has a high influence on whether version numbers, wither fewer version being provided with OR=.444 ($CI_{95}=[.439, .448]$) for free software and OR=.822 ($CI_{95}=[.814, .831]$) for software which is also open source. Regarding developer mentions (see supplementary Table A9) an influence of the year can be identified with an OR=1.019 ($CI_{95}=[1.018, 1.02]$) per year, while domains are iden-

	Allusion	Creation	Deposition	Usage	**Overall**
Application	13.9	1.9	.6	68.7	85.0
OS	.3	0	0	1.7	1.9
PlugIn	.5	.1	0	5.1	5.9
PE	.4	0	0	6.8	7.2
Overall	15.1	2.0	.6	82.3	100.0

Table 6.3: Overview of the relative frequency of software and mention types as well as their combinations over all software mentions. Note that overall numbers do not necessarily sum to 100 due to rounding.

tified to also influence developer mentions in arbitrary directions based on the discipline. Small of even negligible positive effects are identified for the citation count and journal rank with OR=1.008 (CI_{95}=[1.007, 1.008]) and OR=1.004 (CI_{95}=[1.003, 1.004]) per ventile, respectively. Strong effects can be observed with respect to software availability, with developer being mentioned significantly less for free software at a value of OR=.165 (CI_{95}=[.163, .167]), and for software that is additionally open source with OR=.835 (CI_{95}=[.824, .847]).

Formal citations (see supplementary Table A10) are negatively influenced by publication year with an OR=.968 (CI_{95}=[.967, .969]) per year, and further significantly influenced by most publishing domains in arbitrary directions. Further, the journal rank is found to have a positive influence at OR=1.029 (CI_{95}=[1.028, 1.03]) per ventile, while the citation count is found to have a small negative influence with OR=.987 (CI_{95}=[.987, .988]) per ventile. This is suggested differently by Figure 6.15, which shows a small but positive trend. A strong effect can be observed with respect to freely available software with significantly more formal citations being provided with OR=10.072 (CI_{95}=[9.943, 10.202]). The results further show that software which is also open source received fewer formal citations by OR=.777 (CI_{95}=[.768, .785]). Lastly, for URLs (see supplementary Table A11) the most notable effect results again from free software which positively influences the mention of URLs at a value of OR=12.537 (CI_{95}=[12.296, 12.783]). Similarly, software that is also open source receives fewer URL mentions with OR=.529 (CI_{95}=[.521, .536]).

6.2.4 Types of Software Mention

For each software mention, SoftwareKG contains information about the mention type and the software type. Overall, the most frequently mentioned type of software in the analyzed set of articles is *Application* with 85.0%, followed by *PE* with 7.2%, *PlugIn* with 5.9%, and *OS* with 1.9% of the mentions. The distribution is different considering disambiguated software, where 89.0% of software were found to be *Applications*, 10.3% *PlugIns*, .4% *PEs*, and .3% *OSs*. With respect to the type of mention, it was observed that most mentions of software reflect *Usage* with 82.3%, whereas 15.1% represent *Allusion*, while only 2% and .6% of the software mentions represent *Creation* and *Deposition*, respectively. Table 6.3 gives a fine-grained overview of the relation between mention type and software type. It was found that the mention type *Usage* is prevalent over all software types and the software type *Application* over all mention types. Furthermore, the relative frequency for both *Creation* and *Deposition* is zero for *OSs* and *PEs*. When looking into domain specific differences, it can be observed that Decision Science (45.8%), Mathematics (52.6%), Computer Science (54.7%), and Engineering (56.4%) have the lowest share of *Usage* statements, while disciplines such as Energy (91.8%), Materials Science (92.9%), Nursing (93.6%), Dentistry (94.5%), and Veterinary (95.8%) have the highest share of *Usage* statements. The opposite trend can be observed for software *Allusion*, starting with Veterinary (3.7%) and ending with Decision Science (45.4%). *Creation* statements have the highest proportion in Decision Science with 6.8%

Software	# PlugIn	# Articles	Top five *PlugIns* including relative amount of mentions
R	22,031	228,581	Bioconductor (5%), ggplot 2 (3.7%), lme 4 (3.4%), vegan (2.6%), DESeq 2 (2.4%)
MATLAB	4567	17,748	Psychophysics Toolbox (6.9%), Psychtoolbox (6.1%), Statistics Toolbox (3.8%), Image Processing Toolbox (2.7%), Neural Network Toolbox (1.4%)
Python	3320	11,372	scikit - learn (11.6%), SciPy (4.7%), TensorFlow (3.6%), NumPy (2.6%), scipy (2.4%)
python	1503	3411	scikit - learn (9.9%), scipy (4.3%), numpy (3.1%), matplotlib (2.9%), sklearn (2.7%)
ImageJ	1283	9173	Fiji (38.6%), NeuronJ (3.2%), Cell Counter (3.1%), MTrackJ (2.3%), Analyze Particles (1.8%)
Stata	804	2014	metan (6.3%), runmlwin (3.3%), mvmeta (2.1%), Image Composite Editor (2%), metareg (1.9%)
Perl	787	1037	MISA (3.8%), speaks - NONMEM (2.7%), DynamicTrim (1.3%), NONMEM (1.2%), Bioconductor (1.2%)
Excel	643	1735	XLSTAT (18.5%), nSolver (4.3%), Microsatellite Toolkit (2.9%), Analysis ToolPak (2.1%), @ Risk (2%)
Cytoscape	561	5092	ClueGO (13.3%), MCODE (11.6%), NetworkAnalyzer (8%), BiNGO (7.5%), Enrichment Map (6%)
SPM	529	2424	DARTEL (17.1%), MarsBaR (10%), Marsbar (3.1%), DPARSF (2.8%), CONN (2.8%)

Table 6.4: Most frequent host-software, i.e., mentioned together with a *PlugIn*, in combination with the most frequently used *PlugIns* for each of them. # PlugIn: distinct, disambiguated *PlugIns*; # Mentions: number of distinct mentions of the host-software with a *PlugIn* in articles.

of mentions, followed by Mathematics with 6%, Engineering (5.4%), and Computer Science (5.4%). In Veterinary and Dentistry only .4% and .3% are *Creation* statements. Decision Sciences (2.0%), Mathematics (1.6%) and Computer Science (1.4%) have the highest share of *Deposition* statements, in contrast, less than .1% of software mentions are *Deposition* statements in Materials Science, Veterinary, and Dentistry.

The software type *PlugIn* plays a special role, as it can only be used together with another software, requiring the mention of both, the host-software and the *PlugIn*. Table 6.4 lists the top ten host-software together with the number of *PlugIns* identified for this software, the overall amount of *PlugIn* mentions for the software, and its top five *PlugIns*. *R* was found to be by far the software with most *PlugIns* (22,031 distinct *PlugIns*), followed by *MATLAB* and *Python*. *Bioconductor*, *DESeq2*, *ggplot2*, *lme4*, and *limma* were identified as most frequently mentioned *PlugIns* for *R*, covering 17.1% of overall mentions of *R* in combination with a *PlugIn*. Note that the two different spellings “Python” and “python” listed in Table 6.4 were not linked together but reflect a similar distribution of *PlugIns*.

Software Creation and Deposition

Software *Usage* and *Deposition* statements are often accompanied by URLs to provide a location to access a software and make it findable for the scientific community. Table 6.5 shows the most common domains of URLs mentioned in combination with software *Usage* and software *Depositions*. *Usage* domains (right) correspond to specific software, for instance, `blast.ncbi.nlm.nih.gov` or `r-project.org` for *BLAST* and *R*, but also to software repositories such as Github and specific software package repositories such as CRAN. For *Depositions* (left) it was found that most of the URLs point to source code and software repositories. Github (`github.com`) is the most frequent domain with 14% of overall URLs, but other public repositories such as SourceForge, BitBucket or

Deposition			*Usage*		
URL	Absolute	Relative	URL	Absolute	Relative
github.com	7956	12.2	github.com	23,232	4.5
journals.plos.org	5265	8.1	ncbi.nlm.nih.gov	16,966	3.3
plosone.org	986	1.5	r-project.org	12,712	2.5
sourceforge.net	865	1.3	ebi.ac.uk	9813	1.9
cran.r-project.org	674	1.0	blast.ncbi.nlm.nih.gov	8310	1.6
bioconductor.org	612	1.0	pacev2.apexcovantage.com	7258	1.4
ncbi.nlm.nih.gov	504	.8	cbs.dtu.dk	6324	1.2
ebi.ac.uk	501	.8	cran.r-project.org	6232	1.2
bitbucket.org	376	.6	fil.ion.ucl.ac.uk	6186	1.2
code.google.com	344	.5	targetscan.org	5447	1.1

Table 6.5: Top ten most frequent URLs accompanying software *Deposition* and *Usage* statements together with their absolute and relative frequencies.

Google Code are present as well as repositories focused on packages such as CRAN and BioConductor. Note that there are also two URLs that result from extraction errors due to journal specific formatting added in PLoS articles (`journals.plos.org`, `plosone.org`).

Another aspect of recognizing software *Creation* and *Deposition* statements in articles is that they allow to identify journals that are most frequently used for the description of new software. By analyzing the relative number of articles per journal that contain either a software *Creation* or *Deposition* statement, the most active journals for software description were identified. Considering only journals with at least ten articles, *JOSS* has the highest relative amount of articles introducing software, with 88.8% of the articles containing software *Creation* or *Deposition* statements. It is followed by the journal *Source Code for Biology and Medicine* with 85.6%, *Database: the journal of biological databases and curation* with 82%, and *Proceedings of the VLDB Endowment* with 81.5%. The journal *Bioinformatics* also has a high ratio of 79% and contains a high number of 2222 publications in comparison to the other described journals. Overall, 4122 of 10,805 (38.1%) journals contain at least one article with either *Creation* or *Deposition* statements.

Summary

The analysis described above has focused on software mentions and its citation quality in scientific articles, specifically along the dimensions of time, scientific domain, and article impact. Furthermore, specific aspects such as the most common URLs used for software publication, or the most popular journals for software publication were analyzed. Before the limitations and results of the analysis are discussed, a specific case study regarding the relation between software and article retraction is introduced.

6.3 Software and Article Retraction

SoftwareKG and the methods described in Chapter 4 have further been utilized to systematically investigate the interaction between software and article retraction, to demonstrate how the tools developed in this work can be used to gain further insights on the impact of software on science. Article retraction was chosen as the subject for this case study, because prior work has found that software has influenced study results and has led to article retractions in the past. The corresponding literature was introduced in Chapter 1, and the connection was further highlighted through some manually gathered examples from the Retraction Watch (RW) database, which showed that unintended behavior of software has significantly impacted study results. However, while this anecdotal evidence for the relation between software and retraction exists, there has been no systematic analysis of the differences in the scientific software landscape of retracted articles in the existing literature on software in science (see Section 2.1).

Retraction is a self-correction mechanism established in the scientific community that occurs after articles have passed their peer review and have been published [5]. The number of article retractions has increased throughout the last decades in absolute and relative numbers [287, 268, 256, 202]. Different reasons for this increase have been described [270], one of which is lower barriers to publishing flawed articles. For retraction itself, there are also several possible reasons. The two most prominently investigated reasons are *Misconduct* and *Errors*, where *Misconduct* refers to scientific fraud and has been suggested to be performed deliberately [269]. Early work indicates that *Error* is the most common reason [268], while later results suggest that *Misconduct* is more likely as its number was previously underestimated [83]. It has also been suggested that retractions are more likely to occur in high-impact journals as their articles receive more attention and more rigorous screening after publication [56]. Moreover, recent work suggests that more articles might need to be retracted when viewed based on the criteria from the Committee on Publication Ethics, a non-profit collective in Eastleigh, UK [202].

Here, the relationship between software and retraction is analyzed in two steps. Information available from SoftwareKG is used directly by examining all covered retraction notices for software mentions. Then, the landscape of software is compared between 3,271 retracted articles and 32,710 non-retracted control articles, which were selected specifically for this analyses and independently of SoftwareKG.

6.3.1 Retraction Notices

Retraction notices are released by scientific publishers to announce the retraction of scientific articles and the corresponding reasons. Explicit mentions of software within these notices could mean that software had a direct influence on article retractions and give valuable insights on the impact of software on science. Retraction notices were directly analyzed for software mentions to investigate this relation between article retraction and software. The information and content of retraction notices varies depending on publishers and editors. They have been described as ranging from informative and transparent to deeply obscure [287].

Since retraction notices are publisher-specific, they cannot be easily automatically mined and accessed. However, some notices are available from PMC OA and, therefore, represented in SoftwareKG. These notices were automatically identified by the publication type *skg:PubType Retraction*, with the corresponding data collection process illustrated in Figure 6.17 (A and B). Overall, 1,861 retraction notices were covered in the analysis, which contain a description of why an article was retracted. The information on software contained in these notices was directly queried from SoftwareKG, all cases where software was identified were manually verified, and the correspond-

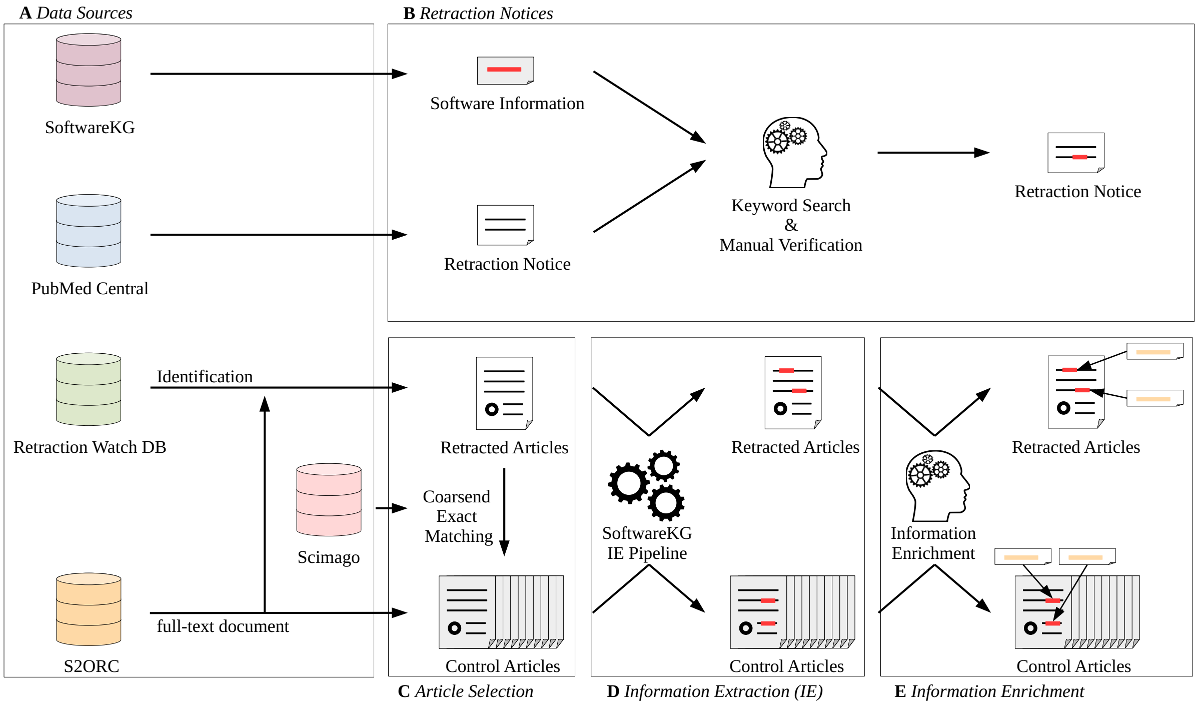


Figure 6.17: Flowchart illustrating the data collection pipeline for retraction notices and full-text documents of retracted and control articles. **Top**: Pipeline for retraction notices. **Bottom**: Pipeline for article selection and IE from articles.

ing mention context was manually examined to classify the reason why the software was stated. Additionally, all full-text documents retraction notices were re-examined for the term *software* to identify generic references to software that would not be covered in SoftwareKG. However, no such cases relevant to the analyses were identified.

Results

Overall, 16 (.9%) retraction notices out of 1,861 were found to contain software mentions, where the software is related to the articles' retraction[6]. The corresponding reasons why the software was mentioned were manually examined and are summarized in detail in the following:

- **Software Errors** The most common reason why software is mentioned within these notices are biased study results caused by errors in software usage. This affects nine notices, with six describing a mistake using point-and-click software concerning SPSS [10], Excel [123], Photoshop [163], uVariants and uDesign [147], and Gene Expression Omnibus [169]. Another three notices are related to programming where source code errors led to data processing errors, twice for Matlab [275, 181] and once for Perl [102]. One software was not specified [100]. Eight of the nine retraction notices described above clearly state how the error was discovered. In four cases, the software error was identified by authors after publication. In three cases, abnormalities in the research papers were found by others, and the error itself was recognized

[6] Additional software unrelated to the articles was discovered, which are not listed here, e.g., software employed for automatic plagiarism testing.

by the authors afterward. In one case, inaccuracies were identified by others without further elaboration.

- **Study Design** There are four instances in which the study design, including the use of specific software, led to unreliable study results. One retraction notice described the study design, including the use of the software DaysyView, as unreliable [145]. Furthermore, two cases describe inconsistent published data concerning Excel [197] and TreeBASE [140]. Another notice suggested application of BLAST for validation, which would have revealed a data error [189].

- **Journal Policy** Two notices reported cases in which the software publication violated the journal policy, leading to the retraction of the corresponding software articles. The software GNARE required user registration even though the journal demanded unrestricted access [274]. The second notice affects TREEFINDER, where the lead author changed the license after publication, no longer allowing usage of the software in the US and multiple European countries. At the same time, the journal policy requires the software to be available for all scientists [131]. Notably, the TREEFINDER publication had a high impact with 833 citations indexed on Web of Science from its publication in 2004 to its retraction in 2015, with further citations indexed after the retraction.

- **Reporting Error** One notice remarked the wrongly reported version of SPSS as a contributing reason for retraction [320].

With respect to the retraction reasons introduced in Section 6.3.2, 13 out of the 16 cases are classified as honest *errors*, while the instance regarding *Photoshop* is labeled as a *misconduct* and intentional altering of study results. The remaining instances correspond to category *other* and concern the cases where authors violated journal policies.

6.3.2 Software Landscape in Retracted Articles

The second part of the analysis investigates the relation between article retraction and scientific software with respect to differences in the software landscape. Even when software is not directly stated as a reason for retraction, differences in software usage practices can still be related to retractions. To this end, software mentions were identified within a set of retracted articles and compared to a control set of non-retracted articles, selected by Coarsened Exact Matching (CEM). The results were analyzed with respect to the software used, its availability, and with respect to software citation habits by analyzing the available information associated with software. The data processing pipeline for this part of the analysis is also illustrated in Figure 6.17.

Dataset

Initially, a list of retracted articles to analyze was gathered with the utilized data sources illustrated in Figure 6.17 (A), because a broad automatic identification of retracted articles is not possible due to the problems described in Section 6.3.1. The RW database was used as a knowledge base on article retractions. It contains manually gathered information on retractions of scientific articles, including a fine-grained summary of the corresponding reasons leading to the retraction. Altogether, the database obtained from RW on January 6th, 2022, includes 32,127 entries ranging from 1756 to 2022. Here, the analysis was limited to articles published between 2000 and 2019. Earlier publications were excluded because too few data samples are available per year, while later publications are excluded because full-text information is only available up to 2019 for control articles (see below).

For the analyses, it was necessary to obtain the full-text documents of the retracted articles. This is a challenging task, as scientific publications are often pay-walled or only available in PDF format. Therefore, it was necessary to integrate the S2ORC corpus for this analysis because otherwise the coverage would have been insufficient. The S2ORC corpus [170] contains numerous English academic papers with metadata and plain, full-text documents for a subset of papers, which were extracted from PDF documents. When S2ORC was published, it contained 8.1M full-text documents, but version 20200705v1 used here was further extended to cover more than 12.7M articles published before April 2020. Articles without available full-text document could not be considered for further analysis. The identifiers from the RW database were then matched against S2ORC to obtain full-text documents of retracted articles. Moreover, as described in Section 4.8.1, the Scimago Journal Rank (SJR) [255] was added for all articles based on journal information contained in S2ORC.

Overall, a set of 3,374 retracted articles with available full-text documents and journal information were identified. To perform a valid statistical analysis with respect to software usage and software citation habits they were compared against a suited control set of non-retracted articles (see Figure 6.17 (C)), selected by CEM [124]. The idea is to generate a set of control articles for the given retracted articles, which is equally distributed in all variables that are attributed an influence on the dependent variable. The results previously presented in this chapter have shown that three variables influence software usage and citation habits: *Publication Date*, *Scientific Domain*, and *Journal Rank*. Therefore, these variables are controlled on article basis as follows: *Publication Date* is coarsened to the year of publication; *Scientific Domain* is matched exactly, keeping the order of domains intact under the assumption that the first named domain is the most influential for multidisciplinary work; and *Journal Rank* is coarsened to percentiles. The specific publishing journal is not controlled based on the argument that the influence of journals on software citation is due to the *Scientific Domain* of the journal and *Journal Rank*. All required information for CEM is available in the outlined data sources. Information on publication date is available from S2ORC, which also contains information on the underlying research domain of articles. The journal rank was based on the added SJR information, containing year based information on journal ranks. It was used to sort the journals and to divide them into percentiles. In difference to the previously described analyses, the journal rank was not calculated accounting for discipline, because the discipline is also added as a separate variable for CEM. Articles for which any of the information was not available were excluded from the subsequent analyses.

Based on the controlled variables, ten control articles from the available S2ORC data were randomly selected without replacement for each retracted article. This was not possible for all articles as rare combinations of the controlled variables occurred in the RW data. Overall, 96.9% (3,271 out of 3,374) of articles were matched, while the remaining unmatched articles were excluded from further analyses, resulting in a control set of 32,710 articles. As ten control articles correspond to each specific retracted article, arbitrary subsets of the retracted articles can be analyzed independent of their distribution, which is necessary, as, for instance, certain retraction reasons might be more likely in specific scientific fields.

Information Extraction and Enrichment

The IE pipeline was applied as described in Section 4.8 for both retracted and control articles as illustrated in Figure 6.17 (D). However, since the corresponding full-text documents in S2ORC were extracted from PDF documents, it could be the case that a higher amount of errors is generated by the IE pipeline due to PDF extraction artifacts in the corpus. Therefore, additional manual quality control was performed to specifically reduce the influence of large-scale errors, with details described below. Moreover, it can be argued that errors—especially on a small scale—do not bias results of comparisons for retracted and control sets because errors are equally likely to be present in both sets.

Information enrichment regarding software availability was performed for the new dataset as described in Section 6.3.2. It was updated for this analyses to also cover the 50% of most common software in this dataset, resulting in the annotation of 243 software. The corresponding data collection step is illustrated in Figure 6.17 (E). During information enrichment, the validity of the results for the most common software were manually checked for: (1) whether extracted names do refer to software and (2) whether there are ED errors for software identities. Two large-scale extraction errors were identified where methods were systematically confused with software. Further, ED errors were identified: twelve false negative errors, where two software tools were not disambiguated; ten large-scale false positive errors, where two software were wrongly disambiguated; and 14 small-scale false positives, where single occurrences were incorrectly accounted to a software. All described errors were manually updated before executing the statistical analyses. It is assumed that potential further errors will be contained in the long tail of software names as they are likely to result from extraction errors that are unlikely to be disambiguated to other software. On the other hand, the long tail may contain further mentions that should be disambiguated to other software, but since the enrichment covers more than 50% of overall software mentions, these can only have a limited impact on the results. Therefore, the reported results for the most common software and software availability are considered as valid.

Furthermore, the information on retraction reasons, which closer identify the circumstances of an article retraction, was extended. There is a large number of different reasons, which can, for instance, vary due to differing communications about the retraction by publishers. To analyze the relationship between the retraction reason, the software usage, and the citation habits, the reasons are summarized as done in prior work [230]. For this purpose, top-level categories were manually determined by analyzing the reason descriptions provided by RW, e.g., all reasons indicating plagiarized work were summarized, including text, images, and data. Reasons that could not be semantically assigned to any top-level reason were summarized as *other*, which includes reasons such as *Ethical Violations*, *Rogue Editor*, or *Copyright Claims*. A detailed overview of the summarized reasons can be found in Table A14.

Results: Software Mentions in Retracted Articles

In general, retracted articles are slightly more likely to mention software than articles from the control set, with 63.2% (CI_{95}=[61.5, 64.8]) of the retracted articles containing at least one software mention and 58.1% (CI_{95}=[57.6, 58.6]) of control articles[7]. A McNemar's χ^2 test for 32,710 paired samples showed that this difference is significant, $\chi^2(1, N{=}32{,}710){=}200.21, p{<}.001, OR{=}1.27$. In both sets, the relative number of articles increased from 2000 to 2019, from 18.8% (CI_{95}=[0, 37.9]) to 76.7% (CI_{95}=[72.0, 81.3]) for retracted and 35.0% (CI_{95}=[27.6, 42.4]) to 63.3% (CI_{95}=[61.6, 65.0]) for control articles. Further details can be found in the Figure A4.

This analysis is further extended to cover specific retraction reasons. As described above, the characteristics of different retraction reasons were analyzed by comparing the articles retracted for a particular reason to the corresponding control set created by selecting the ten control articles for each retracted article. This results in a separate set of retracted and control articles for each retraction reason and ensures that both sets are equally distributed for every comparison. The amount of articles containing software per retraction reason is illustrated in Figure 6.18 and reveals further differences in software usage. For most reasons, the overall trend can be observed with a higher relative number of articles containing software in retracted articles. An exceptionally high proportion

[7] CIs that are provided in figures are sometimes not given in text for better readability.

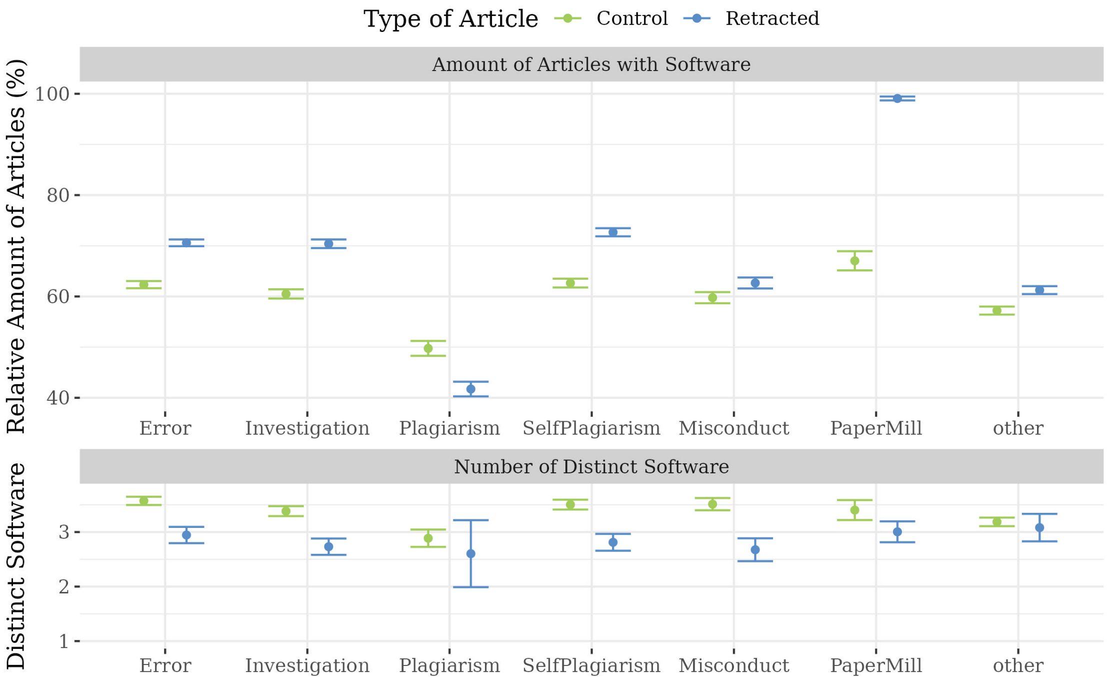


Figure 6.18: Software mentions in scholarly articles per retraction reason separated by retracted and corresponding control articles. The sets of control papers are constructed by selecting the ten corresponding articles for each retracted article. **Top**: Proportion of articles that contain at least one software mention. **Bottom**: Average number of software mentions per article with at least one software mention. Error bars indicate 95% CIs.

can be observed for *PaperMill* articles, with more than 99% of the articles containing software. At the same time, large proportions are also present for *Error*, *Investigation*, and *SelfPlagiarism*. In contrast, articles retracted due to *Plagiarism* are less likely to contain software.

Looking at articles that mention at least one software, it was observed that retracted articles contain less software with an average of 2.92 (CI_{95}=[2.79, 3.05]) software compared to 3.32 (CI_{95}=[3.27, 3.38]) in control articles, a difference found to be significant, $t(2765){=}5.6, p{<}.001$, with a negligible effect of Cohen's d=.11. In both sets, the frequencies increased similarly over the analyzed years. For further details see Figure A5. Similar to the overall result, retracted articles were found to contain less software mentions across all individual retractions reasons except for *Plagiarism* and *other* where no difference was found, illustrated in Figure 6.18.

Besides the plain number of software per article, the frequencies for the mentioned software also differ in retracted articles, illustrated in Figure 6.19. *SPSS* is the most frequently used software in both sets, but there is a high difference in the number of articles mentioning it, with 35.8% of retracted and 20.3% of control articles, respectively. *Prism* and *ImageJ* are also central software in both sets, which are mentioned more in retracted articles with 13.3% and 12.7% compared to 9.8%, 8.4% in the control set. A high difference can be observed for *R*, used in only 4.6% of the retracted articles, while being the third most common software in the control set (9.0%). *TargetScan* is a noticeable case as it is mentioned in 7.9% of retracted articles but only in 1.0% of control articles. A closer analysis concerning retraction reasons revealed that *TargetScan* is mostly mentioned within articles retracted due to *Error*, *Investigation*, *SelfPlagiarism*, *other* and especially in *PaperMill*, where 39.7% (CI_{95}=[33.2, 46.3]) of retracted and 3.7% (CI_{95}=[2.8, 4.7]) of control articles mention the software.

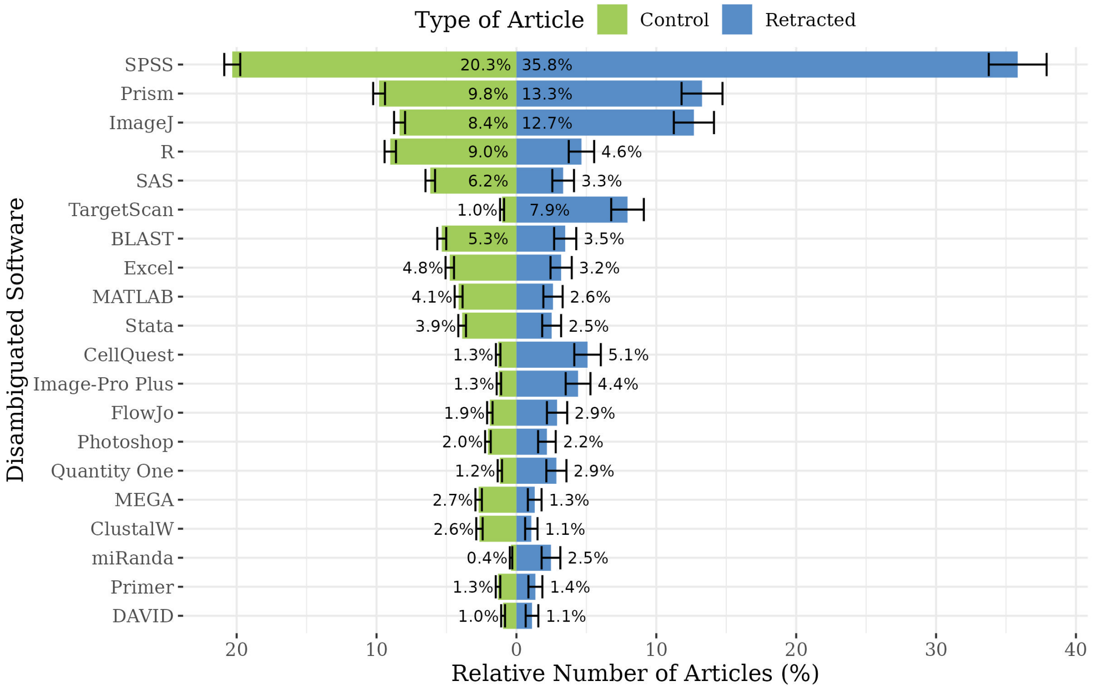


Figure 6.19: Proportion of retracted and control articles mentioning software out of the top 20 most used software. Error bars indicate 95% CIs.

A closer look at the most common statistic software, illustrated in Figure A6, showed that the distribution is strongly skewed towards *SPSS* and *Prism* in the retracted set. In contrast, a more diverse set of tools is employed in control articles, indicated by higher usage of *R*, *SAS*, *Excel*, *Matlab*, and *Stata*. Therefore, the overall distribution of software was further investigated by analyzing how many articles mention any of the top n software. By analyzing the cumulative distribution of the n most frequently used software, it was found that less software is used in a higher number of articles in the retracted set (see Figure A7). In detail, for retracted articles, the top 1 software covers more than 25% of articles while the top 2 software are required to cover 25% in the control set. This difference increases to top 3 compared to top 7 software at 50% of articles and top 21 to top 74 at 75%.

As observed above, software is often not mentioned by its original name, but authors use spelling variations, abbreviations, or full forms. It was analyzed how individual software is indicated in retracted articles but no systematic trends were observed. Within retracted articles, for instance, *SPSS* is mentioned by a less diverse set of names in comparison to control articles and mainly indicated by its standard name, while an inverse trend exists for *ImageJ*. Mentions for *Quantity One*, on the other hand, are similarly distributed. The results are also illustrated in Figure A8.

To assess the state of free and open source software in retracted articles, the differences in the use of free and commercial and open and closed source software were analyzed, based on the manually added enrichment information. As found in large-scale analyses (see Section 6.2.2), a considerable overlap exists between free and open source software in the set of 243 annotated software, with 68% of free software being also open source, and 99% of open source software being free. The results are illustrated in Figure 6.20 and show that retracted articles utilize less free and open source software than control articles.

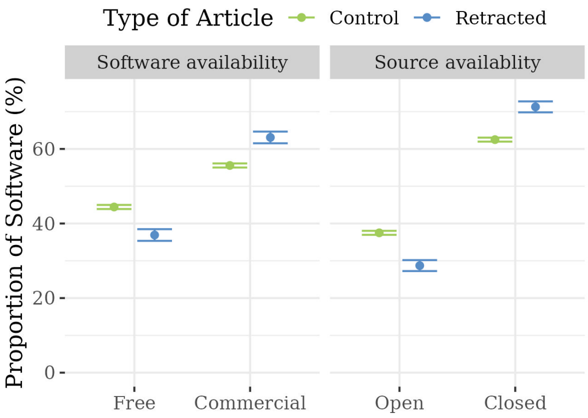


Figure 6.20: Proportion of free or open source software across retracted and control articles. Error bars indicate 95% CIs.

The same trend for free and open source software can be identified across all retraction reasons with exceptionally high differences for *Error* with 38.1% (CI_{95}=[36.0, 40.1]) compared to 47.2% (CI_{95}=[46.5, 47.9]), *Investigation* with 37.7% (CI_{95}=[35.0, 40.3]) to 46.9% (CI_{95}=[45.9, 47.9]), *Misconduct* with 36.1% (CI_{95}=[32.6, 39.6]) to 49.2% (CI_{95}=[48.1, 50.4]), and *SelfPlagiarism* with 34.0% (CI_{95}=[31.6, 36.3]) to 46.5% (CI_{95}=[45.6, 47.4])[8]. The corresponding results are also illustrated in Figure A9. To estimate the difference of software availability for free and open source software in retracted articles, logistic regressions were employed with the following covariates: retraction, number of software per article, and their interaction, see Tables A12 and A13. The number of software per article was included based on the findings of Section 6.2.2. It was found that control articles use significantly (OR=1.35, CI_{95}=[1.15, 1.57]) more free and open source software than retracted articles. Further, the results confirm the findings of Section 6.2.2, that the ratio of free and open software increases with higher number of software per article (OR=1.51, CI_{95}=[1.45, 1.58] per additional software).

Lastly, retracted articles utilize fewer *PlugIns* with 3.7% (CI_{95}=[3.2, 4.2]) compared to 6.0% (CI_{95}=[5.8, 6.2]) in the control set and *PEs* with 3.4% (CI_{95}=[2.9, 3.8]) to 5.3% (CI_{95}=[5.2, 5.5]), but more *Applications* with 89.8% (CI_{95}=[89.1, 90.6]) to 86.4% (CI_{95}=[86.1, 86.7]) as well as OSs with 3.1% (CI_{95}=[2.7, 3.6]) to 2.3% (CI_{95}=[2.2, 2.5]). These results are also illustrated in Figure A10.

Results: Citation Habits in Retracted Articles

This section highlights the results for analyzing software citation practices between retracted and control articles. While the analyses outlined in Section 6.2.3 have focused on in-depth details of software citation, the goal of this analysis is to perform a comparison that highlights the main differences in citation habits. Therefore, broader citation styles are used to provide an easy comparison of the main differences, similar to the prior work of Howison and Bullard [118]. For this purpose, the following four citation styles are considered:

- *No Info* describes software mentions without further metadata;
- *Complete Info* describes mentions with a developer and version number, also typical for mentioning scientific instruments and described by Howison and Bullard [118];
- *Partial Info* describes partial metadata with either version or developer provided;
- *Formal Citation* describes cases in which a formal citation for the software was provided (see Section 1.3). To facilitate the analysis, the categories are defined as mutually exclusive, by considering all cases which provide a formal citation in this category, regardless of what other information they provide.

[8] Numbers are provided for free software

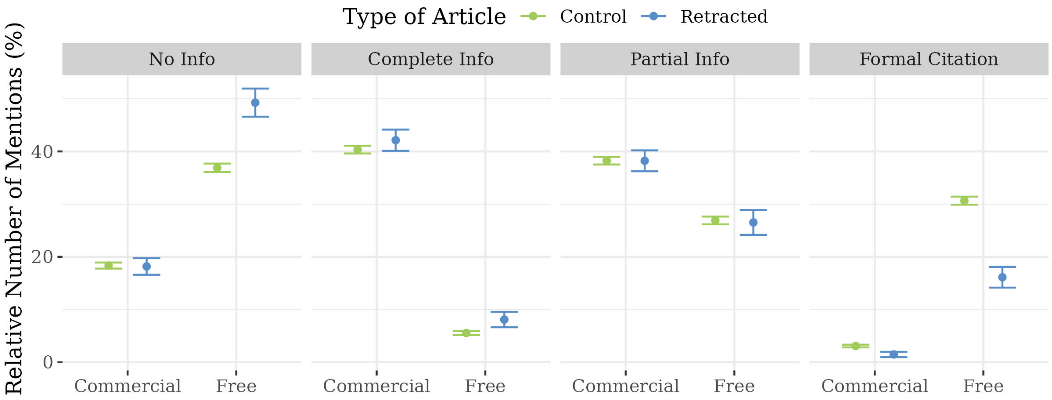


Figure 6.21: Proportion of software mentions across different levels of citation completeness, separated by retracted and control articles, considering whether software is commercial or freely available. No Info: Neither the version, nor the developer of a software are provided; Incomplete Info: Either version or developer is provided; Complete Info: Version and developer are provided; Formal citation: software mention is accompanied by bibliographic citation. Error bars indicate 95% CIs.

Substantial differences in citation habits were identified between retracted and control articles. Both sets mention a comparable amount of software without providing further information, with 36.5% (CI_{95}=[35.2, 37.7]) for retracted and 34.8% (CI_{95}=[34.4, 35.2]) for control set. The number of informal citations is higher in retracted articles with 21.4% (CI_{95}=[20.3, 22.4]) compared to 16.1% (CI_{95}=[15.8, 16.4]) in control articles. On the other hand, retracted articles provide fewer formal citations with 10.7% (CI_{95}=[9.9, 11.5]) compared to the control group with 20.5% (CI_{95}=[20.2, 20.9]). The overall numbers are also illustrated in Figure A11.

Since both, prior work [118, 75] (see Section 1.1) and the analysis performed in Section 6.2.3, showed that citation practices differ between free and commercial software, this difference was further analyzed concerning retracted articles. All trends described in the following concern free and commercial software, but are also present for open source and proprietary software, with details given in Figure A12 and Figure A13. The results are provided in Figure 6.21 and show that commercial software is similarly cited in both retracted and control articles. At the same time, there is a notable difference in the citation style for free oftware. Commercial software is most frequently mentioned similarly to scientific instruments by providing version and developer. It is also a common practice to only mention incomplete information by providing either version or developer. On the other hand, formal citations are almost never used in only 1.5% and 3.1% of instances in retracted and control articles, respectively.

Free software is most commonly mentioned without any information in both sets, with a higher proportion of 49.3% in retracted articles compared to 36.9% in control articles. It is only rarely mentioned similar to scientific instruments with 8.1% and 5.5% in retracted and control articles, respectively, but partial information is more commonly provided in 26.5% and 26.9% of cases. A high difference can be observed for formal citation, with only 16.1% of retracted articles providing formal citations while 30.7% of control articles formally cite free software. To estimate the difference of formal software citation in retracted articles, a logistic regression was employed with the following covariates: retraction state, number of software per article, availability of the software in terms of free and open source software, and all pairwise interactions, see Table 6.6. The results show that control articles are significantly more likely to formally cite software usage than retracted articles (OR=2.89, CI_{95}=[1.94, 4.30]). Further, the test also showed that, independently of other covariates, free and open source software are more likely to receive formal citations compared to

Variable	OR	95% CI	
Open	13.35	6.90	25.85
Free	8.02	4.81	13.36
not retracted	2.89	1.94	4.30
not retracted : Free	1.53	(.93	2.51)
# Software	1.28	1.20	1.37
Free : # Software	1.00	(.95	1.05)
Open : # Software	.99	(.95	1.02)
not retracted : # Software	.89	.84	.94
not retracted : Open	.76	(.52	1.12)
Open : Free	.10	.06	.16

Table 6.6: Result of the logistic regression predicting the mention of formal software citation for retracted papers considering the number of software per article (# Software), the retraction status (not retracted), the software availability (Free; Open), and their interactions. OR=Odds Ratio; OR>1 indicates that a variable is associated with a higher likelihood of formal software citation; non significant values are indicated by the CI including 1, which is also highlighted by the use of brackets; the table is ordered by OR.

commercial and closed source software (free: OR=8.02, CI_{95}=[4.81, 13.36]; open source: OR=13.35, CI_{95}=[6.90, 25.85]), which confirms previous studies [118, 75] and the results of Section 6.2.3. Lastly, the ratio of formal citation was found to increase with a higher number of software per article (OR=1.28, CI_{95}=[1.20, 1.37] per additional software), independently of other covariates.

Based on the initial findings, the citation habits for free and commercial software were further analyzed with respect to specific retraction reasons, summarized in Figure 6.22. It was found that commercial software is mentioned similarly between retracted and control articles, consistent with the overall trend except for articles retracted due to reason *PaperMill*, where software is notably more likely to be mentioned similar to instruments with 69.0% to 54.7%.

Regarding the difference in formal citation of free software, the highest divergence was found for retraction reason *PaperMill* with 0% to 28.5% for retracted and control articles, respectively. Large differences are also present for *Investigation* (8.7% to 30.3%), *Misconduct* (11.0% to 31.4%) and *SelfPlagiarism* (6.1% to 33.6%), while *Error* (16.7% to 31.9%) is close to the overall trend. A smaller difference is present for *other* (19.6% to 29.6%), while no difference can be observed for *Plagiarism*.

6.4 Limitations

The analyses presented in this chapter are based on a complex IE pipeline, where each step is subject to limitations that have been discussed in detail in Section 4.6. It should particularly be noted, that all CIs provided for statistical results in this chapter only express the certainty based on the extracted information. Uncertainty resulting from the IE pipeline, however, is not represented, but does influence the results of the performed analysis. The problem is highlighted, since errors in the IE pipeline have been identified in both the large-scale analysis and the case study on article retraction. Large-scale extraction errors were found to exist for journal specific layout elements, for instance, regarding the publisher PLoS, while errors were also identified for ED (see Section 6.3.2). Furthermore, it can be expected that NER and ED errors exist in the long tail of software mentions, since the high number of 552,105 disambiguated software was identified. While such a high number of software does exist, e.g., with >200 k projects covered in the PyPi python repository as of January

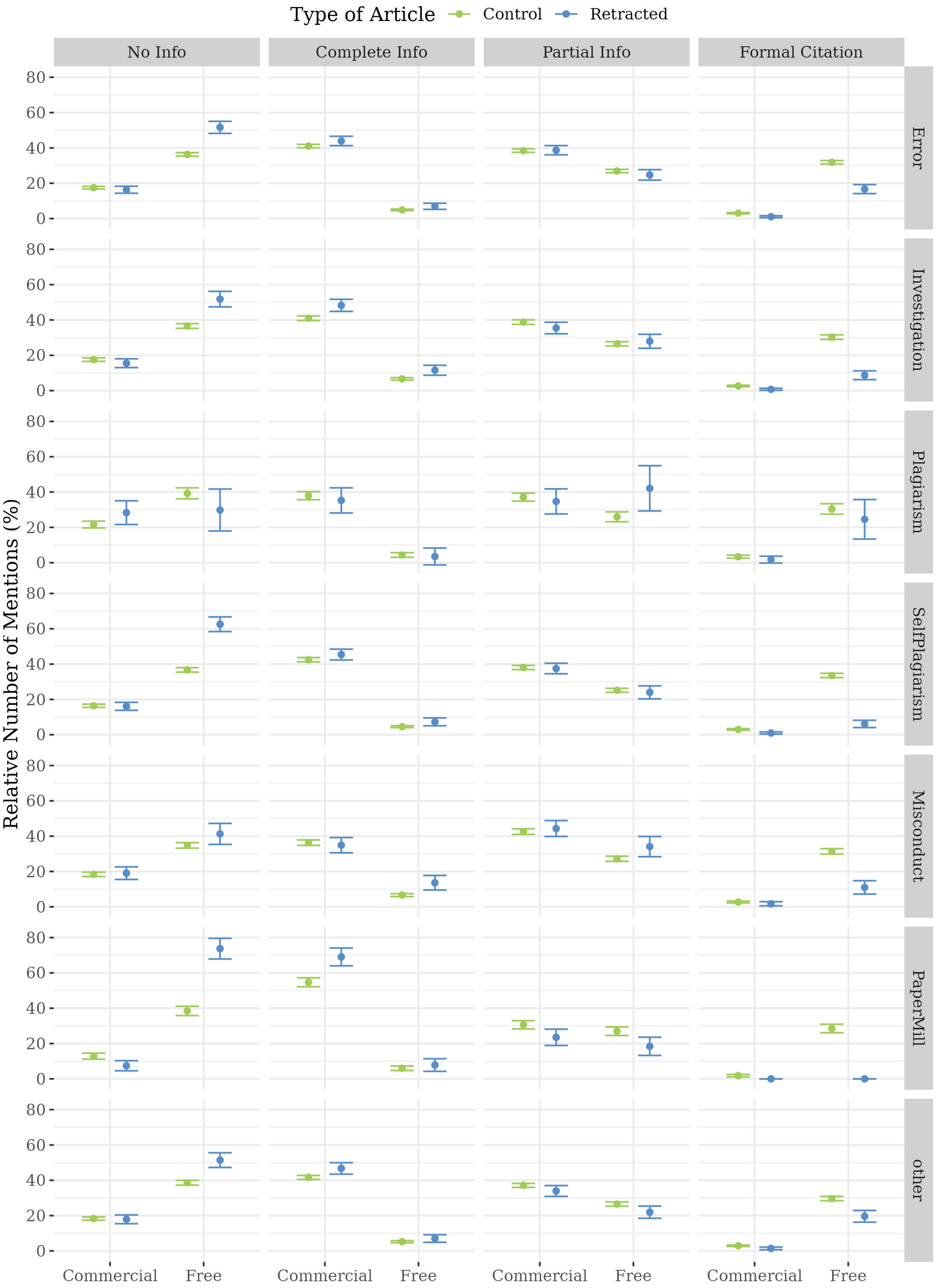


Figure 6.22: Proportion of software mentions across different levels of citation completeness per retraction reason, separated by retracted and control articles, considering whether software is commercial or freely available. No Info: Neither the version, nor the developer of a software is provided; Incomplete Info: Either version or developer are provided; Informal Citation: Version and developer are provided; Formal citation: software mention is accompanied by bibliographic citation (independent from any associated information). Error bars indicate 95% CIs.

2020 and >500 k in February 2024[9], here it is assumed that the number is overestimated.

The articles in SoftwareKG cover a broad range of scientific disciplines, however, the selection of PMC OA as primary data source implies a bias towards Life Sciences and open-access, as discussed in Sections 3.4 and 4.6. Therefore, the trends reported here, might be different when looking at domains such as Computer Science in general. For instance, BLAST is the second most used software in Computer Science, which would likely not be true when looking at Computer Science in general, as this software is primarily used in Bioinformatics. Moreover, results regarding the mention and citation practices of open source software might be overestimated. While the trends were corroborated by the case study regarding article retraction, here the article selection is based on the composition of S2ORC, which also has a dominant bias towards open-access publications.

Several performed analyses rely on external data and are, thus, influenced by their quality. For instance, domain and journal rank information based on Scimago data were only available for 93.3% and 67.8% of articles included in the analyses, respectively, while 85.9% of articles had a citation count >0 and were included in the analyses. For citation impact analysis, articles without a citation count were not considered since it could not be clearly determined if they were missing or never cited. Furthermore, the case study was also dependent on external data sources which strongly restricted its coverage. Full-text and journal information could only be matched for ≈10% of articles contained in the RW database (see Section 6.3.2), because a high number of publications is hidden behind pay-walls. Due to this reason, the overall sample size of retracted articles is also too small to implement specific analyses, such as investigating disciplinary differences. Similarly, analyses regarding software availability are limited by the extent of the manual data enhancement (see Sections 4.8.1 and 6.3.2). As 50% of overall mentions are covered the statements have some representative power, while otherwise no analyses in this regard would be possible. However, it is possible that the distribution differs in the long tail of software mention. For instance, a higher ratio of source code published in the scope of research with few re-uses might be present in the long tail.

The performed analysis of retraction notices is based on a low sample number and has only exemplary character. The current analyses use only retraction notices contained in SoftwareKG, i.e., PMC OA, as it allows the systematic identification of retraction notices and the examination of full-text documents. Further notices were not included because there is currently no corpus of full-text document retraction notices available, however, further exploration of this topic might lead to further insights on the influence of software on scientific results.

An analysis of the publication types contained in SoftwareKG showed that aside the largest groups of *Research Articles* and *Review Articles* it also covers a broad range of other publication types such as *Case Reports*, *Letters*, or *Editorials*. In the scope of this work only *Research Articles* and *Review Articles* were considered for the large-scale analysis, while *Retractions* are covered in the outlined case study. Therefore, it is important to note that the results presented here only represent this specific subset of publications contained in PMC OA, while further analyses are required to investigate software in the remaining types. The corresponding information is included in SoftwareKG, allowing future work to easily extend the analyses (see Chapter 7).

A limitation specific to the performed case study is that the processed documents were based on PDF publications, which might contain text extraction artifacts. This is expected to lead to a higher number of errors in the IE pipeline since it was not trained to handle such artifacts. Therefore, potential errors were initially analyzed during information enrichment, revealing some errors which are discussed above. Overall, the results appear to be valid compared to the initially presented analysis, particularly, comparing the number of articles using software with 72% on

[9] The package counts for PyPi were retrieved on 7 February 2024.

PMC OA compared to 58.1% in the control article set, and the average of mentioned software with 3.93 on PMC OA and 3.32 in the control article set. While the number on software mentions differs, extraction artifacts have not led to a strong overestimation of software mentions, with the difference being most likely caused by a different article distribution, e.g., regarding the underlying research domain that is known to influence the number of mentioned software (see Section 2.1.3). Spelling variations specific to *SPSS* are further similar with 80.3% and 79.8% of all *SPSS*-related mentions using the name "SPSS", in PMC OA and the control set, respectively. Furthermore, it is argued that due to the design of the case study errors are equally distributed between the set of retracted and control articles, leading to valid results for the comparison. However, SoftwareKG is better suited to make general statements about software mentions in scientific publications than the dataset from the case study.

6.5 Discussion: The Role of Software in Science

Software Mentions

With an average of 3.93 software mentions per article, the observed result confirms previous studies, ranging from 3.2 [118] to 5.5 [78] mentions in different subsets of PMC. The results also are in the range of manually annotated data with 4.8 in *PLoS_m* and 2.85 in *PubMed_f* (see Section 3.3.1), while *PLoS_M* only considers methods section and the real number might be even higher, and the considered sample size in *PubMed_f* is too low to allow a reliable comparison. Moreover, the identified top ten software align well between the manual results and the large-scale analyses with only the pair of *SPM* and *BLAST* differing. With over 564 k, the number of different software from over 10.9 M mentions is high, given that 2.5 M articles were included in the analysis. As discussed above, it is assumed that this number overestimates the number of distinct software due to errors in NER and ED, particularly, regarding the long tail of rarely mentioned software.

The distribution of software mentions per software shows that only few software are used in a large number of articles and thus play a major role in science. This distribution partly confirms the general trend of previously reported statements [205, 78] but shows even higher skewness. As expected (see Section 3.3.1), it also shows a higher skewness as found in the analysis on manually annotated data. This amplified trend could be the result of software ED which was not applied in previous studies and highlights the importance of considering all spelling variations for software mention analysis. Regarding software availability, it was found that the overall ratio of used open source software is low with only 39.3%, particularly regarding its advantages for reproducibility, software inspection, and validation, discussed in Chapter 1.

The most frequent software (seven from the top ten) are mainly used for statistical data analysis, and, overall, an increased role of software that can be controlled via scripts rather than point and click software can be observed. A closer look at the domain specific distribution of the top ten software revealed domain specific differences as it characterizes all of the analysed domains uniquely. The specific software toolchain containing *SHELXL*, *SHELXS*, *SAINT*, and *SHELXTL*, for instance, takes a spot among the most frequently used software in Physics, Materials, and Chemistry, while none of the tools is among the most frequent software in any other considered discipline.

The results further showed that software is mentioned differently, with some software being mentioned more often in the context of a single publication than others. Among the top ten software, for instance, *R*, *BLAST*, and *MATLAB* have a particularly high rate of mentions per articles. Multiple mentions of the software could mean that the software is used for multiple purposes. The function range of the software *R*, for instance, can be extended by the use of different *PlugIns* making it applicable in a broad range of settings. However, it could also indicate that authors provide more details about how a specific software was applied in the scope of their research. This might, for instance, be the case when software was applied in a complex setting, and it is necessary to closer describe specifics of how it was applied, e.g., regarding its parameter setup.

Article Level Statistics

The importance of software increased in recent years for both, the actual investigation as well as the reporting within the scholarly publication. This is suggested by the increasing trend of including software mentions into the textual description and the increasing number of different software per article. A reason could be the growing complexity of data driven analyses requiring more software to be employed coupled with a high awareness about transparency and reproducibility in general. The positive correlation between journal rank and number of different software supports this by suggesting strong rigor in the description of the analysis[10]. The observed domain specific difference in software usage could reflect the role of data in those domains. While in Arts and Humanities and Economics only few articles mention only few software suggesting an important role of manual data analysis, in Mathematics, Computer Science, and Decision Science many articles mention multiple different software indicating automatic and complex data collection and analyses.

The analyses identified a relation between the number of software used in an article and the amount of employed free and open source software. One explanation for this relation could be that work applying multiple software is building custom data processing pipelines where individual software is employed for highly specific purposes, for which no commercial tools exist. On the other hand, the application of multiple commercial software can also be a question of cost and institutional availability, while such problems do not need to be considered for free and open source software. Furthermore, researchers combining multiple different software might also have more expertise in software usage, better knowledge of existing tool support, and a better understanding of the benefits of open source software. A small trend towards the use of more free and open source software was also identified in recent years and a small trend can also be identified for higher ranked journals. It is argued that this reflects an increasing awareness for the benefits of open source software in the scientific community. Differences are also found between domains, where a higher number of open source software mentions is found in technical disciplines with particularly high effects identified for Computer Science and Engineering where a high level of software expertise can be assumed. However, it also depends on the availability of specific tool support, further discussed below.

It was found that trends regarding software availability can be notably influenced by specific software tools. In the analyses regarding software availability a discrepancy between free and open source software was identified in the years from 2008 to 2011. A similar effect was observed in research domains regarding Materials Science, Physics, and Chemistry, and in journal rank regarding the lowest quantiles. All were caused by the same toolchain around *SHELX*, which is most dominantly used in these scopes. This highlights the limitation of analyses regarding software availability, described in Section 6.4, and the high impact of the toolchain.

Software Citation Completeness

In general, citation completeness is at a low level, with only 39.5% of articles mentioning versions and 28.5% developers, while 24.8% employ formal citations. These numbers were found to be at a comparable level with the results from the manual analysis (46.6% version, 30.7% developer, 31% formal citation). The values were further summarized to express whether sufficient metadata was provided to allow software identification and developer attribution (51.3%), as well as codebase

[10] When interpreting the journal rank as an indicator of journal quality and thus for review quality

identification (25.4%). While the overall citation completeness is low, it was found that it also did not improve over recent years, with formal citations stagnating, developer mentions decreasing, but version mentions slightly improving. This suggests a lack of awareness about the necessity for proper software citation and its impact on reproducibility.

Differences in software citations were identified between different domains, with technical domains using formal citations more frequently in contrast to research domains related to Medicine. This could be the case because the latter more frequently employ other materials and devices and adapt the same citation style for all research objects other than scholarly publications, while scientists from the technical domains might be software developers themselves, and better understanding the need for scientific attribution of software development described in Section 1.1. With respect to journal rank, a growing trend exists towards the usage of formal software citation with rising rank and a contrary trend in the completeness of in-text software mention for developer and version. This could support the statement of increasing rigor in scholarly review and the request for more traceable descriptions in higher quality journals.

Overall, commercial software with closed source code receives a higher number of developer and version mentions, following the instrument-like citation style that has previously been described by Howison and Bullard [118]. The values for version and developer mentions decrease for software available for free with closed source code, and decrease even further when considering open source software. An inverse trend exists for formal citation, with open source software being most frequently formally cited. This might be the case because free and open source software used in scientific investigations is often built in the scope of research and, therefore, is likely to have a corresponding scientific publication, with the results in Section 3.3.2 showing that software articles are the most common target of formal software citation. Overall, these findings further highlight the lack of awareness for proper attribution, and suggest that authors cite software based on the availability of citable resources, or based on the citation recommendation provided with software, which has been discussed in Section 3.5.

Types of Software Mention

Each software mention is classified according to mention type indicating the reason why the software was mentioned within the scholarly article and software type providing information about the particular kind of software. Analyses of disciplinary differences showed that most software is created and published by scientists from the technical domains (Mathematics, Engineering, and Computer Science). This reflects the domains with the highest interest in automating complex calculations combined with the programming knowledge to implement new suited software. Further, software *Allusions* without actual *Usage* are also more common within scholarly publications from those disciplines indicating more description, discussion, and comparison between software entities. On the other hand, there are disciplines which mostly reuse existing scientific software such as Material Science, Nursing, Dentistry, and Veterinary.

When looking at the different kinds of software, an increasing trend towards the use of *PlugIns* was identified over recent years, see Figure A3. The relative frequency of *Application* mentions, in contrast, declined. This can be seen as an indicator towards the usage and extension of established software frameworks. With respect to the host-software, a notable overlap in the most frequently used software (Table 6.2) and the most important host-software (Table 6.4) was found. This includes the *Programming Environments (PEs) R* and *Matlab*, but also the *Applications ImageJ*, *Stata*, and *Excel*. More than 22 *k* *PlugIns* were found for *R* making it the most important host-software for scientific investigations. While this number seem high at first glance, inspecting the two most important package repositories, CRAN and Bioconductor with 18,312 and 2042 unique packages[11] indicates these results to be plausible. However, the comparatively low FScores for the identification of *PlugIns* might have resulted in an overestimation of less frequent *PlugIns*. This growing interest in *R* and its package universe was previously investigated [166, 165], some results of which are confirmed here. In particular, an overlap of the most frequent *R* packages.

By analyzing the mention types *Creation* and *Deposition*, the most important targets for the publication of software could be identified. On the one hand, this includes web services such as Github for general purpose software and CRAN for *R* packages, on the other hand, software journals. Specifically designed repositories to host and assign DOIs to scientific research data such as Zenodo were not identified as a common source for publishing scientific software with a share of <1% of depositions. It has to be noted that two results for the most frequent deposition URLs were the result of a false software mention detection and its propagation.

Software and Article Retraction

The relation between software and article retraction, an essential mechanism for implementing scientific self-correction [5], was investigated in a specific case study. While the previously discussed results have shown that software is an integral part of modern science and thus influences certain aspects of the data analysis and results reported in scholarly articles, the performed analysis found evidence that software was involved in the retraction of multiple articles and revealed differences in the scientific software landscape and software citation habits in retracted articles.

The analysis shows that software has been explicitly stated in retraction notices and has been the cause for the retraction of articles, with a majority of errors resulting from usage errors and unintended software behavior of which authors were unaware. While these cases only contribute to a small number of overall retractions, they highlight the strong influence of software on science as a single action of the software caused most reported errors. It was further found that authors themselves catch most errors, highlighting the challenges for reproducibility but also suggesting that there might be other unnoticed cases. A potential solution would be to include analysis scripts as part of the scholarly publication, for instance, inter-weaved as one literate data analysis document, as peer review can reveal such errors.

Retracted articles mention less software as well as a less diverse set of software, as observed by the increased usage of the most frequent software. Moreover, retracted articles more frequently employ commercial and closed source software in favor of free and open source software. Citation completeness is also lacking in retracted articles for free and open source software, since they are more likely to mention free and open source software without any information such as version or developer and, in turn, are less likely to provide formal citations. Completeness for commercial and proprietary software, on the other hand, is almost equal to the control set.

A notable outlier from other retracted articles are articles automatically generated by *PaperMill* techniques, which systematically differ from other articles and exaggerate software mentions. The generation mechanisms have learned that software is typically present in scientific publications, which is why almost every *PaperMill* publication contains software. Furthermore, they have learned to mimic proper citation for commonly applied, well-established tools. This is supported by the findings that complete information of proprietary software is provided at a very high rate, but information of free and open source software at a particularly low rate. Looking at specific software, it becomes clear that *PaperMills* have learned that *SPSS* is a commonly used tool as they include it in over 70% of publications.

[11] Package counts for CRAN and Bioconductor were retrieved on 4 October 2021.

7 | Conclusion and Future Work

In the scope of this work, I developed a method to systematically analyze software usage in scientific publications at a large scale. To achieve this goal, I initially developed the high-quality ground-truth dataset SoMeSci based on manual annotation. It covers comprehensive information on software mentions in scientific publications, including metadata, context information and software identity. Based on this resource and preliminary analyses performed on it, I further developed an automatic IE pipeline for the corresponding information. The pipeline was then applied on a large-scale dataset of 3.2 M scientific articles, and the corresponding information was formally modeled to create SoftwareKG—a large-scale research KG of software mentions in scientific literature. I then used this resource, which I made available as a FAIR publication [308], to perform a large-scale analysis of software mentions in scientific publications which provides insights on software impact, its citations, availability, and publication, considering trends along different dimensions such as time or research discipline.

SoMeSci

Initially, a manual data annotation was performed, with the two main goals of assessing the requirements for large-scale analyses of software in scientific publications, and establishing a comprehensive ground-truth dataset for the problem. The resulting dataset SoMeSci contains 3718 software mentions of 884 unique software entities in 1367 publications. It was annotated with a high data quality ensured by assessing the IAA at every annotation step and by using an annotation strategy that leads to an iterative correction of errors during annotation. Furthermore, a customized sampling strategy was used to increase the number of contained software mentions to reduce annotation effort, and to allow the inclusion of the rare—but highly relevant—mention context of software development and its publication as research artifacts.

Aside mentions of the software itself SoMeSci covers a context classification of software mentions, a broad range of metadata, and software identities, linking mentions to real-world software entities. The developed manual annotation process provided first insights into the complexity of software mention extraction and showed that some aspects are challenging for human annotators. Especially, the annotation of contextual information and the software identity were found to be challenging problems for annotators. The main limitation of the dataset is its restriction to articles from PMC OA introducing a bias towards Life Sciences and open-access publications. Since SoMeSci is the basis for developing all subsequent methods the bias also concerns all other described steps.

Information Extraction (IE) pipeline

The automatic IE pipeline, developed to extract software mentions from scientific publications, achieves a high performance, particularly compared with prior approaches to the problem (see

Section 4.7). Therefore, the performance is also considered sufficient to perform reliable large-scale analyses of software in scientific publications. However, the individual steps of NER, RE, and ED, were all found to have individual limitations, outlined below, and it is necessary to considers that errors are propagated throughout the IE pipeline. Overall, the results of the IE step further illustrate the complexity of software mention detection in scientific literature, which was already shown by the initial manual annotation effort.

NER: The initial step of NER was performed with state of the art ML approaches for sequence tagging, which were compared on the problem. The results showed that specialized versions of BERT, adapted for scientific publications, outperformed Bi-LSTM-CRFs, so that SciBERT was selected to perform the extraction. It achieved an FScore of F=86.6% for the problem, while a further generalization test towards PMC OA articles found a performance of F=77.9%. The recognition for metadata is also at a high level with F=86.2% and F=82% in the two respective test settings. Rarely provided metadata, such as extensions and alternative names, were found to be more challenging to extract, and due to the low sample size their performance evaluation is also less reliable. Mention and software type classification by the used SciBERT model are also at high levels with values of F=73.3% and F=78.4%, considering that error propagation from software recognition is taken into account in their evaluation. However, particularly challenging extraction targets were identified with software type *PlugIn* and mention type *Allusion*.

RE: The developed RE approach was based on a manually engineered set of rules in combination with a Random Forest classifier and achieves a high performance of F=95.4%. Here, a less complex classifier was chosen because the problem is considerably easier than NER, since all extracted metadata is required to have an anchor entity, which, in a majority of cases, is the software mention itself. The only exception are optional relations between two software, such as the relation between *PlugIn* and host-software that was extracted with a lower performance of F=77.6%.

ED: The problem of software ED proved to be a challenging problem, with an initial approach showing that disambiguation by EL is not possible, because software used in science is insufficiently covered in existing databases. Therefore, two different approaches were developed for the problem, a manually engineered rule-based approach, and an ED method adapted from prior work of NLP ED that generates tuple distances based on an ML model as described in Section 2.3.2. Both were tested in two separate settings to evaluate their performance on the given dataset and their generalization performance with larger data size and noisy data. Both methods were found to work well on the problem with the ML model showing better performance of F=96.4% and F=93.7% in both employed test setting, respectively.

Overall, the developed ED provides reasonable results; 440 different spelling were identified for 539 k mentions of the software *SPSS* across all of PMC OA[1]. For the different spellings "python" and "Python" (see Table 6.4), however, no common cluster could be determined. While this clearly represents an error and should be identified based on string similarity, the ML based distance additionally considers the context and accompanying entities such as developer and URL. The reason here could be the low number of spelling variations that prevent the semantic drift to counteract misleading matching probabilities. In consequence, this would mean that more frequent software (with many different contexts and spelling variations) are more likely to be disambiguated than less frequent software (with fewer contexts and spelling variations), which may have had a

[1] All 440 were manually verified. Alternative Names are not counted in difference to Table 6.2, since the number of times the names are mentioned are counted below.

reinforcing effect on the power law distribution of mentioned software. For software with more frequent spelling and context variations, in contrast, this might result in more false positives and thus overestimate the software use. For *Excel*, 54 spelling variations were found that represent 152 k mentions across all of PMC OA. From those only about 150 k mentions (from 13 different spellings) can be considered as correctly classified. The remaining mentions refer to software such as Firefox (1162, .7%) or F (185, .1%). A similar phenomenon could be observed for Stata, while ED errors were also found in the dataset created to analyze software retractions.

In general, my work showed that the ED problem shifts with larger sample numbers, making it challenging to find suited decision boundaries for large-scale datasets based on gold standard training data alone, as the samples get denser in the features space. In the given work this was taken into account in a special evaluation setting adding automatically extracted software mentions from 100 k articles to increase the data size, and to also include noisy samples which further increases the difficulty of the problem. However, the clustering performance is still only based on the set of 3718 mentions corresponding to 884 distinct software. Therefore, a central challenge for future work will be to establish a suited benchmark and evaluation setting for software ED. On the one hand, this benchmark does have to consider the shifting objective, described above. On the other hand, it has to extend the set of considered software. This has to be performed by manual selection and tuning of the dataset, for instance, samples of different software with the same name should be systematically added. Further, both software well represented in databases and rare cases of long-tail software should be included, while it is also important to consider mentions with different amounts of provided metadata. The corresponding evaluation of the benchmark, should also consider the weighting of individual software. Due to the distribution of software, matching of common software tools receives a high weight, which is the case in the evaluation performed here. Errors in the long tail of software, however, have only small impact on the overall score. Future evaluation settings should account for this problem by assigning a higher weight to correct matching in software samples with smaller sample number to allow a more general performance estimation.

While the outlined ED approach could be applied for the given large-scale analysis, it has a run-time complexity of $\mathcal{O}(n^2)$ and is not able to scale far beyond the given dataset. In the given setting, the algorithm had to be optimized for run-time and it was necessary to impose restrictions on the data. Specifically, ED is only performed between software with different names, even so Section 3.1.3 showed that different software with the same name exists, even when only considering SoMeSci. Therefore, future work is required to consider extending NLP ED approaches, which have only been applied on benchmark data and do not consider scalability for large-scale application. Suited methods for this problem can be adapted from the problem of ER in large-scale databases, where research is focused on reducing run-time requirements for ED by using methods such as blocking by Autoencoders or Language Models, as described in Section 2.3. Moreover, future work could also extend ED to not only consider software but also its metadata. This could enable the distinction between Person and Organization for software developers that is assumed by the SDO [89].

Regarding the discussion on ED it should also be mentioned that there are efforts that aim to solve the problem by introducing persistent identifiers for software, introduced in Section 2.1.4. With proper implementation and compliance by the scientific community such effort could alleviate the problem of ED in future publications. The corresponding databases, however, require crowd sourcing to achieve a suited coverage of software, particularly by software developers to timely add new software to make it citable in subsequent work. Furthermore, such approaches are associated with a high curation effort to solve issues such as entry deduplication and to ensure high data quality. During data annotation for SoMeSci the usage of persistent identifiers for software was observed to be extremely rare, with only one identifier and one archive link provided in the entire dataset, both within formal citations.

Future work: Beside the research directions on software ED, described above, future work should focus on extending available datasets for the problem. More data is primarily required to allow a generalization to different research disciplines, as both prior work and this work have shown that software mentions differs strongly between domains. The work of Du et al. [74], for instance, shows that the amount of software mentions strongly varies between Life Sciences and Economics. Moreover, generalization might also be challenging due to other domain specific aspects. Disciplines such as Computer Science, for instance, might have a different culture regarding code sharing and re-use, because researcher are better educated in software development and more likely to re-use and adapt existing source code and scripts. An extension of the dataset is also required, because data is still sparse for specific aspects regarding software citation such as licenses or specific software types, leading to challenges in training and evaluation (see above). To improve this problem, future work could initially make use of the results established in the scope of a CZI Hackathon (see Section 3.3) to merge Softcite v2 and SoMeSci, which would lead to a notable increase in sample size, before systematically extending the dataset towards other disciplines such as Computer Science.

SoftwareKG

Based on the established IE pipeline, SoftwareKG was built—a large-scale research KG representing information on 11.8 M software mentions, in 3.2 M scientific publications, published in 11 k journals, covering 27 top-level scientific disciplines, and a total of more than 300 M RDF triples. SoftwareKG was populated using all publications in PMC OA published up to January 2021. It is the largest KG of software in scientific research and has upon its publication represented the largest dataset regarding software mentions in scientific publications[2]. It is based on a data model that I developed to represent software mentions in scientific articles, distinguishing between software on mention and on entity level. The data model was developed re-using existing vocabularies and the KG was made publicly available to the scientific community as a FAIR publication [308] to allow easy reuse and facilitate future research on software in science. Additionally, the developed data model was designed to allow an easy integration of further data on software mentions identified in the scope of future work. How SoftwareKG and the resources established in this work can be applied in future work has also been demonstrated in a case study on the relation between software and article retraction described in Section 6.3, which provided valuable insights on the impact of software on science.

Future work might plan to extend SoftwareKG, for instance, with information on additional research domains or by adding recently published articles, since the current version only includes publications up to January 2021. Therefore, it is necessary to consider an update strategy for SoftwareKG. As described above, the developed data model does already consider an extension of the data contained in SoftwareKG by different data sources, with differing document types, and varying extraction quality. When performing updates to the knowledge base, it is essential to consider that ED and entity level inference need to be recalculated. This makes updates expensive in terms of run-time and computational resources. It is, thus, suggested to extend the dataset in suited intervals maybe on a half-yearly basis. Note that this is a recommendation for future work, and not part this work.

[2] Recently, Istrate et al. [128] published a corpus that considers a larger article collection, but does not provide the data as Linked Open Data.

The Role of Software in Science

The importance of software in science has been growing in recent years, with both its absolute and relative number increasing over the course of the last ten years. The high impact of software on scientific investigations is further highlighted by the performed case study on article retraction, which revealed multiple instances where software has been the primary reason for article retraction. Most software mentioned in scholarly articles are software for statistical analysis such as *SPSS*, *R*, and *Prism*, which are applied across a broad range of scientific disciplines, while my work also identified many domain specific peculiarities in software usage. Both, the amount of software usage as well as the most used software and their application purpose, varies significantly between domains, while the top ten software allow a unique characterization of domains.

Interestingly, my work identified a trend towards the use of extendable software architectures instead of stand-alone software, which allow the problem specific extension of general software frameworks. The most important representative of these frameworks is *R* by a notable margin, followed by *MATLAB*, and *Python*. Overall, their designs allow an easy extension and offers high flexibility. Especially adding functionality and publishing new packages or *PlugIns* is facilitated. This trend could also be shown by analyzing which infrastructure is used by scientists to publish their software. Here, Github plays a central role as a repository for publication and re-use of research software, as previously assumed [236], but CRAN and Bioconductor are especially important in combination with *R*.

Software Citation Completeness: While the importance of software increased in recent years, as described above, there was no improvement in software citation completeness, which is at an overall insufficient level with respect to all considered metadata. However, a positive trend exists regarding version mention, which is consistently improving. The completeness of developer mentions, on the other hand, has seen a decrease rather than an improvement over recent years. This leads me to believe that there is still a lack in awareness for software citation in science, even so software citation guidelines have been available and promoted since 2016, e.g., by Smith et al. [263].

The most recent citation guidelines for software [135] recommend citing software formally by a reference referring to the software directly, to enable a unique identification. Prior work has already shown that formal citation is uncommon, which was further corroborated by the findings of this work, with only 24.8% of mentioned software being formally cited. A more detailed analyses revealed that the relative amount of provided formal citation did not change throughout the time from 2009 to 2020, and was at its maximum in 2006. Regarding journal ranks, however, a tendency towards more formal citations in higher ranked journals was identified, while and opposing trends exists for in-text citation completeness, which is found to be lower for higher ranked journals. A similar observation can be made for domains, where opposing trends exists between domains related to Medicine, preferring in-text citation, while technical domains tend to more formal citations. I believe that both finding can be attributed to a better awareness for proper software attribution.

Moreover, my results show that free and open source software is more likely to receive formal citations, which has already been reported in the work of Howison and Bullard [118]. I believe one reason is that authors choose to attribute software resulting from research work. On the other hand, open source software published as research output might also be easier citable than commercial software because they often have corresponding software articles and citation recommendations made by their developers. That citation requests by authors influence citation practices for software has been previously reported [75].

The manual analyses further explored the peculiarities of formal software citation. These analyses show that the use of formal citation often leads to a reduced software identifiability, and, thus, lower reproducibility. The main issue is that software is most commonly cited by software articles, which are unsuited to cite software, as they do not account for the central issue of software versioning. This problem was inherently caused since software was seen a weaker contribution than articles [101], but still exists in the examined data. The issue is further exacerbated since the results also showed that significantly fewer metadata on software versions is provided when software is cited by software articles. Overall, the findings suggest a lack of awareness regrading proper software citation and particularly the importance of software codebase identification.

Moreover, an analysis of the representation quality for formal software citations—referring to software directly—by providers of bibliographic data revealed strong oversights. The results showed that metadata on software is often misrespresented or even omitted by providers of bibliometric data, and that systematic analyses of software in scientific publications are not possible based on the current scientometric system. For one, publishers lack a specialized format for citations to scientific software, which was never encountered throughout the entire data annotation, while all analyzed bibliographic databases are currently unable to adequately represent software citation. To represent the intricacies of software citation and to enable and facilitate systematic representation and analyses of formal software citation it would be necessary to model at least two different views to software: (1) provide all information about the particular software as given by the authors in a structured manner to enable reproducibility, and (2) link citations of the same software—including different versions—to a common element to enable software tracking.

Overall, several stakeholders need to adjust their practices to improve the state of formal software citation:

- Software developers need to update their citation requests to conform with software citation guidelines, so they do not impose a conflict between citation formats on authors;
- Publishers need to update their citation representation to allow a representation of software citations and to include the corresponding metadata;
- Bibliometric databases need to update their citation handling to not only consider publications, but different forms of research artifacts and their intricacies.

While action is primarily needed from the stakeholders above, future work can support their efforts by establishing an automatic classification of formal software citation and the contained information, corresponding to the classification of informal software citation performed in this work. This would provide the necessary tools to implement proper software representation, while the corresponding results could also be used for large-scale analyses to further validate the manual results found in the scope of this work.

Open Source Software: The results also provide an overview of the current state of free and open source software in scientific research, with a central finding being that with increasing number of software mentions the amount of employed free and open source rises. I believe that this results from highly specialized requirements of the employed analyses in terms of software usage, with no commercial tool support existing for these applications or that commercial tools lack the required flexibility, extendability, and validation to be applied in such settings. Regarding the amount of open source software, my work showed that proprietary software accounts for the majority of mentioned software, while a positive trend to open source software was identified over recent years, which I see as a positive development as it leads to higher reproducibility and re-use due to missing vendor lock-in. A higher frequency of open source software is also present in higher ranked journals with the highest ventile using more than 50% open source software. This could illustrate that these publications conduct more complex analyses and have a better understanding for the benefits of open source software.

It is necessary to note that the analyses performed on software availability are based on manual annotation, and only include the most common software with a coverage of 50% of overall software mentions. Therefore, only the 428 most popular software are included in the analysis. The limitations of this approach are demonstrated by the software toolchain around *SHELX*, which due to its high popularity from 2008 to 2011 in specific journals related to the domains of Materials Science, Physics, and Chemistry, notably impacted trends along these dimensions. Furthermore, I believe that the distribution of open source software does differ in the long-tail of software, and argue that it does contain more open source software, which is highly specialized for research problems and hence only rarely re-used. Further work is, therefore, required to fully asses the state of open source software in science. A potential approach is to include external information in the classification, for instance, by developing an EL of long-tail software against common publication platforms, such as Github, Bitbucket, or CRAN, as identified in Table 6.5, and automatically gather license information. This could be based on existing work of Lopez et al. [173] and Istrate et al. [128], which includes methods of software EL.

Software in Retracted Articles: An analysis of the relation between software and article retraction found valuable insights, but also demonstrates how the techniques presented in my work can be applied to new analysis and different data. It was found that the software landscape in retracted articles is less diverse and that less free and open source software is employed. Moreover, my results show that citation practices for free and open source software in retracted articles are negligent with a substantially lower amount of formal citations. Moreover, the work showed that software has been the primary reason for article retractions, as described above.

However, some of analyses performed on the relationship between software and article retraction have to be considered as preliminary due to the low sample size they are based on. Future work could extend these analyses by including a larger number of retraction notices and more retracted articles. Furthermore, different publication types could be systematically analyzed as they might reveal further ways in which software impacts science. The outlined analyses only consider software in *Research* and *Review Articles*, and in *Retraction Notices*, while further cases might exist where software analysis is of interest, e.g., *Corrections* or *Reports*. In the scope of retraction notices, for instance, it was often found that software was used and explicitly mentioned for automatic detection of plagiarism. These findings were not presented in this work, as they do not reflect software usage by researchers, but they demonstrate the potential of SoftwareKG for future investigations.

KG of Software in Science

In the scope of this work, SoftwareKG was used to perform large-scale analyses of software in science. However, in future work it can build the basis to develop software related features required by the scientific community, for instance, a mapping for available software and newly established software, or to track software usage and establish software impact measures. This requires knowledge enrichment, e.g., by linking SoftwareKG with other knowledge bases, which makes the structure of the published data particularly useful. Software developers, for instance, could be linked to general purpose KGs such as Wikidata or to software repositories such as CRAN, while SoftwareKG could be linked to OpenALEX (see Section 2.2.1) to add further information on authors, affiliations, and more specific fields of research.

Currently, information on software entities in SoftwareKG is based on aggregation over articles and subject to error propagation from the IE pipeline. In general, it is desirable to make this information as reliable and complete as possible. SoftwareKG is well suited to build the basis for such an knowledge base due to its publication format, and because it allows to identify software with missing information. Therefore, future work should focus on improving the ED step as discussed above, while EL to external resources can further be used to extend and correct information on software entities. For this purpose, information can be aggregated from multiple sources, e.g., from general purpose KGs for common software tools, from initiatives tracking software such as SwMATH or SciCrunch (see Section 2.1), and from package repositories such as CRAN (as described above).

The identified software *Creation* and *Deposition* in SoftwareKG are particularly useful to establish a mapping of the scientific software landscape as it allows to track new software publications with low latency. Furthermore, these results allow to provide researchers with recommendations on where to publish their software and the corresponding description, by identifying the most common publishing platforms and journals for research software. Overall this can, in future work, also build the bases to extend SoftwareKG to implement a search and recommendation for software.

Lastly, future work could also extend the pipeline I developed in this work to different scientific artifacts such as data. For this purpose, the pipeline serves as a model of how knowledge from scientific articles can be systematically extracted and modeled. It provides insights on all individual steps, from structuring data annotation to scalability considerations for ED. In this context, my work also serves as a demonstration of major caveats that need to be taken into account when developing similar approaches, and allows an estimation of the required effort. Large-scale information on other research artifacts would enable further analyses and might provide unexpected insights into scientific research when explored in combination with SoftwareKG.

Software

I utilized multiple different software throughout my work. ML and data processing was mainly performed in Python version 3.11.3 [289], using several additional packages. PyTorch version 1.13.0 [208, 207] was used for implementation of deep learning models, and the Huggingface transformers version 4.6.1 [311, 310] specifically for BERT models. Scikit-learn version 0.24.1 [211, 210] was further used for Random Forest models and to implement evaluation functions, while source code from seqeval version 1.2.2 [191] was adapted for evaluation of sequence tagging. Moreover, lxml version 4.6.3 [26] and NLTK version 3.6.2 [172, 171] were used for extraction and pre-processing of articles documents, and Gensim version 4.0.1 [228, 227] to handle word embeddings for Bi-LSTM-CRFs. Lastly, rdflib version 6.3.2 [225] was used for KG construction. Data analysis was performed with R version 4.3.0 [222] including several packages. Tidyverse version 1.3.1 [304, 303] was used for data analyses, magrittr version 2.0.2 [19] for data wrangling. Furthermore, ggplot2 version 3.3.5 [302, 301], patchwork version 1.1.1 [209], ggalluvial [40], and easyalluvial [146] were used for visualization. Statistical analyses were implemented with DescTools [257], rcompanion [180], and effectsize version 0.8.3 [29, 28], and SPARQL 1.16 [286] was used to query data from SoftwareKG. Moreover, xtable [61] was used to automatically generate tabular outputs, Quarto [7] was used to generate documents for literate data analysis, and RStudio [219] was used to support all analyses performed in R. Additionally, BRAT [271, 272] was used to implement several data annotations.

Availability

Source code and data developed in the scope of my work has been made publicly available for all publications contributing to this work, as well as the extension of SoftwareKG described in this work. Source code for pre-processing article documents were published in the Python package *articlenizer* [247] at Github (https://github.com/dave-s477/articlenizer). The source code implementing Bi-LSTM-CRFs and Transformer based models, Relation Extraction, and Entity linking was published in the Python package *SoMeNLP* [241], that was iteratively extended throughout this work and published at Github (https://github.com/dave-s477/SoMeNLP). Analysis scripts for software citation in SoftwareKG were made available at Github (https://github.com/f-krueger/SoftwareKG-PMC-Analysis), and the literature data analyses scripts for the software in retracted articles were published at https://github.com/dave-s477/Software-and-Article-Retraction. Moreover, the analysis scripts for citation representation quality by providers of bibliographic data were published at https://github.com/dave-s477/SoMeSci_Citation. SoMeSci—the gold standard for software mentions—is made available to be directly used as a ground-truth dataset for ML at https://github.com/dave-s477/SoMeSci, and is further represented as a KG to facilitate further analyses and exploration with a corresponding website and SPARQL endpoint at https://data.gesis.org/somesci. The data itself was also made available at Zenodo to ensure long time archival [243], and corresponding source code is published at https://github.com/dave-s477/SoMeSci_Code. The extension of formal citation annotation published with the original SoMeSci data, while analyses scripts for formal citation are available at https://github.com/dave-s477/SoMeSci_Citation. Lastly, the different versions of SoftwareKG were made publicly available. The first version was published at Zenodo [252] and has a corresponding website at https://data.gesis.org/softwarekg/softwarekg-social/, while the second version (covering PMC) is also available at Zenodo [242] and has a corresponding website at https://data.gesis.org/softwarekg/softwarekg-pubmed/. The last version of SoftwareKG, described here and based on an updated data model, is available from Zenodo at https://doi.org/10.5281/zenodo.10951778 [240] under CC-BY license to guarantee long time availability and archival of the data. Overall, the publication can be considered as 5 star open data as defined by Berners-Lee [31]. However, the published version of SoftwareKG only contains information available under open licenses. As this is not the case for bibliometric data gathered from Scimago through PubMedKG, those parts where excluded from publication. However, Scimago identifiers were added so the information can be gathered in future investigations working with the data.

Abbreviations

CDCR Cross-Document Coreference Resolution

CEM Coarsened Exact Matching

CI Confidence Interval

CNN Convolutional Neural Network

CRAN The Comprehensive R Archive Network

CRF Conditional Random Fields

CZI Chan Zuckerberg Initiative

DOI Digital Object Identifier

ED Entity Disambiguation

EL Entity Linking

ER Entity Resolution

IAA Inter Annotator Agreement

IE Information Extraction

IRI Internationalized Resource Identifier

JATS Journal Article Tag Suite

JOSS Journal of Open Source Software

KG Knowledge Graph

LLM Large-Scale Language Model

LSH Locality Sensitive Hashing

LSTM Long Short Term Memory Network

MAG Microsoft Academic Graph

ML Machine Learning

NER Named Entity Recognition

NLP Natural Language Processing

NN Neural Network

OR odds-ratio

OS Operating System

PE Programming Environment

PLoS Public Library of Science

PMC PubMed Central

PMC OA PubMed Central Open Access Subset

pp percentage points

PyPi Python Package Index

RDF Resource Description Framework

RE Relation Extraction

RW Retraction Watch

S2AG Semantic Scholar Academic Graph

SDO Software Description Ontology

SGD Stochastic Gradient Descent

SHACL Shapes Constraint Language

SJR Scimago Journal Rank

SWO Software Ontology

Bibliography

[1] Mandhri Abeysooriya, Megan Soria, Mary Sravya Kasu, and Mark Ziemann. “Gene name errors: Lessons not learned”. In: *PLOS Computational Biology* 17.7 (2021), e1008984. DOI: 10.1371/journal.pcbi.1008984.

[2] Serge Abiteboul. “Querying semi-structured data”. In: *Database Theory — ICDT ’97*. 1997, pp. 1–18. DOI: 10.1007/3-540-62222-5_33.

[3] Jean-François Abramatic, Roberto Di Cosmo, and Stefano Zacchiroli. “Building the universal archive of source code”. In: *Communications of the ACM* 61.10 (2018), pp. 29–31. DOI: 10.1145/3183558.

[4] Marcel R Ackermann. *7 million publications [Blog]*. Online; accessed 25 January 2024. 2024. URL: https://blog.dblp.org/2024/01/01/7-million-publications/.

[5] Isola Ajiferuke and Janet O. Adekannbi. “Correction and retraction practices in library and information science journals”. In: *Journal of Librarianship and Information Science* 52.1 (2018), pp. 169–183. DOI: 10.1177/0961000618785408.

[6] Boanerges Aleman-Meza, Farshad Hakimpour, I. Budak Arpinar, and Amit P. Sheth. “SwetoDblp ontology of Computer Science publications”. In: *Journal of Web Semantics* 5.3 (2007), pp. 151–155. ISSN: 1570-8268. DOI: https://doi.org/10.1016/j.websem.2007.03.001.

[7] J.J. Allaire, Charles Teague, Carlos Scheidegger, Yihui Xie, and Christophe Dervieux. *Quarto*. [Software] version 1.2.475. 2023. DOI: 10.5281/zenodo.5960048. URL: https://github.com/quarto-dev/quarto-cli.

[8] Alice Allen, Peter J. Teuben, and P. Wesley Ryan. “Schroedinger’s Code: A Preliminary Study on Research Source Code Availability and Link Persistence in Astrophysics”. In: *The Astrophysical Journal Supplement Series* 236.1 (2018), p. 10. DOI: 10.3847/1538-4365/aab764.

[9] Pierre Alliez, Roberto Di Cosmo, Benjamin Guedj, Alain Girault, Mohand-Said Hacid, Arnaud Legrand, and Nicolas Rougier. “Attributing and Referencing (Research) Software: Best Practices and Outlook From Inria”. In: *Computing in Science & Engineering* 22.1 (2020), pp. 39–52. DOI: 10.1109/mcse.2019.2949413.

[10] Fatma Dilara Altunbas, Baris Onen Unsalver, and Alisan Burak Yasar. “Aspects of cognitive performance relating to Theory of Mind (ToM) among people diagnosed with Post-Traumatic Stress Disorder (PTSD) [Retraction]”. In: *Neuropsychiatr Dis Treat* 15 (2019), p. 2415. DOI: 10.2147/NDT.S227512.

[11] Simone Angioni, Angelo Salatino, Francesco Osborne, Diego Reforgiato Recupero, and Enrico Motta. “AIDA: A knowledge graph about research dynamics in academia and industry”. In: *Quantitative Science Studies* 2.4 (2021), pp. 1356–1398. DOI: 10.1162/qss_a_00162.

[12] Renzo Angles and Claudio Gutierrez. “Survey of graph database models”. In: *ACM Computing Surveys (CSUR)* 40.1 (2008), pp. 1–39. DOI: 10.1145/1322432.1322433.

[13] Renzo Angles and Claudio Gutierrez. “The expressive power of SPARQL”. In: *International Semantic Web Conference*. Karlsruhe, Germany, 2008, pp. 114–129. DOI: 10.1007/978-3-540-88564-1_8.

[14] Ron Artstein and Massimo Poesio. “Inter-Coder Agreement for Computational Linguistics”. In: *Computational Linguistics* 34.4 (2008), pp. 555–596. DOI: 10.1162/coli.07-034-R2.

[15] Muhammad Ahtisham Aslam and Naif Radi Aljohani. “SPedia: a central hub for the linked open data of scientific publications”. In: *International Journal on Semantic Web and Information Systems (IJSWIS)* 13.1 (2017), pp. 128–147. DOI: 10.4018/IJSWIS.2017010108.

[16] Sören Auer, Allard Oelen, Muhammad Haris, Markus Stocker, Jennifer D’Souza, Kheir Eddine Farfar, Lars Vogt, Manuel Prinz, Vitalis Wiens, and Mohamad Yaser Jaradeh. “Improving access to scientific literature with knowledge graphs”. In: *Bibliothek Forschung und Praxis* 44.3 (2020), pp. 516–529. DOI: 10.1515/bfp-2020-2042.

[17] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. “Layer normalization”. In: *arXiv preprint arXiv:1607.06450* (2016). DOI: 10.48550/arXiv.1607.06450.

[18] Jeroen Baas, Michiel Schotten, Andrew Plume, Grégoire Côté, and Reza Karimi. “Scopus as a curated, high-quality bibliometric data source for academic research in quantitative science studies”. In: *Quantitative Science Studies* 1.1 (2020), pp. 377–386. DOI: 10.1162/qss_a_00019.

[19] Stefan Milton Bache and Hadley Wickham. *magrittr: A Forward-Pipe Operator for R*. [R Package] version 2.0.3. 2022. URL: https://CRAN.R-project.org/package=magrittr.

[20] Andrea Bagnacani, Paolo Ciancarini, Angelo Di Iorio, Andrea Giovanni Nuzzolese, Silvio Peroni, and Fabio Vitali. “The semantic lancet project: a linked open dataset for scholarly publishing”. In: *Knowledge Engineering and Knowledge Management: EKAW 2014*. Linköping, Sweden, 2015, pp. 101–105. DOI: 10.1007/978-3-319-17966-7_10.

[21] Dzmitry Bahdanau, K. Cho, and Y. Bengio. “Neural Machine Translation by Jointly Learning to Align and Translate”. In: *arXiv preprint arXiv:1409.0473* (2015). DOI: 10.48550/arXiv.1409.0473.

[22] Anita Bandrowski, Esteban Gonzalez, Tom Honeyman, James Howison, Anne L’Hôte, Arcangelo Massari, and David Schindler. *softMeScite*. 2023. URL: https://github.com/annelhote/softMeScite.

[23] Shany Barhom, Vered Shwartz, Alon Eirew, Michael Bugert, Nils Reimers, and Ido Dagan. “Revisiting Joint Modeling of Cross-document Entity and Event Coreference Resolution”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 2019, pp. 4179–4189. DOI: 10.18653/v1/P19-1409.

[24] Aricia Bassinet, Laetitia Bracco, Anne L’Hôte, Eric Jeangirard, Patrice Lopez, and Laurent Romary. “Large-scale Machine-Learning analysis of scientific PDF for monitoring the production and the openness of research data and software in France”. In: (2023). URL: https://hal.science/hal-04121339.

[25] Carlo Batini, Cinzia Cappiello, Chiara Francalanci, and Andrea Maurino. “Methodologies for data quality assessment and improvement”. In: *ACM Computing Surveys* 41.3 (2009), pp. 1–52. DOI: 10.1145/1541880.1541883.

[26] Stefan Behnel, Martijn Faassen, Ian Bicking, Holger Joukl, Simon Sapin, Marc-Antoine Parent, et al. *lxml*. [Python Package] version 4.6.3. 2010. URL: https://github.com/lxml/lxml.

[27] Iz Beltagy, Kyle Lo, and Arman Cohan. "SciBERT: A Pretrained Language Model for Scientific Text". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China, 2019, pp. 3615–3620. DOI: 10.18653/v1/D19-1371.

[28] Mattan S. Ben-Shachar, Daniel Lüdecke, Dominique Makowski, et al. *effectsize*. [R Package] version 0.8.3. 2023. URL: https://github.com/easystats/effectsize.

[29] Mattan S. Ben-Shachar, Daniel Lüdecke, and Dominique Makowski. "effectsize: Estimation of Effect Size Indices and Standardized Parameters". In: *Journal of Open Source Software* 5.56 (2020), p. 2815. DOI: 10.21105/joss.02815.

[30] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166. DOI: 10.1109/72.279181.

[31] Tim Berners-Lee. *Is your linked open data 5 star?* 2010. URL: http://www.w3.org/DesignIssues/LinkedData#fivestar.

[32] Gilbert Bigras, Wei-Feng Dong, Sarah Canil, Judith Hugh, Richard Berendt, George Wood, and Hua Yang. "New robust and reproducible stereological IHC Ki67 breast cancer proliferative assessment to replace traditional biased labeling index". In: *Applied Immunohistochemistry & Molecular Morphology* 25.10 (2017), p. 687. DOI: 10.1097/PAI.0000000000000371.

[33] Kathrin Blagec, Adriano Barbosa-Silva, Simon Ott, and Matthias Samwald. "A curated, ontology-based, large-scale knowledge graph of artificial intelligence tasks and benchmarks". In: *Scientific Data* 9.1 (2022), p. 322. DOI: 10.1038/s41597-022-01435-x.

[34] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. "Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge". In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. SIGMOD '08. Vancouver, Canada, 2008, pp. 1247–1250. DOI: 10.1145/1376616.1376746.

[35] Piero Andrea Bonatti, Stefan Decker, Axel Polleres, and Valentina Presutti. "Knowledge graphs: New directions for knowledge representation on the semantic web (dagstuhl seminar 18371)". In: *Dagstuhl reports*. Vol. 8. 9. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2019. DOI: 10.4320/DagRep.8.9.29.

[36] Boyle, Connor. *Scikit-Learn's F-1 calculator is broken [Blog]*. https://connorboyle.io/2023/12/17/sklearn-f1-bug.html. Online; accessed 25 January 2024. 2023.

[37] Leo Breiman. "Bagging predictors". In: *Machine learning* 24.2 (1996), pp. 123–140. DOI: 10.1007/BF00058655.

[38] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32. DOI: 10.1023/A:1010933404324.

[39] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, and Clemens Winter et al. "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901.

[40] Jason Cory Brunson and D. Quentin. *ggalluvial: Alluvial Plots in 'ggplot2'*. [R Package] version 0.12.5. 2023. URL: http://corybrunson.github.io/ggalluvial/.

[41] Mateusz Buda, Atsuto Maki, and Maciej A Mazurowski. "A systematic study of the class imbalance problem in convolutional neural networks". In: *Neural networks* 106 (2018), pp. 249–259. DOI: https://doi.org/10.1016/j.neunet.2018.07.011.

[42] Davide Buscaldi, Danilo Dessì, Enrico Motta, Francesco Osborne, and Diego Reforgiato Recupero. "Mining scholarly publications for scientific knowledge graph construction". In: *The Semantic Web: ESWC 2019 Satellite Events*. Portorož, Slovenia, 2019, pp. 8–12. DOI: 10.1007/978-3-030-32327-1_2.

[43] Avi Caciularu, Arman Cohan, Iz Beltagy, Matthew E Peters, Arie Cattan, and Ido Dagan. "CDLM: Cross-Document Language Modeling". In: *Findings of the Association for Computational Linguistics: EMNLP 2021*. 2021, pp. 2648–2662. DOI: 10.18653/v1/2021.findings-emnlp.225.

[44] Arie Cattan, Alon Eirew, Gabriel Stanovsky, Mandar Joshi, and Ido Dagan. "Cross-document Coreference Resolution over Predicted Mentions". In: *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. 2021, pp. 5100–5107. DOI: 10.18653/v1/2021.findings-acl.453.

[45] Arie Cattan, Sophie Johnson, Daniel Weld, Ido Dagan, Iz Beltagy, Doug Downey, and Tom Hope. "Scico: Hierarchical cross-document coreference for scientific concepts". In: *arXiv preprint arXiv:2104.08809* (2021). DOI: 10.48550/arXiv.2104.08809.

[46] Chan Zuckerberg Initiative Science. *Mapping the Impact of Software in Science [Blog]*. Online; accessed 19 January 2024. 2023. URL: https://www.aboutamazon.com/news/innovation-at-amazon/making-search-easier.

[47] Chang, Spencer. *Scaling Knowledge Access and Retrieval at Airbnb [Blog]*. Online; accessed 8 January 2024. 2018. URL: https://medium.com/airbnb-engineering/scaling-knowledge-access-and-retrieval-at-airbnb-665b6ba21e95.

[48] Yani Chen, Juan Du, Yu Wang, Haiyan Shi, Qiuyu Jiang, Yangfeng Wang, Huahua Zhang, Yameng Wei, Wanjuan Xue, Zhiying Pu, et al. "MicroRNA-497-5p induces cell cycle arrest of cervical cancer cells in S phase by targeting CBX4". In: *OncoTargets and therapy* 12 (2019), p. 10535. DOI: 10.2147/OTT.S210059.

[49] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. "GradNorm: Gradient normalization for adaptive loss balancing in deep multitask networks". In: 2018. DOI: 10.48550/arXiv.1711.02257.

[50] Jason PC Chiu and Eric Nichols. "Named entity recognition with bidirectional LSTM-CNNs". In: *Transactions of the association for computational linguistics* 4 (2016), pp. 357–370. DOI: 10.1162/tacl_a_00104.

[51] Haeran Cho and Yi Yu. "Link prediction for interdisciplinary collaboration via co-authorship network". In: *Social Network Analysis and Mining* 8 (2018), pp. 1–12. DOI: 10.1007/s13278-018-0501-6.

[52] Prafulla Kumar Choubey and Ruihong Huang. "Event Coreference Resolution by Iteratively Unfolding Inter-dependencies among Events". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark, 2017, pp. 2124–2133. DOI: 10.18653/v1/D17-1226.

[53] Hagen Chrapary, Wolfgang Dalitz, Winfried Neun, and Wolfram Sperber. "Design, Concepts, and State of the Art of the swMATH Service". In: *Mathematics in Computer Science* 11.3 (2017), pp. 469–481. DOI: 10.1007/s11786-017-0305-5.

[54] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. "An overview of end-to-end entity resolution for big data". In: *ACM Computing Surveys (CSUR)* 53.6 (2020), pp. 1–42. DOI: 10.1145/3418896.

[55] Jacob Cohen. "A Coefficient of Agreement for Nominal Scales". In: *Educational and Psychological Measurement* 20.1 (1960), pp. 37–46. DOI: 10.1177/001316446002000104.

[56] M. Cokol, I. Iossifov, R. Rodriguez-Esteban, and A. Rzhetsky. "How many scientific papers should be retracted?" In: *EMBO Rep* 8.5 (2007), pp. 422–423. DOI: 10.1038/sj.embor.7400970.

[57] Roberto Di Cosmo, Morane Gruenpeter, and Stefano Zacchiroli. "Referencing Source Code Artifacts: A Separate Concern in Software Citation". In: *Computing in Science & Engineering* 22.2 (2020), pp. 33–43. DOI: 10.1109/mcse.2019.2963148.

[58] Agata Cybulska and Piek Vossen. "Translating Granularity of Event Slots into Features for Event Coreference Resolution." In: *Proceedings of the The 3rd Workshop on EVENTS: Definition, Detection, Coreference, and Representation*. Denver, Colorado, 2015, pp. 1–10. DOI: 10.3115/v1/W15-0801.

[59] Agata Cybulska and Piek Vossen. "Using a sledgehammer to crack a nut? Lexical diversity and event coreference resolution." In: *LREC*. 2014, pp. 4545–4552.

[60] Bruce D'Arcus and Frédérick Giasson. *Bibliographic Ontology Specification Revision: 1.3*. https://bibliontology.com/. [Ontology Specification for BIBO]. 2009.

[61] David B. Dahl, David Scott, Charles Roosen, Arni Magnusson, and Jonathan Swinton. *xtable: Export Tables to LaTeX or HTML*. [R Package] version 1.8-4. 2019. URL: https://CRAN.R-project.org/package=xtable.

[62] Jean Daunizeau, Vincent Adam, and Lionel Rigoux. "VBA: a probabilistic treatment of nonlinear models for neurobiological and behavioural data". In: *PLoS computational biology* 10.1 (2014), e1003441. DOI: 10.1371/journal.pcbi.1003441.

[63] DCMI Usage Board. *DCMI Metadata Terms*. http://dublincore.org/specifications/dublin-core/dcmi-terms/2020-01-20/. [Ontology Specification for DCT]. 2020.

[64] Antonio Del Sol and Pablo Carbonell. "The modular organization of domain structures: insights into protein–protein binding". In: *PLoS computational biology* 3.12 (2007), e239. DOI: 10.1371/journal.pcbi.0030239.

[65] Danilo Dessí, Francesco Osborne, Diego Reforgiato Recupero, Davide Buscaldi, and Enrico Motta. "CS-KG: A Large-Scale Knowledge Graph of Research Entities and Claims in Computer Science". In: *The Semantic Web – ISWC 2022*. 2022, pp. 678–696. DOI: 10.1007/978-3-031-19433-7_39.

[66] Danilo Dessí, Francesco Osborne, Diego Reforgiato Recupero, Davide Buscaldi, Enrico Motta, and Harald Sack. "AI-KG: an automatically generated knowledge graph of artificial intelligence". In: *The Semantic Web–ISWC 2020: 19th International Semantic Web Conference*. Athens, Greece, 2020, pp. 127–143. DOI: 10.1007/978-3-030-62466-8_9.

[67] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota, 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423.

[68] Roberto Di Cosmo, Morane Gruenpeter, Bruno Marmol, Alain Monteil, Laurent Romary, and Jozefina Sadowska. "Curated archiving of research software artifacts: lessons learned from the French open archive (HAL)". In: *IDCC 2020-International Digital Curation Conference*. 2020. DOI: 10.2218/ijdc.v15i1.698.

[69] Roberto Di Cosmo and Stefano Zacchiroli. "Software heritage: Why and how to preserve software source code". In: *iPRES 2017-14th International Conference on Digital Preservation*. 2017, pp. 1–10.

[70] Angelo Di Iorio, Andrea Giovanni Nuzzolese, Silvio Peroni, David M Shotton, and Fabio Vitali. "Describing bibliographic references in RDF." In: *SePublica*. 2014.

[71] Michelle L Dion, Jane Lawrence Sumner, and Sara McLaughlin Mitchell. "Gendered citation patterns across political science and social science methodology fields". In: *Political analysis* 26.3 (2018), pp. 312–327. DOI: 10.1017/pan.2018.12.

[72] Mark Dredze, Paul McNamee, Delip Rao, Adam Gerber, Tim Finin, et al. "Entity disambiguation for knowledge base population". In: *Proceedings of the 23rd International Conference on Computational Linguistics*. 2010.

[73] Stephan Druskat. "Software and Dependencies in Research Citation Graphs". In: *Computing in Science & Engineering* 22.2 (2020), pp. 8–21. DOI: 10.1109/mcse.2019.2952840.

[74] Caifan Du, Johanna Cohoon, Patrice Lopez, and James Howison. "Softcite dataset: A dataset of software mentions in biomedical and economic research publications". In: *Journal of the Association for Information Science and Technology* 72.7 (2021), pp. 870–884. DOI: 10.1002/asi.24454.

[75] Caifan Du, Johanna Cohoon, Patrice Lopez, and James Howison. "Understanding progress in software citation: a study of software citation in the CORD-19 corpus". In: *PeerJ Computer Science* 8 (2022), e1022. DOI: 10.7717/peerj-cs.1022.

[76] Geraint Duck, Aleksandar Kovacevic, David L Robertson, Robert Stevens, and Goran Nenadic. "Ambiguity and variability of database and software names in bioinformatics". In: *Journal of biomedical semantics* 6 (2015), pp. 1–11. DOI: 10.1186/s13326-015-0026-0.

[77] Geraint Duck, Goran Nenadic, Andy Brass, David L Robertson, and Robert Stevens. "bioNerDS: exploring bioinformatics' database and software use through literature mining". In: *BMC bioinformatics* 14.1 (2013), p. 194. DOI: 10.1186/1471-2105-14-194.

[78] Geraint Duck, Goran Nenadic, Michele Filannino, Andy Brass, David L. Robertson, and Robert Stevens. "A Survey of Bioinformatics Database and Software Usage through Mining the Literature". In: *PLOS ONE* 11.6 (2016), pp. 1–25. DOI: 10.1371/journal.pone.0157989.

[79] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. "Distributed representations of tuples for entity resolution". In: *Proc. VLDB Endow.* 11.11 (2018), pp. 1454–1467. DOI: 10.14778/3236187.3236198.

[80] Anders Eklund, Thomas E Nichols, and Hans Knutsson. “Cluster failure: why fMRI inferences for spatial extent have inflated false-positive rates”. In: *Proceedings of the National Academy of Sciences* (2016), p. 201602413. DOI: 10.1073/pnas.1602413113.

[81] Emily Escamilla, Lamia Salsabil, Martin Klein, Jian Wu, Michele C Weigle, and Michael L Nelson. “It’s Not Just GitHub: Identifying Data and Software Sources Included in Publications”. In: *International Conference on Theory and Practice of Digital Libraries*. 2023, pp. 195–206. DOI: 10.1007/978-3-031-43849-3_17.

[82] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. “A density-based algorithm for discovering clusters in large spatial databases with noise”. In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD’96. Portland, Oregon, 1996, pp. 226–231.

[83] Ferric C Fang, R Grant Steen, and Arturo Casadevall. “Misconduct accounts for the majority of retracted scientific publications”. In: *Proceedings of the National Academy of Sciences* 109.42 (2012), pp. 17028–17033. DOI: 10.1073/pnas.1212247109.

[84] Michael Färber. “The microsoft academic knowledge graph: A linked data source with 8 billion triples of scholarly data”. In: *The Semantic Web–ISWC 2019: 18th International Semantic Web Conference*. Auckland, New Zealand, 2019, pp. 113–129. DOI: 10.1007/978-3-030-30796-7_8.

[85] David Filip, Shaun McCance, Dave Lewis, Christian Lieske, Arle Lommel, Jirka Kosek, Felix Sasaki, and Yves Savourel. “Internationalization Tag Set (ITS) Version 2.0”. In: *W3C recommendation* (2013). URL: http://www.w3.org/TR/2013/REC-its20-20131029/.

[86] Karën Fort, Maud Ehrmann, and Adeline Nazarenko. “Towards a Methodology for Named Entities Annotation”. In: *Proceedings of the Third Linguistic Annotation Workshop*. ACL-IJCNLP ’09. Suntec, Singapore, 2009, pp. 142–145.

[87] Mirko Gabelica, Ružica Bojčić, and Livia Puljak. “Many researchers were not compliant with their published data sharing statement: a mixed-methods study”. In: *Journal of Clinical Epidemiology* 150 (2022), pp. 33–41. DOI: 10.1016/j.jclinepi.2022.05.019.

[88] Lahiru Gangoda, Shivakumar Keerthikumar, Pamali Fonseka, Laura E Edgington, Ching-Seng Ang, Cemil Ozcitti, Matthew Bogyo, Belinda S Parker, and Suresh Mathivanan. “Inhibition of cathepsin proteases attenuates migration and sensitizes aggressive N-Myc amplified human neuroblastoma cells to doxorubicin”. In: *Oncotarget* 6.13 (2015), p. 11175. DOI: 10.18632/oncotarget.3579.

[89] Daniel Garijo, Varun Ratnakar, Yolanda Gil, Deborah Khider, and Maximiliano Osorio. *The Software Description Ontology. Revision: 1.4.0*. https://w3id.org/okn/o/sd/1.4.0. [Ontology Specification for SDO]. 2019.

[90] Andrew Gelman and Jennifer Hill. *Data analysis using regression and multilevel/hierarchical models*. Cambridge university press, 2006.

[91] Yolanda Gil, Varun Ratnakar, and Daniel Garijo. “OntoSoft: Capturing Scientific Software Metadata”. In: *Proceedings of the 8th International Conference on Knowledge Capture*. K-CAP 2015. Palisades, NY, USA, 2015. DOI: 10.1145/2815833.2816955.

[92] Joseph Glaz and Cristina P Sison. “Simultaneous confidence intervals for multinomial proportions”. In: *Journal of Statistical Planning and Inference* 82.1-2 (1999), pp. 251–262. DOI: 10.1016/S0378-3758(99)00047-6.

[93] Carole Goble. "Better Software, Better Research". In: *IEEE Internet Computing* 18.5 (2014), pp. 4–8. DOI: 10.1109/mic.2014.88.

[94] Teresa Gomez-Diaz and Tomas Recio. "On the evaluation of research software: the CDUR procedure". In: *F1000Research* 8 (2019). DOI: 10.12688/f1000research.19994.2.

[95] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[96] Gert-Martin Greuel and Wolfram Sperber. "swMATH–an information service for mathematical software". In: *International Congress on Mathematical Software*. 2014, pp. 691–701. DOI: 10.1007/978-3-662-44199-2_103.

[97] Paul Groth, Andrew Gibson, and Jan Velterop. "The anatomy of a nanopublication". In: *Information services & use* 30.1-2 (2010), pp. 51–56. DOI: 10.3233/ISU-2010-0613.

[98] Yu Gu, Robert Tinn, Hao Cheng, Michael Lucas, Naoto Usuyama, Xiaodong Liu, Tristan Naumann, Jianfeng Gao, and Hoifung Poon. "Domain-specific language model pretraining for biomedical natural language processing". In: *ACM Transactions on Computing for Healthcare (HEALTH)* 3.1 (2021), pp. 1–23. DOI: 10.1145/3458754.

[99] Ramanathan V Guha, Dan Brickley, and Steve Macbeth. "Schema.org: evolution of structured data on the web". In: *Communications of the ACM* 59.2 (2016), pp. 44–51. DOI: 10.1145/2844544.

[100] Ankit Gupta, Pruthvi Nagilla, Hai-Son Le, Coulton Bunney, Courtney Zych, Anbupalam Thalamuthu, Ziv Bar-Joseph, Sinnakaruppan Mathavan, and Velpandi Ayyavoo. "Retraction: Comparative expression profile of miRNA and mRNA in primary peripheral blood mononuclear cells infected with human immunodeficiency virus (HIV-1)". In: *PLoS One* 7.8 (2012). DOI: 10.1371/annotation/d28d38b2-41a3-42a6-b421-68f9460a676d.

[101] Lou Hafer and Arthur E Kirkpatrick. "Assessing open source software as a scholarly contribution". In: *Communications of the ACM* 52.12 (2009), pp. 126–129. DOI: 10.1145/1610252.1610285.

[102] B. G. Hall and S. J. Salipante. "Retraction: Measures of clade confidence do not correlate with accuracy of phylogenetic trees". In: *PLoS Comput Biol* 3.7 (2007), e158. DOI: 10.1371/journal.pcbi.0030158.

[103] Tony Hammond, Michele Pasin, and Evangelos Theodoridis. "Data integration and disintegration: Managing Springer Nature SciGraph with SHACL and OWL." In: *ISWC (Posters, Demos & Industry Tracks)*. 2017.

[104] Jo Erskine Hannay, Carolyn MacLeod, Janice Singer, Hans Petter Langtangen, Dietmar Pfahl, and Greg Wilson. "How do scientists develop and use scientific software?" In: *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*. IEEE, 2009. DOI: 10.1109/secse.2009.5069155.

[105] Wilhelm Hasselbring, Leslie Carr, Simon Hettrick, Heather Packer, and Thanassis Tiropanis. "Open source research software". In: *Computer* 53.8 (2020), pp. 84–88. DOI: 10.1109/MC.2020.2998235.

[106] Stefanie Haustein and Vincent Larivière. "The Use of Bibliometrics for Assessing Research: Possibilities, Limitations and Adverse Effects". In: *Incentives and Performance*. 2014, pp. 121–139. DOI: 10.1007/978-3-319-09785-5_8.

[107] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[108] Sebastian Hellmann, Jens Lehmann, Sören Auer, and Martin Brümmer. "Integrating NLP Using Linked Data". In: *The Semantic Web – ISWC 2013*. Berlin, Heidelberg, 2013, pp. 98–113. DOI: 10.1007/978-3-642-41338-4_7.

[109] Ginny Hendricks, Dominika Tkaczyk, Jennifer Lin, and Patricia Feeney. "Crossref: The sustainable source of community-owned scholarly metadata". In: *Quantitative Science Studies* 1.1 (2020), pp. 414–427. DOI: 10.1162/qss_a_00022.

[110] Simon Hettrick. *Research software sustainability: Report on a Knowledge Exchange workshop*. 2016.

[111] Geoffrey E Hinton and Ruslan R Salakhutdinov. "Reducing the dimensionality of data with neural networks". In: *Science* 313.5786 (2006), pp. 504–507. DOI: 10.1126/science.1127647.

[112] Tin Kam Ho. "Random decision forests". In: *Proceedings of 3rd International Conference on Document Analysis and Recognition*. Vol. 1. 1995, 278–282 vol.1. DOI: 10.1109/ICDAR.1995.598994.

[113] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[114] Johannes Hoffart, Fabian M Suchanek, Klaus Berberich, Edwin Lewis-Kelham, Gerard De Melo, and Gerhard Weikum. "YAGO2: exploring and querying world knowledge in time, space, context, and many languages". In: *Proceedings of the 20th international conference companion on World wide web*. 2011, pp. 229–232. DOI: 10.1145/1963192.1963296.

[115] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, et al. "Knowledge graphs". In: *ACM Computing Surveys (Csur)* 54.4 (2021), pp. 1–37. DOI: 10.1007/978-3-031-01918-0.

[116] Neil Chue Hong. "Why do we need to compare research software, and how should we do it". In: *Proceedings of the 4th Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE4. 1)*. Vol. 1686. 2016.

[117] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5 (1989), pp. 359–366. DOI: 10.1016/0893-6080(89)90020-8.

[118] James Howison and Julia Bullard. "Software in the scientific literature: Problems with seeing, finding, and using software mentioned in the biology literature". In: *Journal of the Association for Information Science and Technology* 67.9 (2016), pp. 2137–2155. DOI: 10.1002/asi.23538.

[119] James Howison, Ewa Deelman, Michael J McLennan, Rafael Ferreira da Silva, and James D Herbsleb. "Understanding the scientific software ecosystem and its impact: Current and future measures". In: *Research Evaluation* 24.4 (2015), pp. 454–470. DOI: 10.1093/reseval/rvv014.

[120] James Howison, Patrice Lopez, Caifan Du, and Hannah Cohoon. *Softcite Dataset Version 2 (2.0)*. [Dataset]. Zenodo, 2023. DOI: https://doi.org/10.5281/zenodo.7995565.

[121] George Hripcsak and Adam S. Rothschild. "Agreement, the F-Measure, and Reliability in Information Retrieval". In: *Journal of the American Medical Informatics Association* 12.3 (2005), pp. 296–298. DOI: 10.1197/jamia.M1733.

[122] Michael Hucka and Matthew J Graham. “Software search is not a science, even among scientists: A survey of how scientists and engineers find software”. In: *Journal of Systems and Software* 141 (2018), pp. 171–191. DOI: 10.1016/j.jss.2018.03.047.

[123] P. R. Hunter and A. Prüss-Ustün. “Retraction: Have We Substantially Underestimated the Impact of Improved Sanitation Coverage on Child Health? A Generalized Additive Model Panel Analysis of Global Data on Child Mortality and Malnutrition”. In: *PLoS One* 12.5 (2017), e0178903. DOI: 10.1371/journal.pone.0178903.

[124] Stefano M Iacus, Gary King, and Giuseppe Porro. “Causal inference without balance checking: Coarsened exact matching”. In: *Political analysis* 20.1 (2012), pp. 1–24. DOI: 10.1093/pan/mpr013.

[125] Piotr Indyk and Rajeev Motwani. “Approximate nearest neighbors: towards removing the curse of dimensionality”. In: *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 1998, pp. 604–613. DOI: 10.1145/276698.276876.

[126] Sergey Ioffe and Christian Szegedy. “Batch normalization: accelerating deep network training by reducing internal covariate shift”. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML’15. Lille, France, 2015, pp. 448–456.

[127] Ornella Irrera, Andrea Mannocci, Paolo Manghi, and Gianmaria Silvello. “Tracing Data Footprints: Formal and Informal Data Citations in the Scientific Literature”. In: *International Conference on Theory and Practice of Digital Libraries*. 2023, pp. 79–92. DOI: 10.1007/978-3-031-43849-3_7.

[128] Ana-Maria Istrate, Donghui Li, Dario Taraborelli, Michaela Torkar, Boris Veytsman, and Ivana Williams. “A large dataset of software mentions in the biomedical literature”. In: *Proceedings of ISSI 2023 — the 19th International Conference of the International Society for Scientometrics and Informetrics*. 2023, pp. 155–174. DOI: 10.5281/zenodo.8305981.

[129] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. “Population based training of neural networks”. In: *arXiv preprint arXiv:1711.09846* (2017). DOI: 10.48550/arXiv.1711.09846.

[130] Ashish Jaiswal, Ashwin Ramesh Babu, Mohammad Zaki Zadeh, Debapriya Banerjee, and Fillia Makedon. “A survey on contrastive self-supervised learning”. In: *Technologies* 9.1 (2020), p. 2. DOI: 10.3390/technologies9010002.

[131] Gangolf Jobb, Arndt Von Haeseler, and Korbinian Strimmer. “Retraction Note: TREEFINDER: a powerful graphical analysis environment for molecular phylogenetics”. In: *BMC Evol Biol* 15 (2015), p. 243. DOI: 10.1186/s12862-015-0513-z.

[132] Matthew B. Jones, Carl Boettiger, Abby Cabunoc Mayes, Arfon Smith, Peter Slaughter, Kyle Niemeyer, Yolanda Gil, Martin Fenner, Krzysztof Nowak, Mark Hahnel, Luke Coy, Alice Allen, Mercè Crosas, Ashley Sands, Neil Chue Hong, Daniel S. Katz, and Carole Goble. *CodeMeta: an exchange schema for software metadata. Version 2.0.* https://github.com/codemeta/codemeta. 2017. DOI: 10.5063/schema/codemeta-2.0.

[133] JoSS. *The Journal of Open Source Software [Portal]*. https://joss.theoj.org/about. Online; accessed 20 October 2022. n.d.

[134] Upulee Kanewala and James M Bieman. “Testing scientific software: A systematic literature review”. In: *Information and software technology* 56.10 (2014), pp. 1219–1232. DOI: 10.1016/j.infsof.2014.05.006.

[135] Daniel Katz, Neil Chue Hong, Tim Clark, August Muench, Shelley Stall, Daina Bouquin, Matthew Cannon, et al. "Recognizing the value of software: a software citation guide". In: *F1000Research* 9 (2021), p. 1257. DOI: 10.12688/f1000research.26932.2.

[136] Michael S Kavic and Richard M Satava. "Scientific literature and evaluation metrics: impact factor, usage metrics, and altmetrics". In: *JSLS: Journal of the Society of Laparoscopic & Robotic Surgeons* 25.3 (2021). DOI: 10.4293/JSLS.2021.00010.

[137] John D Kelleher, Brian Mac Namee, and Aoife D'arcy. *Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies*. MIT press, 2020.

[138] Diane Kelly. "An analysis of process characteristics for developing scientific software". In: *Journal of Organizational and End User Computing (JOEUC)* 23.4 (2011), pp. 64–79. DOI: 10.4018/joeuc.2011100105.

[139] Alex Kendall, Yarin Gal, and Roberto Cipolla. "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7482–7491. DOI: 10.1109/CVPR.2018.00781.

[140] S. Khodami, J. V. McArthur, L. Blanco-Bercial, and P. M. Arbizu. "Retraction Note: Molecular Phylogeny and Revision of Copepod Orders (Crustacea: Copepoda)". In: *Sci Rep* 10.1 (2020), p. 17602. DOI: 10.1038/s41598-020-74404-2.

[141] Hyeong-Geug Kim, Jung-Hyo Cho, Sa-Ra Yoo, Jin-Seok Lee, Jong-Min Han, Nam-Hun Lee, Yo-Chan Ahn, and Chang-Gue Son. "Antifatigue Effects of Panax ginseng C.A. Meyer: A Randomised, Double-Blind, Placebo-Controlled Trial". In: *PLOS ONE* 8.4 (2013), pp. 1–8. DOI: 10.1371/journal.pone.0061271.

[142] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014). DOI: 10.48550/arXiv.1412.6980.

[143] Rodney Kinney, Chloe Anastasiades, Russell Authur, Iz Beltagy, Jonathan Bragg, Alexandra Buraczynski, Isabel Cachola, Stefan Candra, Yoganand Chandrasekhar, Arman Cohan, Miles Crawford, Doug Downey, Jason Dunkelberger, Oren Etzioni, Rob Evans, Sergey Feldman, Joseph Gorney, David Graham, Fangzhou Hu, and Regan Huff et al. *The Semantic Scholar Open Data Platform*. 2023. DOI: 10.48550/arXiv.2301.10140.

[144] Julian Klingbeil, Max-Lennart Brandt, Max Wawrzyniak, Anika Stockert, Hans R. Schneider, Petra Baum, Karl-Titus Hoffmann, and Dorothee Saur. "Retraction of: Association of Lesion Location and Depressive Symptoms Poststroke". In: *Stroke* 52.7 (2021). DOI: 10.1161/str.0000000000000380.

[145] M. C. Koch, J. Lermann, N. van de Roemer, S. K. Renner, S. Burghaus, J. Hackl, R. Dittrich, S. Kehl, P. G. Oppelt, T. Hildebrandt, C. C. Hack, U. G. Pöhls, S. P. Renner, and F. C. Thiel. "Retraction Note: Improving usability and pregnancy rates of a fertility monitor by an additional mobile application: results of a retrospective efficacy study of Daysy and DaysyView app". In: *Reprod Health* 16.1 (2019), p. 54. DOI: 10.1186/s12978-019-0728-3.

[146] Bjoern Koneswarakantha. *easyalluvial: Generate Alluvial Plots with a Single Line of Code*. [R Package] version 0.3.1. 2022. URL: https://CRAN.R-project.org/package=easyalluvial.

[147] D. A. F. Al-Koofee, J. M. Ismael, and S. M. H. Mubarak. "Retraction notice to 'Point mutation detection by economic HRM protocol primer design '[Biochem. Biophys. Rep. 18 (2019) 100628]". In: *Biochem Biophys Rep* 20 (2019), p. 100688. DOI: 10.1016/j.bbrep.2019.100688.

[148] Shriram Krishnamurthi and Jan Vitek. “The real software crisis: Repeatability as a core value”. In: *Communications of the ACM* 58.3 (2015), pp. 34–36. DOI: 10.1145/2658987.

[149] Arun Krishnan. *Making search easier: How Amazon’s Product Graph is helping customers find products more [Blog]*. https://www.aboutamazon.com/news/innovation-at-amazon/making-search-easier. Online; accessed 8 January 2024. 2018.

[150] Frank Krüger and David Schindler. “A Literature Review on Methods for the Extraction of Usage Statements of Software and Data”. In: *Computing in Science & Engineering* 22.1 (2020), pp. 26–38. DOI: 10.1109/MCSE.2019.2943847.

[151] Seth Kulick, Ann Bies, Mark Liberman, Mark Mandel, Ryan McDonald, Martha Palmer, Andrew Schein, Lyle Ungar, Scott Winters, and Pete White. “Integrated Annotation for Biomedical Information Extraction”. In: *HLT-NAACL 2004 Workshop: Linking Biological Literature, Ontologies and Databases*. Boston, Massachusetts, USA, 2004, pp. 61–68.

[152] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. “Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data”. In: *Proceedings of the Eighteenth International Conference on Machine Learning*. ICML ’01. San Francisco, CA, USA, 2001, pp. 282–289.

[153] Rachael Lammey. “CrossRef text and data mining services”. In: *Insights* 28.2 (2015).

[154] Lammey, Rachael. *Data and software citation deposit guide [Documentation]*. https://www.crossref.org/documentation/reference-linking/data-and-software-citation-deposit-guide/. Online; accessed 19 January 2024. 2022.

[155] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. “Neural Architectures for Named Entity Recognition”. In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California, 2016, pp. 260–270. DOI: 10.18653/v1/N16-1030.

[156] J Richard Landis and Gary G Koch. “The measurement of observer agreement for categorical data”. In: *biometrics* (1977), pp. 159–174.

[157] Timothy Lebo, Satya Sahoo, Deborah McGuinness, Khalid Belhajjame, James Cheney, David Corsar, Daniel Garijo, Stian Soiland-Reyes, Stephan Zednik, and Jun Zhao. “Prov-o: The prov ontology”. In: *W3C recommendation* 30 (2013).

[158] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *Nature* 521.7553 (2015), pp. 436–444. DOI: 10.1038/nature14539.

[159] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[160] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. “BioBERT: a pre-trained biomedical language representation model for biomedical text mining”. In: *Bioinformatics* 36.4 (2019), pp. 1234–1240. DOI: 10.1093/bioinformatics/btz682.

[161] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. “DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia”. In: *Semantic Web* 6 (2015), pp. 167–195. DOI: 10.3233/SW-140134.

[162] Loet Leydesdorff, Lutz Bornmann, and Caroline S Wagner. “The relative influences of government funding and international collaboration on citation impact”. In: *Journal of the Association for Information Science and Technology* 70.2 (2019), pp. 198–201. DOI: 10.1002/asi.24109.

[163] Chuan Li, Cheng Wang, Lingxue Meng, Jiewen Xing, Tianya Wang, Hua Yang, Yingyin Yao, Huiru Peng, Zhaorong Hu, Qixin Sun, et al. “Retraction: Ectopic Expression of a Maize Hybrid Down-Regulated Gene ZmARF25 Decreases Organ Size by Affecting Cellular Proliferation in Arabidopsis”. In: *PLoS One* 11.5 (2016), e0155904. DOI: 10.1371/journal.pone.0155904.

[164] Kai Li, Xia Lin, and Jane Greenberg. “Software citation, reuse and metadata considerations: An exploratory study examining LAMMPS”. In: *Proceedings of the Association for Information Science and Technology* 53.1 (2016), pp. 1–10. DOI: 10.1002/pra2.2016.14505301072.

[165] Kai Li and Erjia Yan. “Co-mention network of R packages: Scientific impact and clustering structure”. In: *Journal of Informetrics* 12.1 (2018), pp. 87–100. DOI: 10.1016/j.joi.2017.12.001.

[166] Kai Li, Erjia Yan, and Yuanyuan Feng. “How is R cited in research outputs? Structure, impacts, and citation standard”. In: *Journal of Informetrics* 11.4 (2017), pp. 989–1002. DOI: 10.1016/j.joi.2017.08.003.

[167] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. “Hyperband: A novel bandit-based approach to hyperparameter optimization”. In: *Journal of Machine Learning Research* 18.185 (2018), pp. 1–52.

[168] Ruey-Hsia Li and Geneva G Belford. “Instability of decision tree classification algorithms”. In: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2002, pp. 570–575. DOI: 10.1145/775047.775131.

[169] Wenhua Liu, Lan Zhang, Donglin Zheng, and Yijie Zhang. “Umbilical cord blood-based gene signatures related to prenatal major depressive disorder: Retraction”. In: *Medicine (Baltimore)* 99.8 (2020), e19445. DOI: 10.1097/MD.0000000000016373.

[170] Kyle Lo, Lucy Lu Wang, Mark Neumann, Rodney Kinney, and Daniel Weld. “S2ORC: The Semantic Scholar Open Research Corpus”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online, 2020, pp. 4969–4983. DOI: 10.18653/v1/2020.acl-main.447.

[171] Edward Loper, Steven Bird, et al. *NLTK*. [Python Package] version 3.6.2. 2002. URL: https://github.com/nltk/nltk.

[172] Edward Loper and Steven Bird. “NLTK: The Natural Language Toolkit”. In: *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*. ETMTNLP ’02. Philadelphia, Pennsylvania, 2002, pp. 63–70. DOI: 10.3115/1118108.1118117.

[173] Patrice Lopez, Caifan Du, Johanna Cohoon, Karthik Ram, and James Howison. “Mining Software Entities in Scientific Literature: Document-Level NER for an Extremely Imbalance and Large-Scale Task”. In: *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM ’21)*. Virtual Event, QLD, Australia, 2021. DOI: 10.1145/3459637.3481936.

[174] Ilya Loshchilov and Frank Hutter. “Decoupled weight decay regularization”. In: *arXiv preprint arXiv:1711.05101* (2017). DOI: 10.48550/arXiv.1711.05101.

[175] Yi Luan, Luheng He, Mari Ostendorf, and Hannaneh Hajishirzi. “Multi-Task Identification of Entities, Relations, and Coreference for Scientific Knowledge Graph Construction”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium, 2018, pp. 3219–3232. DOI: 10.18653/v1/D18-1360.

[176] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. “Effective Approaches to Attention-based Neural Machine Translation”. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. 2015, pp. 1412–1421. DOI: 10.18653/v1/D15-1166.

[177] Xuezhe Ma and Eduard Hovy. “End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2016, pp. 1064–1074. DOI: 10.18653/v1/P16-1101.

[178] James Malone, Andy Brown, Allyson L. Lister, Jon Ison, Duncan Hull, Helen Parkinson, and Robert Stevens. “The Software Ontology (SWO): a resource for reproducibility in biomedical data analysis, curation and digital preservation”. In: *Journal of Biomedical Semantics* 5.25 (2014). DOI: 10.1186/2041-1480-5-25.

[179] Michela Malvisi, Fiorentina Palazzo, Nicola Morandi, Barbara Lazzari, John L Williams, Giulio Pagnacco, and Giulietta Minozzi. “Responses of bovine innate immunity to Mycobacterium avium subsp. paratuberculosis infection revealed by changes in gene expression and levels of microRNA”. In: *PloS one* 11.10 (2016), e0164461. DOI: 10.1371/journal.pone.0164461.

[180] Salvatore S. Mangiafico. *rcompanion: Functions to Support Extension Education Program Evaluation*. [R Package] version 2.4.30. Rutgers Cooperative Extension. New Brunswick, New Jersey, 2023. URL: https://CRAN.R-project.org/package=rcompanion/.

[181] Richard P Mann, Andrea Perna, Daniel Strömbom, Roman Garnett, James E Herbert-Read, David JT Sumpter, and Ashley JW Ward. “Retraction: Multi-scale inference of interaction rules in animal groups using Bayesian model selection”. In: *PLoS computational biology* 8.8 (2012). DOI: 10.1371/annotation/7bc3a37e-db82-4813-8242-7d34877125c5.

[182] Aniko Maraz, Borbála Hende, Róbert Urbán, and Zsolt Demetrovics. “Pathological grooming: Evidence for a single factor behind trichotillomania, skin picking and nail biting”. In: *PLoS ONE* 12.9 (2017), e0183806. DOI: 10.1371/journal.pone.0183806.

[183] Adam Marcus. *‘In hindsight the mistake was quite stupid’: Authors retract paper on stroke*. https://retractionwatch.com/2021/07/13/in-hindsight-the-mistake-was-quite-stupid-authors-retract-paper-on-stroke/. Online; Accessed: 2022-08-16. 2021.

[184] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. “Distributed representations of words and phrases and their compositionality”. In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[185] Alistair Miles, Brian Matthews, Michael Wilson, and Dan Brickley. “SKOS Core: Simple Knowledge Organisation for the Web”. In: *Proceedings of the 2005 International Conference on Dublin Core and Metadata Applications: Vocabularies in Practice*. DCMI ’05. Madrid, Spain, 2005.

[186] Greg Miller. “A Scientist’s Nightmare: Software Problem Leads to Five Retractions”. In: *Science* 314.5807 (2006), pp. 1856–1857. DOI: 10.1126/science.314.5807.1856.

[187] Makoto Miwa and Mohit Bansal. "End-to-End Relation Extraction using LSTMs on Sequences and Tree Structures". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany, 2016, pp. 1105–1116. DOI: 10.18653/v1/P16-1105.

[188] Marius Mosbach, Maksym Andriushchenko, and Dietrich Klakow. "On the Stability of Fine-tuning BERT: Misconceptions, Explanations, and Strong Baselines". In: *arXiv preprint arXiv:2006.04884* (2020). DOI: 10.48550/arXiv.2006.04884.

[189] X. Mu, Y. Yang, Y. Liu, D. Luo, M. Xu, H. Wei, D. Gu, H. Song, and Y. Hu. "Retraction Note to: The complete mitochondrial genomes of two freshwater snails provide new protein-coding gene rearrangement models and phylogenetic implications". In: *Parasit Vectors* 10.1 (2017), p. 350. DOI: 10.1186/s13071-017-2287-1.

[190] Martijn J Mulder, Wouter Boekel, Roger Ratcliff, and Birte U Forstmann. "Corticosubthalamic connection predicts individual differences in value-driven choice bias". In: *Brain Structure and Function* 219 (2014), pp. 1239–1249. DOI: 10.1007/s00429-013-0561-3.

[191] Hiroki Nakayama. *seqeval: A Python framework for sequence labeling evaluation*. [Python Package] version 1.2.2. 2018. URL: https://github.com/chakki-works/seqeval.

[192] U. Nangia and D. S. Katz. "Understanding Software in Research: Initial Results from Examining Nature and a Call for Collaboration". In: *2017 IEEE 13th International Conference on e-Science (e-Science)*. 2017, pp. 486–487. DOI: 10.1109/eScience.2017.78.

[193] Udit Nangia and Daniel S Katz. "Track 1 paper: surveying the US National Postdoctoral Association regarding software use and training in research". In: *Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE 5.1)*. 2017. DOI: 10.5281/zenodo.814102.

[194] Francesco Napolitano, Xiaopeng Xu, and Xin Gao. "Impact of computational approaches in the fight against COVID-19: an AI guided review of 17 000 studies". In: *Briefings in Bioinformatics* 23.1 (2021). DOI: 10.1093/bib/bbab456.

[195] Engineering National Academies of Sciences and Medicine. *Open Source Software Policy Options for NASA Earth and Space Sciences*. The National Academies Press, 2018. DOI: 10.17226/25217.

[196] Roberto Navigli, Simone Conia, and Björn Ross. "Biases in Large Language Models: Origins, Inventory and Discussion". In: *ACM Journal of Data and Information Quality* (2023). DOI: 10.1145/3597307.

[197] H. Nguyen, P. Dayan, Z. Pujic, J. Cooper-White, and G. J. Goodhill. "Retraction: A mathematical model explains saturating axon guidance responses to molecular gradients". In: *Elife* 7 (2018). DOI: 10.7554/eLife.37048.

[198] Richard Van Noorden, Brendan Maher, and Regina Nuzzo. "The top 100 papers". In: *Nature* 514.7524 (2014), pp. 550–553. DOI: 10.1038/514550a.

[199] Curtis G Northcutt, Anish Athalye, and Jonas Mueller. "Pervasive label errors in test sets destabilize machine learning benchmarks". In: *arXiv preprint arXiv:2103.14749* (2021). DOI: 10.48550/arXiv.2103.14749.

[200] Natasha Noy, Yuqing Gao, Anshu Jain, Anant Narayanan, Alan Patterson, and Jamie Taylor. "Industry-scale Knowledge Graphs: Lessons and Challenges: Five diverse technology companies show how it's done". In: *Queue* 17.2 (2019), pp. 48–75. DOI: 10.1145/3331166.

[201] Andrea Giovanni Nuzzolese, Anna Lisa Gentile, Valentina Presutti, and Aldo Gangemi. "Conference Linked Data: The ScholarlyData Project". In: *The Semantic Web – ISWC 2016*. Ed. by Paul Groth, Elena Simperl, Alasdair Gray, Marta Sabou, Markus Krötzsch, Freddy Lecue, Fabian Flöck, and Yolanda Gil. Cham: Springer International Publishing, 2016, pp. 150–158. DOI: 10.1007/978-3-319-46547-0_16.

[202] Ivan Oransky. "Retractions are increasing, but not enough". In: *Nature* 608.7921 (2022), pp. 9–9. DOI: 10.1038/d41586-022-02071-6.

[203] Oleksandr Ostrenko, Pietro Incardona, Rajesh Ramaswamy, Lutz Brusch, and Ivo F Sbalzarini. "pSSAlib: The partial-propensity stochastic chemical network simulator". In: *PLOS Computational Biology* 13.12 (2017), e1005865. DOI: 10.1371/journal.pcbi.1005865.

[204] Xuelian Pan, Erjia Yan, Ming Cui, and Weina Hua. "Examining the usage, citation, and diffusion patterns of bibliometric mapping software: A comparative study of three tools". In: *Journal of informetrics* 12.2 (2018), pp. 481–493. DOI: 10.1016/j.joi.2018.03.005.

[205] Xuelian Pan, Erjia Yan, Qianqian Wang, and Weina Hua. "Assessing the impact of software on science: A bootstrapped learning of software entities in full-text papers". In: *Journal of Informetrics* 9.4 (2015), pp. 860–871. DOI: 10.1016/j.joi.2015.07.012.

[206] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: *International Conference on Machine Learning*. 2013, pp. 1310–1318.

[207] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. *PyTorch*. [Python Package] version 1.13.0. 2019. URL: https://github.com/pytorch/pytorch.

[208] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems* 32 (2019), pp. 8026–8037.

[209] Thomas Lin Pedersen. *patchwork: The Composer of Plots*. [R Package] version 1.1.2. 2020. URL: https://CRAN.R-project.org/package=patchwork.

[210] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, et al. *Scikit-learn*. [Python Package] version 0.24.1. 2010. URL: https://github.com/scikit-learn/scikit-learn.

[211] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[212] Alberto Pepe, Alyssa Goodman, August Muench, Merce Crosas, and Christopher Erdmann. "How do astronomers share data? Reliability and persistence of datasets linked in AAS publications and a qualitative study of data practices among US astronomers". In: *PloS one* 9.8 (2014), e104798. DOI: 10.1371/journal.pone.0104798.

[213] Leonardo Perez de Souza, Thomas Naake, Takayuki Tohge, and Alisdair R Fernie. "From chromatogram to analyte to metabolite. How to pick horses for courses from the massive web resources for mass spectral plant metabolomics". In: *GigaScience* 6.7 (2017), gix037. ISSN: 2047-217X. DOI: 10.1093/gigascience/gix037.

[214] Silvio Peroni, Paolo Ciancarini, Aldo Gangemi, Andrea Giovanni Nuzzolese, Francesco Poggi, and Valentina Presutti. "The practice of self-citations: a longitudinal study". In: *Scientometrics* 123.1 (2020), pp. 253–282. DOI: 10.1007/s11192-020-03397-6.

[215] Silvio Peroni, Alexander Dutton, Tanya Gray, and David Shotton. "Setting our bibliographic references free: towards open citation data". In: *Journal of Documentation* 71.2 (2015), pp. 253–277. DOI: 10.1108/JD-12-2013-0166.

[216] Silvio Peroni, David Shotton, and Fabio Vitali. "One year of the OpenCitations Corpus: Releasing RDF-based scholarly citation data into the Public Domain". In: *The Semantic Web–ISWC 2017: 16th International Semantic Web Conference*. Vienna, Austria, 2017, pp. 184–192. DOI: 10.1007/978-3-319-68204-4_19.

[217] David G Pina, Lana Barać, Ivan Buljan, Francisco Grimaldo, and Ana Marušić. "Effects of seniority, gender and geography on the bibliometric output and collaboration networks of European Research Council (ERC) grant recipients". In: *PLoS One* 14.2 (2019), e0212286. DOI: 10.1371/journal.pone.0212286.

[218] PLoS. *Submission Guidelines [Website]*. https://journals.plos.org/plosone/s/submission-guidelines. Online; accessed 10 October 2023. n.d.

[219] Posit team. *RStudio: Integrated Development Environment for R*. [Software] version 2023.3.1.446. Posit Software, PBC. Boston, MA, 2023. URL: http://www.posit.co/.

[220] Jason Priem, Heather Piwowar, and Richard Orr. *OpenAlex: A fully-open index of scholarly works, authors, venues, institutions, and concepts*. 2022. DOI: 10.48550/arXiv.2205.01833.

[221] Sampo Pyysalo, Filip Ginter, Hans Moen, Tapio Salakoski, and Sophia Ananiadou. "Distributional Semantics Resources for Biomedical Text Processing". In: *Proc. of LBM '13*. 2013.

[222] R Core Team. *R: A Language and Environment for Statistical Computing*. [Software] version 4.3.0. R Foundation for Statistical Computing. Vienna, Austria, 2023. URL: https://www.R-project.org/.

[223] Maria B Rabaglino, Maureen Keller-Wood, and Charles E Wood. "Transcriptomics of the late gestation ovine fetal brain: modeling the co-expression of immune marker genes". In: *BMC genomics* 15.1 (2014), pp. 1–9. DOI: 10.1186/1471-2164-15-1001.

[224] James Ravenscroft, Amanda Clare, Arie Cattan, Ido Dagan, and Maria Liakata. "CD2CR: Co-reference resolution across documents and domains". In: *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. 2021, pp. 270–280. DOI: 10.18653/v1/2021.eacl-main.21.

[225] RDFLib Team. *RDFLib*. [Python Package] version 6.3.2. 2002. URL: https://github.com/RDFLib/rdflib.

[226] Daniel E. Re, Jillian J. M. O'Connor, Patrick J. Bennett, and David R. Feinberg. "Preferences for Very Low and Very High Voice Pitch in Humans". In: *PLOS ONE* 7.3 (2012), pp. 1–8. DOI: 10.1371/journal.pone.0032719.

[227] Radim Řehůřek, Petr Sojka, et al. *Gensim*. [Python Package] version 4.0.1. 2010. URL: https://github.com/piskvorky/gensim.

[228] Radim Řehůřek and Petr Sojka. "Software Framework for Topic Modelling with Large Corpora". English. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta, 2010, pp. 45–50.

[229] Nils Reimers and Iryna Gurevych. “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China, 2019, pp. 3982–3992. DOI: 10.18653/v1/D19-1410.

[230] MD Ribeiro and Sonia MR Vasconcelos. “Retractions covered by Retraction Watch in the 2013–2015 period: prevalence for the most productive countries”. In: *Scientometrics* 114.2 (2018), pp. 719–734. DOI: 10.1007/s11192-017-2621-6.

[231] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *arXiv preprint arXiv:1609.04747* (2016). DOI: 10.48550/arXiv.1609.04747.

[232] Sebastian Ruder. “An overview of multi-task learning in deep neural networks”. In: *arXiv preprint arXiv:1706.05098* (2017).

[233] Sebastian Ruder. “Neural transfer learning for natural language processing”. PhD thesis. NUI Galway, 2019.

[234] Almudena Ruiz-Iniesta and Oscar Corcho. “A review of ontologies for describing scholarly and scientific documents.” In: *SePublica* (2014).

[235] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *Nature* 323.6088 (1986), p. 533. DOI: 10.1038/323533a0.

[236] Pamela H. Russell, Rachel L. Johnson, Shreyas Ananthan, Benjamin Harnke, and Nichole E. Carlson. “A large-scale analysis of bioinformatics on GitHub”. In: *PLOS ONE* 13.10 (2018), e0205898. DOI: 10.1371/journal.pone.0205898.

[237] Danilo Russo and Christian C. Voigt. “The use of automated identification of bat echolocation calls in acoustic monitoring: A cautionary note for a sound analysis”. In: *Ecological Indicators* 66 (2016), pp. 598–602. DOI: 10.1016/j.ecolind.2016.02.036.

[238] RW. *Retraction Watch [Portal]*. https://retractionwatch.com/. Online; accessed 20 October 2022. n.d.

[239] Manuel Salinas-Navarro, Luis Alarcón-Martínez, Francisco Javier Valiente-Soriano, Arturo Ortín-Martínez, Manuel Jiménez-López, Marcelino Avilés-Trigueros, María Paz Villegas-Pérez, Pedro de la Villa, and Manuel Vidal-Sanz. “Functional and morphological effects of laser-induced ocular hypertension in retinas of adult albino Swiss mice”. In: *Molecular Vision* 15 (2009), p. 2578.

[240] David Schindler. *SoftwareKG-Mention-Entity*. Dataset available at: https://doi.org/10.5281/zenodo.10951778. Version 0.1. 2024. DOI: 10.5281/zenodo.10951778.

[241] David Schindler. *SoMeNLP*. [Python Package]. 2021. URL: https://github.com/dave-s477/SoMeNLP.

[242] David Schindler, Felix Bensmann, Stefan Dietze, and Frank Krüger. *SoftwareKG-PMC*. Dataset available at: https://doi.org/10.5281/zenodo.5713973. Version 0.1. 2021. DOI: 10.5281/zenodo.5713973.

[243] David Schindler, Felix Bensmann, Stefan Dietze, and Frank Krüger. *SoMeSci*. 2021. DOI: 10.5281/zenodo.4968738.

[244] David Schindler, Felix Bensmann, Stefan Dietze, and Frank Krüger. “SoMeSci- A 5 Star Open Data Gold Standard Knowledge Graph of Software Mentions in Scientific Articles”. In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. CIKM ’21. Virtual Event, Queensland, Australia, 2021, pp. 4574–4583. DOI: 10.1145/3459637.3482017.

[245] David Schindler, Felix Bensmann, Stefan Dietze, and Frank Krüger. “The role of software in science: a knowledge graph-based analysis of software mentions in PubMed Central”. In: *PeerJ Computer Science* 8 (2022), e835. DOI: 10.7717/peerj-cs.835.

[246] David Schindler, Tazin Hossain, Sascha Spors, and Frank Krüger. “A multi-level analysis of data quality for formal software citation”. In: *Quantitative Science Studies* (2024). Accepted for publication; currently available as Preprint from https://doi.org/10.48550/arXiv.2306.17535.

[247] David Schindler and Frank Krüger. *articlenizer*. [Python Package]. 2021. URL: https://github.com/dave-s477/articlenizer.

[248] David Schindler, Erjia Yan, Sascha Spors, and Frank Krüger. “Retracted articles use less free and open-source software and cite it worse”. In: *Quantitative Science Studies* (2023), pp. 1–23. DOI: 10.1162/qss_a_00275.

[249] David Schindler, Erjia Yan, Sascha Spors, and Frank Krüger. “Software use in retracted papers”. In: *Proceedings of ISSI 2023 — the 19th International Conference of the International Society for Scientometrics and Informetrics*. 2023, pp. 411–414. DOI: 10.5281/zenodo.8428921.

[250] David Schindler, Kristina Yordanova, and Frank Krüger. “An annotation scheme for references to research artefacts in scientific publications”. In: *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. 2019, pp. 52–57. DOI: 10.1109/PERCOMW.2019.8730730.

[251] David Schindler, Benjamin Zapilko, and Frank Krüger. “Investigating Software Usage in the Social Sciences: A Knowledge Graph Approach”. In: *The Semantic Web – ESWC 2020*. 2020, pp. 271–286. DOI: 10.1007/978-3-030-49461-2_16.

[252] David Schindler, Benjamin Zapilko, and Frank Krüger. *SoftwareKG (1.0)*. 2020. DOI: 10.5281/zenodo.3715147.

[253] Mike Schuster and Kuldip K Paliwal. “Bidirectional recurrent neural networks”. In: *IEEE transactions on Signal Processing* 45.11 (1997), pp. 2673–2681. DOI: 10.1109/78.650093.

[254] SciCrunch. *RRID Portal [Portal]*. https://scicrunch.org/resources/. Online; accessed 20 October 2022. n.d.

[255] SCImago. *SJR — SCImago Journal & Country Rank [Portal]*. http://www.scimagojr.com. Online; accessed 16 June 2022. n.d.

[256] Xin Shuai, Jason Rollins, Isabelle Moulinier, Tonya Custis, Mathilda Edmunds, and Frank Schilder. “A multidimensional investigation of the effects of publication retraction on scholarly impact”. In: *Journal of the Association for Information Science and Technology* 68.9 (2017), pp. 2225–2236. DOI: 10.1002/asi.23826.

[257] Andri Signorell. *DescTools: Tools for Descriptive Statistics*. [R Package] version 0.99.48. 2023. URL: https://CRAN.R-project.org/package=DescTools.

[258] Dalmeet Singh Chawla. “The unsung heroes of scientific software”. In: *Nature* 529.7584 (2016), pp. 115–116. DOI: 10.1038/529115a.

[259] Singhal, Amit. *Introducing the Knowledge Graph: things, not strings [Blog]*. https://blog.google/products/search/introducing-knowledge-graph-things-not/. Online; accessed 8 January 2024. 2012.

[260] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June Hsu, and Kuansan Wang. "An Overview of Microsoft Academic Service (MAS) and Applications". In: *Proceedings of the 24th international conference on world wide web*. 2015, pp. 243–246. DOI: 10.1145/2740908.2742839.

[261] Cristina P Sison and Joseph Glaz. "Simultaneous confidence intervals and sample size determination for multinomial proportions". In: *Journal of the American Statistical Association* 90.429 (1995), pp. 366–369. DOI: 10.2307/2291162.

[262] Magnus Thorstein Sletholt, Jo Erskine Hannay, Dietmar Pfahl, and Hans Petter Langtangen. "What do we know about scientific software development's agile practices?" In: *Computing in Science & Engineering* 14.2 (2011), pp. 24–37. DOI: 10.1109/MCSE.2011.113.

[263] Arfon M Smith, Daniel S Katz, and Kyle E Niemeyer. "Software citation principles". In: *PeerJ Computer Science* 2 (2016), e86. DOI: 10.7717/peerj-cs.86.

[264] Vanessa Sochat, Nicholas May, Ian Cosden, Carlos Martinez-Ortiz, and Sadie Bartholomew. "The Research Software Encyclopedia: a community framework to define research software". In: *Journal of Open Research Software* (2022). DOI: 10.5334/jors.359.

[265] Springer Nature Staff. *SN SciGraph Latest Release: Patents, Clinical Trials and many new features [Blog]*. https://communities.springernature.com/posts/sn-scigraph-latest-release-patents-clinical-trials-and-many-new-features. Online; accessed 25 January 2024. 2019.

[266] SSI. *Software Sustainability Institute [Portal]*. https://www.software.ac.uk. Online; accessed 20 October 2022. n.d.

[267] Shelley Stall, Geoffrey Bilder, Matthew Cannon, Neil Chue Hong, Scott Edmunds, Christopher C Erdmann, Michael Evans, Rosemary Farmer, Patricia Feeney, Michael Friedman, et al. "Journal Production Guidance for Software and Data Citations". In: *Scientific Data* 10.1 (2023), p. 656. DOI: 10.1038/s41597-023-02491-7.

[268] R Grant Steen. "Retractions in the scientific literature: is the incidence of research fraud increasing?" In: *Journal of medical ethics* 37.4 (2011), pp. 249–253. DOI: 10.1136/jme.2010.040923.

[269] R. G. Steen. "Retractions in the scientific literature: do authors deliberately commit research fraud?" In: *Journal of Medical Ethics* 37.2 (2010), pp. 113–117. DOI: 10.1136/jme.2010.038125.

[270] R. Grant Steen, Arturo Casadevall, and Ferric C. Fang. "Why Has the Number of Scientific Retractions Increased?" In: *PLOS ONE* 8.7 (2013), pp. 1–9. DOI: 10.1371/journal.pone.0068397.

[271] Pontus Stenetorp, Sampo Pyysalo, and Goran Topić. *BRAT*. [Software] version 1.3. URL: https://github.com/nlplab/brat.

[272] Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. "BRAT: a Web-based Tool for NLP-Assisted Text Annotation". In: *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*. Avignon, France, 2012, pp. 102–107.

[273] Emma Strubell, Ananya Ganesh, and Andrew McCallum. “Energy and Policy Considerations for Deep Learning in NLP”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy, 2019, pp. 3645–3650. DOI: 10.18653/v1/P19-1355.

[274] Dinanath Sulakhe, Mark D’Souza, Mustafa Syed, Alexis Rodriguez, Yi Zhang, E Glass, M Romine, and Natalia Maltsev. “Article withdrawn: GNARE: a grid-based server for the analysis of user submitted genomes”. In: *Nucleic Acids Res* 40.22 (2012), e177. DOI: 10.1093/nar/gkm366.

[275] S. H. Sun, M. S. Jiang, X. C. Ma, C. Y. Li, and L. M. Liang. “Retraction: Hacking on decoy-state quantum key distribution system with partial phase randomization”. In: *Sci Rep* 8 (2018), p. 46943. DOI: 10.1038/srep46943.

[276] Charles Sutton, Andrew McCallum, et al. “An Introduction to Conditional Random Fields”. In: *Foundations and Trends® in Machine Learning* 4.4 (2012), pp. 267–373. DOI: 10.1561/2200000013.

[277] Arunrat Tangmunkongvorakul, Patou Masika Musumari, Kulvadee Thongpibul, Kriengkrai Srithanaviboonchai, Teeranee Techasrivichien, S. Pilar Suguimoto, Masako Ono-Kihara, and Masahiro Kihara. “Association of excessive smartphone use with psychological well-being among university students in Chiang Mai, Thailand”. In: *PLOS ONE* 14.1 (2019), pp. 1–13. DOI: 10.1371/journal.pone.0210294.

[278] Leho Tedersoo, Rainer Küngas, Ester Oras, Kajar Köster, Helen Eenmaa, Äli Leijen, Margus Pedaste, Marju Raju, Anastasiya Astapova, Heli Lukner, et al. “Data sharing practices and data availability upon request differ across scientific disciplines”. In: *Scientific data* 8.1 (2021), p. 192.

[279] *The Research Software Encyclopedia*. https://docs.google.com/document/d/1wDb0udH9OrFWrMBsAVb8RrUMCKKRHoyEep7yveJ1d0k/edit. Online; accessed 19 October 2023. 2020.

[280] Saravanan Thirumuruganathan, Han Li, Nan Tang, Mourad Ouzzani, Yash Govind, Derek Paulsen, Glenn Fung, and AnHai Doan. “Deep learning for blocking in entity matching: a design space exploration”. In: *Proceedings of the VLDB Endowment* 14.11 (2021), pp. 2459–2472. DOI: 10.14778/3476249.3476294.

[281] Erik F. Tjong Kim Sang and Fien De Meulder. “Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition”. In: *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*. 2003, pp. 142–147.

[282] Robert Tomaszewski. “Analyzing citations to software by research area: A case study of MATLAB”. In: *Science & Technology Libraries* (2022), pp. 1–16. DOI: 10.1080/0194262X.2022.2070329.

[283] Yoshimasa Tsuruoka, Yuka Tateishi, Jin-Dong Kim, Tomoko Ohta, John McNaught, Sophia Ananiadou, and Jun’ichi Tsujii. “Developing a Robust Part-of-Speech Tagger for Biomedical Text”. In: *Advances in Informatics*. 2005, pp. 382–392. DOI: 10.1007/11573036_36.

[284] Yoshimasa Tsuruoka and Jun’ichi Tsujii. “Bidirectional Inference with the Easiest-First Strategy for Tagging Sequence Data”. In: *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*. Vancouver, British Columbia, Canada, 2005, pp. 467–474.

[285] Kurt Ulm. “Simple method to calculate the confidence interval of a standardized mortality ratio (SMR)”. In: *American journal of epidemiology* 131.2 (1990), pp. 373–375. DOI: 10.1093/oxfordjournals.aje.a115507.

[286] Willem Robert van Hage, with contributions from: Tomi Kauppinen, Benedikt Graeler, Christopher Davis, Jesper Hoeksema, Alan Ruttenberg, and Daniel Bahls. *SPARQL: SPARQL client*. [R Package] version 1.16. 2013. URL: https://CRAN.R-project.org/package=SPARQL.

[287] Richard Van Noorden et al. “Science publishing: The trouble with retractions”. In: *Nature* 478.7367 (2011), pp. 26–28. DOI: 10.1038/478026a.

[288] Richard Van Noorden, Brendan Maher, and Regina Nuzzo. “The top 100 papers”. In: *Nature News* 514.7524 (2014), p. 550.

[289] Guido Van Rossum and Fred L. Drake. *Python*. [Software] version 3.8.16. 2022. URL: https://www.python.org/.

[290] Simon Vandenhende, Stamatios Georgoulis, Wouter Van Gansbeke, Marc Proesmans, Dengxin Dai, and Luc Van Gool. “Multi-task learning for dense prediction tasks: A survey”. In: *IEEE transactions on pattern analysis and machine intelligence* 44.7 (2021), pp. 3614–3633. DOI: 10.1109/TPAMI.2021.3054719.

[291] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. “Attention is all you need”. In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[292] WN Venables and BD Ripley. “Modern applied statistics with S”. In: *Statistics and Computing. New York: Springer* (2002). DOI: 10.1007/978-0-387-21706-2.

[293] Denny Vrandečić and Markus Krötzsch. “Wikidata: a free collaborative knowledgebase”. In: *Communications of the ACM* 57.10 (2014), pp. 78–85. DOI: 10.1145/2629489.

[294] Alina Vretinaris, Chuan Lei, Vasilis Efthymiou, Xiao Qin, and Fatma Özcan. “Medical entity disambiguation using graph neural networks”. In: *Proceedings of the 2021 international conference on management of data*. 2021, pp. 2310–2318. DOI: 10.1145/3448016.3457328.

[295] Alex D Wade. “The semantic scholar academic graph (S2AG)”. In: *Companion Proceedings of the Web Conference 2022*. 2022, pp. 739–739. DOI: 10.1145/3487553.3527147.

[296] Caroline S Wagner and Koen Jonkers. “Open countries have strong science”. In: *Nature* 550.7674 (2017), pp. 32–33. DOI: 10.1038/550032a.

[297] Lena Wallensteen, Marius Zimmermann, Malin Thomsen Sandberg, Anton Gezelius, Anna Nordenström, Tatja Hirvikoski, and Svetlana Lajic. “Retraction notice to”Evaluation of behavioral problems after prenatal dexamethasone treatment in Swedish adolescents at risk of CAH” [Hormones and Behavior 85C (2016) 5-11]”. In: *Hormones and Behavior* 103 (2018), p. 140. DOI: 10.1016/j.yhbeh.2018.06.009.

[298] Sean Wallis. “Binomial confidence intervals and contingency tests: mathematical fundamentals and the evaluation of alternative methods”. In: *Journal of Quantitative Linguistics* 20.3 (2013), pp. 178–208. DOI: 10.1080/09296174.2013.799918.

[299] Ruijie Wang, Yuchen Yan, Jialu Wang, Yuting Jia, Ye Zhang, Weinan Zhang, and Xinbing Wang. “AceKG: A large-scale knowledge graph for academic data mining”. In: *Proceedings of the 27th ACM international conference on information and knowledge management*. 2018, pp. 1487–1490. DOI: 10.1145/3269206.3269252.

[300] Anne S. Warlaumont and Megan K. Finnegan. “Learning to Produce Syllabic Speech Sounds via Reward-Modulated Neural Plasticity”. In: *PLOS ONE* 11.1 (2016), pp. 1–30. DOI: 10.1371/journal.pone.0145096.

[301] Hadley Wickham et al. *ggplot2*. [R Package] version 3.3.5. 2023. URL: https://CRAN.R-project.org/package=ggplot2.

[302] Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN: 978-3-319-24277-4.

[303] Hadley Wickham et al. *tidyverse*. [R Package] version 2.0.0. 2021. URL: https://CRAN.R-project.org/package=tidyverse/.

[304] Hadley Wickham et al. “Welcome to the tidyverse”. In: *Journal of Open Source Software* 4.43 (2019), p. 1686. DOI: 10.21105/joss.01686.

[305] Michel Wijkstra, Timo Lek, Tobias Kuhn, Kasper Welbers, and Mickey Steijaert. “Living literature reviews”. In: *Proceedings of the 11th on Knowledge Capture Conference*. 2021, pp. 241–248. DOI: 10.1145/3460210.3493567.

[306] Eva C Wikberg, Pascale Sicotte, Fernando A Campos, and Nelson Ting. “Between-group variation in female dispersal, kin composition of groups, and proximity patterns in a black-and-white colobus monkey (Colobus vellerosus)”. In: *PLoS One* 7.11 (2012), e48740. DOI: 10.1371/journal.pone.0048740.

[307] Wikidata. *Wikidata:Statistics [Website]*. https://www.wikidata.org/wiki/Wikidata:Statistics. Online; accessed 25 January 2024. n.d.

[308] Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. “The FAIR Guiding Principles for scientific data management and stewardship”. In: *Scientific data* 3.1 (2016), pp. 1–9. DOI: 10.1038/sdata.2016.18.

[309] Christopher KI Williams and Carl Edward Rasmussen. “Model Selection and Adaptation of Hyperparameters”. In: *Gaussian Processes for Machine Learning*. The MIT Press, 2005. DOI: 10.7551/mitpress/3206.003.0008.

[310] Thomas Wolf et al. *Transformers*. [Python Package] version 4.6.1. 2020. URL: https://github.com/huggingface/transformers.

[311] Thomas Wolf et al. “Transformers: State-of-the-Art Natural Language Processing”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, 2020, pp. 38–45. DOI: 10.18653/v1/2020.emnlp-demos.6.

[312] Adrienne Wood, Jared Martin, and Paula Niedenthal. “Towards a social functional account of laughter: Acoustic features convey reward, affiliation, and dominance”. In: *PloS ONE* 12.8 (2017), e0183811. DOI: 10.1371/journal.pone.0183811.

[313] Jian Xu, Sunkyu Kim, Min Song, Minbyul Jeong, Donghyeon Kim, Jaewoo Kang, Justin F. Rousseau, Xin Li, Weijia Xu, Vetle I. Torvik, Yi Bu, Chongyan Chen, Islam Akef Ebeid, Daifeng Li, and Ying Ding. “Building a PubMed knowledge graph”. In: *Scientific Data* 7.1 (2020). DOI: 10.1038/s41597-020-0543-2.

[314] Jian Xu, Sunkyu Kim, Min Song, Minbyul Jeong, Donghyeon Kim, Jaewoo Kang, Justin F. Rousseau, Xin Li, Weijia Xu, Vetle I. Torvik, Yi Bu, Chongyan Chen, Islam Akef Ebeid, Daifeng Li, and Ying Ding. *PubMedKG*. [Dataset] PKG2020S4 (1781-Dec. 2020), Version 4. 2020. URL: http://er.tacc.utexas.edu/datasets/ped.

[315] Bishan Yang, Claire Cardie, and Peter Frazier. "A Hierarchical Distance-dependent Bayesian Model for Event Coreference Resolution". In: *Transactions of the Association for Computational Linguistics* 3 (2015), pp. 517–528. DOI: 10.1162/tacl_a_00155.

[316] Alexandros Zeakis, George Papadakis, Dimitrios Skoutas, and Manolis Koubarakis. "Pre-trained embeddings for entity resolution: An experimental analysis". In: *Proceedings of the VLDB Endowment* 16.9 (2023), pp. 2225–2238. DOI: 10.14778/3598581.3598594.

[317] Barry R Zeeberg, Joseph Riss, David W Kane, Kimberly J Bussey, Edward Uchio, W Marston Linehan, J Carl Barrett, and John N Weinstein. "Mistaken Identifiers: Gene name errors can be introduced inadvertently when using Excel in bioinformatics". In: *BMC Bioinformatics* 5.1 (2004), p. 80. DOI: 10.1186/1471-2105-5-80.

[318] Yutao Zeng, Xiaolong Jin, Saiping Guan, Jiafeng Guo, and Xueqi Cheng. "Event Coreference Resolution with their Paraphrases and Argument-aware Embeddings". In: *Proceedings of the 28th International Conference on Computational Linguistics*. Barcelona, Spain (Online), 2020, pp. 3084–3094. DOI: 10.18653/v1/2020.coling-main.275.

[319] Jinghui Zhang, David A Wheeler, Imtiaz Yakub, Sharon Wei, Raman Sood, William Rowe, Paul P Liu, Richard A Gibbs, and Kenneth H Buetow. "SNPdetector: A Software Tool for Sensitive and Accurate SNP Detection". In: *PLOS Computational Biology* 1.5 (2005), pp. 1–10. DOI: 10.1371/journal.pcbi.0010053.

[320] Kejun Zhang, Dong Chen, Xuelong Jiao, Shaoyan Zhang, Xiangping Liu, Jingyu Cao, Liqun Wu, and Dongsheng Wang. "Retraction. Slug enhances invasion ability of pancreatic cancer cells through upregulation of matrix metalloproteinase-9 and actin cytoskeleton remodeling". In: *Lab Invest* 92.12 (2012), p. 1801. DOI: 10.1038/labinvest.2012.138.

[321] Wei Zhang, Hao Wei, Bunyamin Sisman, Xin Luna Dong, Christos Faloutsos, and Davd Page. "Autoblock: A hands-off blocking framework for entity matching". In: *Proceedings of the 13th International Conference on Web Search and Data Mining*. 2020, pp. 744–752. DOI: 10.1145/3336191.3371813.

[322] Xinsong Zhang, Pengshuai Li, Weijia Jia, and Hai Zhao. "Multi-labeled relation extraction with attentive capsule network". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 7484–7491. DOI: 10.1609/aaai.v33i01.33017484.

[323] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 19–27. DOI: 10.1109/ICCV.2015.11.

[324] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. "A comprehensive survey on transfer learning". In: *Proceedings of the IEEE* 109.1 (2020), pp. 43–76. DOI: 10.1109/JPROC.2020.3004555.

[325] Mark Ziemann, Yotam Eren, and Assam El-Osta. "Gene name errors are widespread in the scientific literature". In: *Genome Biology* 17.1 (2016). DOI: 10.1186/s13059-016-1044-7.

Appendix

A Software Analyses

	Article	Approach	Dataset	Domain	Performance
Manual	Howison and Bullard [118]	Analysis	90 Articles	B	O=83%
	Nangia and Katz [192]	Analysis	40 Articles	N	-
	Duck et al. [77]	Annotation	85 Articles	BI	F=80%
	Du et al. [74]	Annotation	4971 Art.	LS & E	F=76%
Automatic	Duck et al. [77]	Rules	BioNerDs	BI	F=53%
	Duck et al. [76]	Rules & CRF	BioNerDs	BI	F=63%
	Duck et al. [78]	Rules & RF	BioNerDs	BI	F=67%
	Pan et al. [205]	IB	-	LS	F=58%
	Lopez et al. [173]	SciBERT-CRF	Softcite	LS & E	F=71−74%†
	Bassinet et al. [24]	SciBERT-CRF	Softcite v2	LS & E	F=81%
	Istrate et al. [128]	SciBERT	Softcite	LS & E	F=92%†

Table A1: Overview of research on software mention identification, split into the main categories of manual and automatic. Manual approaches are further divided between approaches focused on analysis and approaches for data annotation. For automatic methods the approach closer describes the used algorithms: R=manually engineered rules; CRF=Conditional Random Fields; RF=Random Forest; IB=Iterative Bootstrapping. For manual approaches the dataset describes the number of analyzed articles; for automatic approaches it describes the dataset for training. Domains: B=Biology; N=Nature (Journal); BI=Bioinformatics; LS=Life Sciences; E=Economics. Note that provided quality measures cannot be directly compared, but are provided to allow a better assessment of individual approaches: O=Overlap; F=FScore. †: Lopez et al. [173] and Istrate et al. [128] use different validation methods on Softcite making their results not directly comparable. Note that the method of Li and Yan [165] is not listed here due to its restriction to R packages.

B Confidence Intervals (CIs)

For count-based variables the calculation of CIs is based on the Poisson distribution, with the calculation based on the formula of Ulm [285]:

$$C_l = \frac{\chi^2_{\alpha/2,\,2n}}{2},\ C_u = \frac{\chi^2_{1-\alpha/2,\,2(n+1)}}{2} \tag{7.1}$$

with n denoting the number of samples and α=.05 for a 95% CI, and C_l and C_u denoting the lower and upper boundaries of the interval, respectively.

For binary variables, CIs are based on the binomial distribution, calculated as commonly known under the term Wald-interval, using an approximation by a normal distribution as described by Wallis [298]:

$$C_{l/u} \pm z\sqrt{\frac{\hat{p}(1-\hat{p})}{n}} \tag{7.2}$$

with $\hat{p}$ being the estimated probability, n denoting the sample size and a z-score of z=1.96 to achieve a 95% CI, and C_l and C_u denoting the lower and upper boundaries of the interval, respectively.

A multinomial distribution is considered in one specific case. Here, the calculation of the CIs is based on the methods proposed by Glaz and Sison [92] and Sison and Glaz [261].

C Statistics: Manual Analysis

				Structure						Correctness			
Metadata	Citation	Database	n	NA	(%)	US	(%)	S	(%)	C	(%)	E	(%)
Name	D	JATS	158	9	5.7	129	81.6	20	12.7	-	-	-	-
	D	CRO	147	11	7.5	120	81.6	16	10.9	136	92.5	0	0
	D	SEM	121	6	5.0	86	71.1	29	24.0	111	91.7	4	3.3
	M	JATS	47	4	8.5	29	61.7	14	29.8	-	-	-	-
	M	CRO	41	7	17.1	24	58.5	10	24.4	34	82.9	0	0
	M	SEM	40	3	7.5	13	32.5	24	60.0	35	87.5	2	5.0
Identifier	D	JATS	158	86	54.4	0	0	72	45.6	-	-	-	-
	D	CRO	147	90	61.2	6	4.1	51	34.7	57	38.8	0	0
	D	SEM	121	103	85.1	13	10.7	5	4.1	15	12.4	3	2.5
	M	JATS	47	28	59.6	0	0	19	40.4	-	-	-	-
	M	CRO	41	29	70.7	1	2.4	11	26.8	12	29.3	0	0
	M	SEM	40	40	100.0	-	-	-	-	-	-	-	-
Creator	D	JATS	158	18	11.4	88	55.7	52	32.9	-	-	-	-
	D	CRO	147	36	24.5	87	59.2	24	16.3	94	63.9	17	11.6
	D	SEM	121	41	33.9	39	32.2	41	33.9	55	45.5	25	20.7
	M	JATS	47	1	2.1	19	40.4	27	57.4	-	-	-	-
	M	CRO	41	16	39.0	16	39.0	9	22.0	17	41.5	8	19.5
	M	SEM	40	6	15.0	7	17.5	27	67.5	29	72.5	5	12.5
Version	D	JATS	158	60	38.0	92	58.2	6	3.8	-	-	-	-
	D	CRO	147	64	43.5	80	54.4	3	2.0	81	55.1	2	1.4
	D	SEM	121	53	43.8	67	55.4	1	.8	51	42.1	17	14.0
	M	JATS	47	36	76.6	10	21.3	1	2.1	-	-	-	-
	M	CRO	41	32	78.0	9	22.0	0	0	9	22.0	0	0
	M	SEM	40	31	77.5	8	20.0	1	2.5	6	15.0	3	7.50
Date	D	JATS	158	20	12.7	89	56.3	49	31.0	-	-	-	-
	D	CRO	147	19	12.9	80	54.4	48	32.7	128	87.1	0	0
	D	SEM	121	18	14.9	10	8.3	93	76.9	77	63.6	26	21.5
	M	JATS	47	2	4.3	17	36.2	28	59.6	-	-	-	-
	M	CRO	41	2	4.9	17	41.5	22	53.7	37	90.2	2	4.9
	M	SEM	40	2	5.0	0	0	38	95.0	16	40.0	22	55.0

Table A2: Overview of direct software citation and citation to manuals representation by publisher, Crossref, and Semantic Scholar across different metadata. Column "Citation", distinguishes between direct citations (D) and manuals (M); "Database" distinguishes the data source between publisher (JATS), Crossref (CRO), and Semantic Scholar (SEM); "Structure" NA refers to not available information, US to unstructured information, and S to structured information; "Correctness" C refers to correct information and E to wrong information. The JATS data has no correctness information because it is correct by definition. The values of NA, US, and S sum to one, and the values of NA, C, and E, also, because a correctness cannot be determined for not represented information. Missing references are not included in any counts.

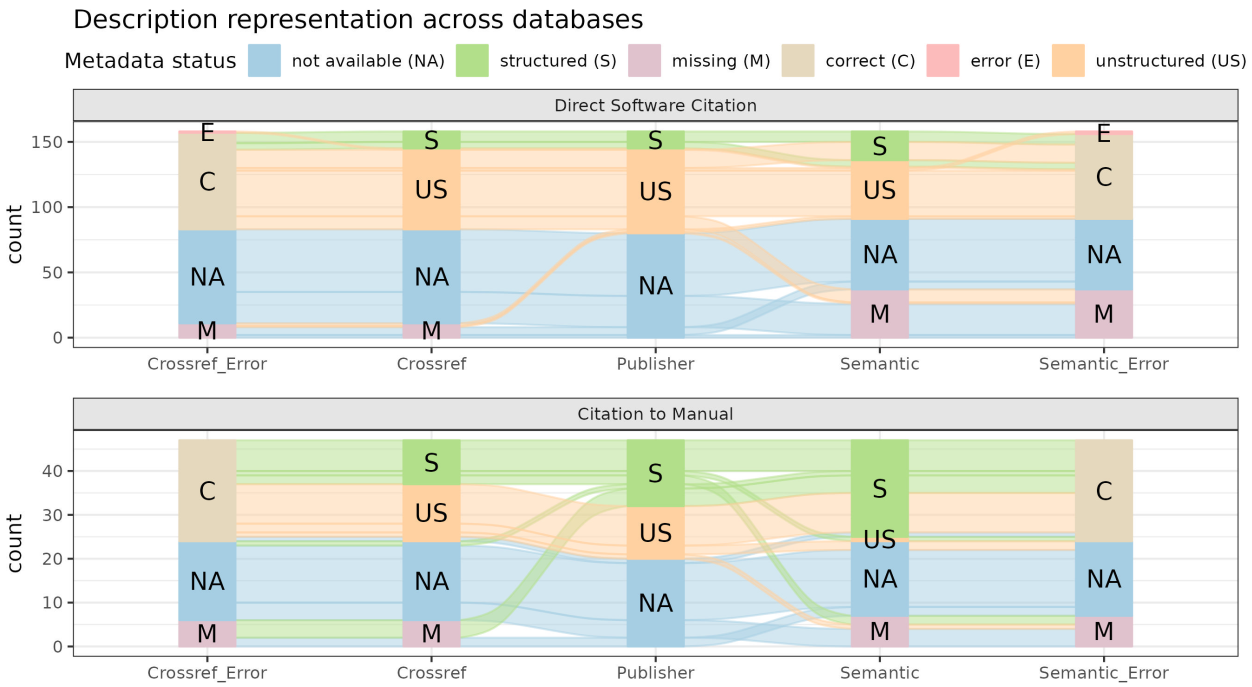


Figure A1: Adapted alluvial plot illustrating accuracy of software description representation in terms of availability, structure, and correctness within direct software citations and citations to manuals, following the principle of Figure 3.6.

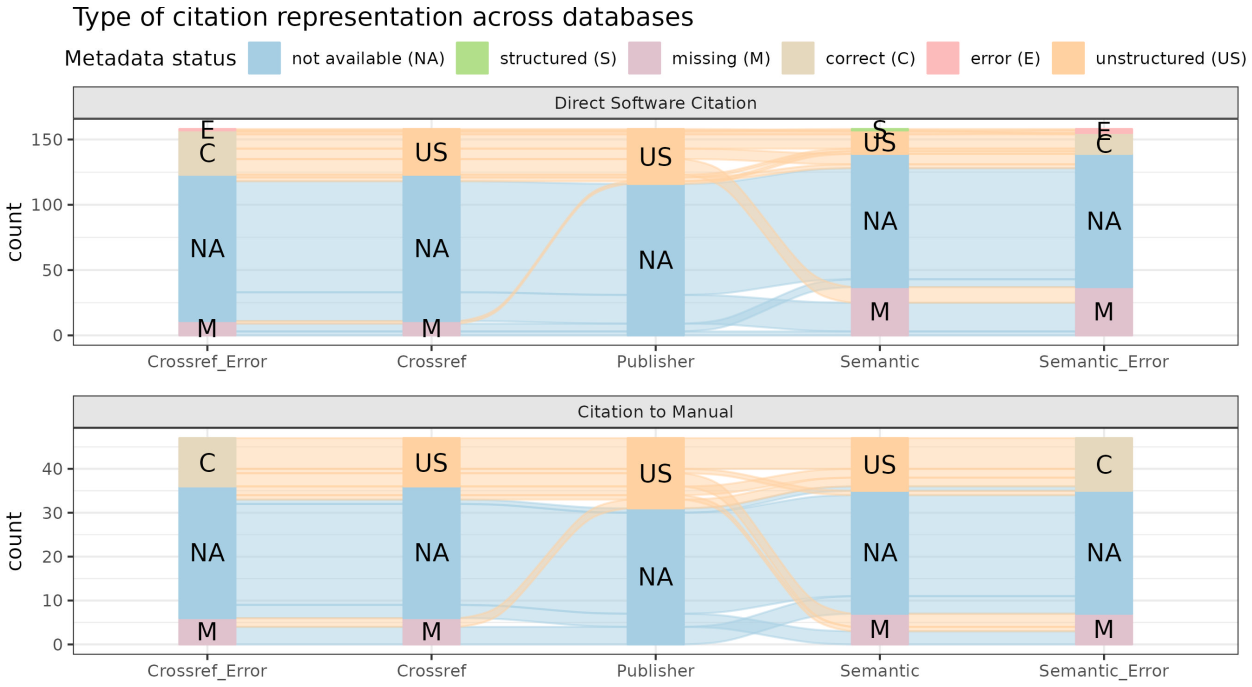


Figure A2: Adapted alluvial plot illustrating accuracy of software type of citation representation in terms of availability, structure, and correctness within direct software citations and citations to manuals, following the principle of Figure 3.6.

D Automatic Information Extraction

```
select distinct ?u ?l ?n ?red_name ?w_l ?s ?d ?d_foaf_name ?d_label ?d_dbp_name ?d_label_ori where {
    {
        ?s rdf:type dbo:Software.
        FILTER NOT EXISTS {?s rdf:type dbo:VideoGame}
        OPTIONAL {
            ?red dbo:wikiPageRedirects ?s.
            OPTIONAL {?red rdfs:label ?red_name.}
        }
        OPTIONAL {
            ?s rdfs:label ?l.
            FILTER (LANG(?l) = 'de' or LANG(?l) = 'fr' or LANG(?l) = 'es') .
        }
        ?s rdfs:label ?u. FILTER (LANG(?u) = 'en').
        OPTIONAL {?s foaf:name ?n.}
        OPTIONAL {
            ?wiki_dis dbo:wikiPageDisambiguates ?s.
            ?wiki_dis rdfs:label ?w_l.
            FILTER ( LANG( ?w_l) ='en').}
        OPTIONAL {
            ?s dbo:developer ?d.
            OPTIONAL {?d foaf:name ?d_foaf_name.}
            OPTIONAL {
                ?d rdfs:label ?d_label.
                FILTER(LANG(?d_label) = 'en').
            }
            OPTIONAL {?d dbp:name ?d_dbp_name.}
            OPTIONAL {
                ?d dbo:wikiPageRedirects ?d_ori.
                ?d_ori rdfs:label ?d_label_ori.
                FILTER (LANG(?d_label_ori)='en')
            }
        }
    } UNION {
        ?s rdf:type dbo:ProgrammingLanguage.
        ?s rdfs:label ?u.
        FILTER(LANG(?u)='en')
    }
}
```

Listing A22: Executed SPARQL query gathering information on software contained in DBpedia. Different alternative names for software are extracted utilizing the fields *rdfs:label*, *foaf:name*, *dbo:wikiPageDisambiguates*, and *dbo:wikiPageRedirects*. The query considers names in English, Spanish, German, and French to cover language based naming differences. The query considers all subcategories of software except video games, and gathers programming languages separately as they are not considered software in DBpedia.

E Statistics: Large-Scale Analysis

Main research domain	Research subcategories
Physics and Astronomy	Acoustics and Ultrasonics, Astronomy and Astrophysics, Atomic and Molecular Physics, and Optics, Condensed Matter Physics, Instrumentation, Nuclear and High Energy Physics, Physics and Astronomy (miscellaneous), Radiation, Statistical and Nonlinear Physics, Surfaces and Interfaces
Chemistry	Analytical Chemistry, Chemistry (miscellaneous), Electrochemistry, Inorganic Chemistry, Organic Chemistry, Physical and Theoretical Chemistry, Spectroscopy
Social Sciences	Anthropology, Archeology, Communication, Cultural Studies, Demography, Development, E-learning, Education, Gender Studies, Geography, Planning and Development, Health (social science), Human Factors and Ergonomics, Law, Library and Information Sciences, Life-span and Life-course Studies, Linguistics and Language, Political Science and International Relations, Public Administration, Safety Research, Social Sciences (miscellaneous), Social Work, Sociology and Political Science, Transportation, Urban Studies
Materials Science	Biomaterials, Ceramics and Composites, Electronic, Optical and Magnetic Materials, Materials Chemistry, Materials Science (miscellaneous), Metals and Alloys, Nanoscience and Nanotechnology, Polymers and Plastics, Surfaces, Coatings and Films
Engineering	Aerospace Engineering, Architecture, Automotive Engineering, Biomedical Engineering, Building and Construction, Civil and Structural Engineering, Computational Mechanics, Control and Systems Engineering, Electrical and Electronic Engineering, Engineering (miscellaneous), Industrial and Manufacturing Engineering, Mechanical Engineering, Mechanics of Materials, Media Technology, Ocean Engineering, Safety, Risk, Reliability and Quality
Economics, Econometrics and Finance	Economics and Econometrics, Economics, Econometrics and Finance (miscellaneous), Finance
Multidisciplinary	Multidisciplinary
Energy	Energy (miscellaneous), Energy Engineering and Power Technology, Fuel Technology, Nuclear Energy and Engineering, Renewable Energy, Sustainability and the Environment
Agricultural and Biological Sciences	Agricultural and Biological Sciences (miscellaneous), Agronomy and Crop Science, Animal Science and Zoology, Aquatic Science, Ecology, Evolution, Behavior and Systematics, Food Science, Forestry, Horticulture, Insect Science, Plant Science, Soil Science
Environmental Science	Ecological Modeling, Ecology, Environmental Chemistry, Environmental Engineering, Environmental Science (miscellaneous), Global and Planetary Change, Health, Toxicology and Mutagenesis, Management, Monitoring, Policy and Law, Nature and Landscape Conservation, Pollution, Waste Management and Disposal, Water Science and Technology
Veterinary	Equine, Food Animals, Small Animals, Veterinary (miscellaneous)
Nursing	Advanced and Specialized Nursing, Assessment and Diagnosis, Care Planning, Community and Home Care, Critical Care Nursing, Emergency Nursing, Fundamentals and Skills, Gerontology, Issues, Ethics and Legal Aspects, LPN and LVN, Leadership and Management, Maternity and Midwifery, Medical and Surgical Nursing, Nurse Assisting, Nursing (miscellaneous), Nutrition and Dietetics, Oncology (nursing), Pediatrics, Pharmacology (nursing), Psychiatric Mental Health, Research and Theory, Review and Exam Preparation
Decision Sciences	Statistics, Probability and Uncertainty, Information Systems and Management, Management Science and Operations Research
Earth and Planetary Sciences	Atmospheric Science, Computers in Earth Sciences, Earth and Planetary Sciences (miscellaneous), Earth-Surface Processes, Economic Geology, Geochemistry and Petrology, Geology, Geophysics, Geotechnical Engineering and Engineering Geology, Oceanography, Paleontology, Space and Planetary Science, Stratigraphy
Pharmacology, Toxicology and Pharmaceutics	Drug Discovery, Pharmaceutical Science, Pharmacology, Pharmacology, Toxicology and Pharmaceutics (miscellaneous), Toxicology

Mathematics	Algebra and Number Theory, Analysis, Applied Mathematics, Computational Mathematics, Control and Optimization, Discrete Mathematics and Combinatorics, Geometry and Topology, Logic, Mathematical Physics, Mathematics (miscellaneous), Modeling and Simulation, Numerical Analysis, Statistics and Probability, Theoretical Computer Science
Computer Science	Artificial Intelligence, Computational Theory and Mathematics, Computer Graphics and Computer-Aided Design, Computer Networks and Communications, Computer Science (miscellaneous), Computer Science Applications, Computer Vision and Pattern Recognition, Hardware and Architecture, Human-Computer Interaction, Information Systems, Signal Processing, Software
Biochemistry, Genetics and Molecular Biology	Aging, Biochemistry, Biochemistry, Genetics and Molecular Biology (miscellaneous), Biophysics, Biotechnology, Cancer Research, Cell Biology, Clinical Biochemistry, Developmental Biology, Endocrinology, Genetics, Molecular Biology, Molecular Medicine, Physiology, Structural Biology
Dentistry	Dentistry (miscellaneous), Oral Surgery, Orthodontics, Periodontics
Neuroscience	Behavioral Neuroscience, Biological Psychiatry, Cellular and Molecular Neuroscience, Cognitive Neuroscience, Developmental Neuroscience, Endocrine and Autonomic Systems, Neurology, Neuroscience (miscellaneous), Sensory Systems
Arts and Humanities	Archeology (arts and humanities), Arts and Humanities (miscellaneous), Conservation, History, History and Philosophy of Science, Language and Linguistics, Literature and Literary Theory, Music, Philosophy, Religious Studies, Visual Arts and Performing Arts
Psychology	Applied Psychology, Clinical Psychology, Developmental and Educational Psychology, Experimental and Cognitive Psychology, Neuropsychology and Physiological Psychology, Psychology (miscellaneous), Social Psychology
Business, Management and Accounting	Accounting, Business and International Management, Business, Management and Accounting (miscellaneous), Industrial Relations, Management Information Systems, Management of Technology and Innovation, Marketing, Organizational Behavior and Human Resource Management, Strategy and Management, Tourism, Leisure and Hospitality Management
Medicine	Anatomy, Anesthesiology and Pain Medicine, Biochemistry (medical), Cardiology and Cardiovascular Medicine, Complementary and Alternative Medicine, Critical Care and Intensive Care Medicine, Dermatology, Drug Guides, Embryology, Emergency Medicine, Endocrinology, Diabetes and Metabolism, Epidemiology, Family Practice, Gastroenterology, Genetics (clinical), Geriatrics and Gerontology, Health Informatics, Health Policy, Hematology, Hepatology, Histology, Immunology and Allergy, Infectious Diseases, Internal Medicine, Medicine (miscellaneous), Microbiology (medical), Nephrology, Neurology (clinical), Obstetrics and Gynecology, Oncology, Ophthalmology, Orthopedics and Sports Medicine, Otorhinolaryngology, Pathology and Forensic Medicine, Pediatrics, Perinatology and Child Health, Pharmacology (medical), Physiology (medical), Psychiatry and Mental Health, Public Health, Environmental and Occupational Health, Pulmonary and Respiratory Medicine, Radiology, Nuclear Medicine and Imaging, Rehabilitation, Reproductive Medicine, Reviews and References (medical), Rheumatology, Surgery, Transplantation, Urology
Immunology and Microbiology	Applied Microbiology and Biotechnology, Immunology, Immunology and Microbiology (miscellaneous), Microbiology, Parasitology, Virology
Health Professions	Chiropractics, Complementary and Manual Therapy, Health Information Management, Health Professions (miscellaneous), Medical Laboratory Technology, Occupational Therapy, Optometry, Pharmacy, Physical Therapy, Sports Therapy and Rehabilitation, Podiatry, Radiological and Ultrasound Technology, Speech and Hearing, Sports Science
Chemical Engineering	Bioengineering, Catalysis, Chemical Engineering (miscellaneous), Chemical Health and Safety, Colloid and Surface Chemistry, Filtration and Separation, Fluid Flow and Transfer Processes, Process Chemistry and Technology

Table A3: Overview of the 27 main research domains and 3 of their sub categories that were used to group journals. Bold font highlights the abbreviation of the respective research domain used here.

Variable	OR	95% CI	
Agricultural	1.81	1.79	1.83
Computer	1.73	1.67	1.79
Physics	1.73	1.68	1.77
Business	1.57	1.37	1.80
Biochemistry	1.40	1.38	1.41
Multidisciplinary	1.25	1.23	1.27
Mathematics	1.21	1.16	1.27
Health	1.21	1.18	1.25
Dentistry	1.16	1.12	1.21
Materials	1.13	1.10	1.15
Decision	1.13	(.91	1.42)
Year	1.09	1.09	1.09
Neuroscience	1.06	1.04	1.08
Journal Rank	1.05	1.05	1.05
Medicine	1.03	1.02	1.04
Citation Rank	.99	.98	.99
Immunology	.96	.95	.97
Veterinary	.96	.92	.99
Psychology	.89	.87	.91
Pharmacology	.84	.83	.85
Energy	.82	.74	.90
Environmental	.80	.78	.81
Nursing	.73	.71	.75
Chemistry	.71	.69	.72
Engineering	.70	.68	.72
Social	.68	.66	.70
Earth	.56	.51	.62
Economics	.49	.43	.56
Chemical	.47	.46	.48
Arts	.46	.43	.49

Table A4: Result of the logistic regression predicting if an article is using software (true=1, false=0) considering its publication year (Year), scientific domain, impact of the publishing journal (Journal Rank), and the article impact in citation count (Citation Rank). OR=Odds Ratio; OR>1 indicates that a variable is associated with a higher likelihood of a software usage in an article; non significant values are indicated by the CI including 1, which is also highlighted by the use of brackets; the table is ordered by OR.

Variable	OR	95% CI	
Computer	2.39	2.36	2.43
Decision	2.10	1.95	2.25
Biochemistry	1.34	1.33	1.34
Immunology	1.22	1.21	1.22
Materials	1.21	1.19	1.23
Engineering	1.19	1.17	1.20
Agricultural	1.17	1.16	1.17
Physics	1.16	1.15	1.18
Multidisciplinary	1.10	1.09	1.11
Year	1.03	1.03	1.03
Journal Rank	1.03	1.03	1.03
Neuroscience	1.03	1.02	1.04
Chemistry	1.03	1.02	1.05
Citation Rank	1.01	1.01	1.01
Pharmacology	.98	.98	.99
Health	.92	.90	.94
Arts	.89	.84	.94
Business	.84	.79	.89
Psychology	.84	.83	.86
Energy	.84	.80	.88
Environmental	.82	.81	.83
Social	.82	.80	.84
Veterinary	.79	.77	.81
Earth	.79	.74	.84
Mathematics	.69	.68	.70
Medicine	.67	.67	.68
Dentistry	.67	.65	.69
Economics	.60	.52	.69
Nursing	.56	.55	.57
Chemical	.52	.51	.53

Table A5: Result of the quasi-poisson regression predicting the number of software an article mentions considering its publication year (Year), scientific domain, impact of the publishing journal (Journal Rank), and the article impact in citation count (Citation Rank). OR=Odds Ratio; OR>1 indicates that a variable is associated with a higher number of a software per article; non significant values are indicated by the CI including 1, which is also highlighted by the use of brackets; the table is ordered by OR.

Variable	OR	95% CI	
Materials	3.28	3.21	3.36
Decision	3.00	2.51	3.57
Computer	2.29	2.22	2.35
Agricultural	1.54	1.53	1.55
Earth	1.47	1.29	1.68
# Software	1.35	1.35	1.35
Psychology	1.33	1.30	1.37
Mathematics	1.31	1.27	1.36
Immunology	1.30	1.29	1.31
Arts	1.23	1.13	1.35
Engineering	1.20	1.16	1.23
Biochemistry	1.18	1.17	1.19
Physics	1.18	1.15	1.21
Multidisciplinary	1.13	1.12	1.15
Environmental	1.09	1.07	1.11
Veterinary	1.04	1.01	1.07
Journal Rank	1.03	1.03	1.03
Year	1.01	1.01	1.01
Citation Rank	.99	.99	.99
Neuroscience	.94	.93	.95
Chemistry	.91	.89	.93
Energy	.89	.82	.97
Pharmacology	.86	.85	.88
Medicine	.63	.63	.64
Social	.62	.59	.65
Health	.60	.58	.62
Chemical	.60	.59	.62
Business	.58	.51	.65
Economics	.55	.42	.72
Dentistry	.35	.33	.37
Nursing	.34	.33	.35

Table A6: Result of the logistic regression predicting if individual software mentioned in articles is free considering its publication year (Year), scientific domain, impact of the publishing journal (Journal Rank), the article impact in citation count (Citation Rank), and the number of software used within the article (# Software). OR=Odds Ratio; OR>1 indicates that a variable is associated with a higher amount of freely available software; non significant values are indicated by the CI including 1, which is also highlighted by the use of brackets; the table is ordered by OR.

Variable	OR	95% CI	
Decision	3.01	2.59	3.50
Engineering	2.54	2.48	2.61
Earth	1.76	1.55	2.00
Computer	1.59	1.55	1.63
Agricultural	1.53	1.52	1.54
Psychology	1.40	1.36	1.43
Mathematics	1.33	1.29	1.37
Immunology	1.28	1.27	1.29
# Software	1.22	1.22	1.22
Biochemistry	1.21	1.20	1.22
Multidisciplinary	1.18	1.16	1.19
Veterinary	1.16	1.12	1.19
Arts	1.15	1.05	1.25
Environmental	1.12	1.10	1.14
Journal Rank	1.05	1.05	1.05
Year	1.02	1.02	1.02
Neuroscience	1.01	(1.00	1.03)
Citation Rank	1.00	(.99	1.00)
Chemical	1.00	(.97	1.03)
Pharmacology	.95	.93	.96
Physics	.84	.81	.86
Chemistry	.80	.78	.82
Business	.75	.67	.85
Energy	.74	.69	.80
Health	.70	.67	.73
Medicine	.65	.65	.66
Social	.63	.60	.66
Economics	.46	.34	.63
Materials	.42	.41	.43
Dentistry	.37	.34	.39
Nursing	.32	.31	.33

Table A7: Result of the logistic regression predicting if individual software mentioned in articles is open source considering its publication year (Year), scientific domain, impact of the publishing journal (Journal Rank), the article impact in citation count (Citation Rank), and the number of software used within the article (# Software). OR=Odds Ratio; OR>1 indicates that a variable is associated with a higher amount of open source software; non significant values are indicated by the CI including 1, which is also highlighted by the use of brackets; the table is ordered by OR.

Variable	OR	95% CI	
Materials	1.62	1.58	1.65
Nursing	1.39	1.35	1.43
Medicine	1.24	1.23	1.24
Chemistry	1.20	1.17	1.23
Energy	1.19	1.11	1.28
Agricultural	1.17	1.16	1.17
Environmental	1.16	1.14	1.18
Chemical	1.14	1.11	1.17
Dentistry	1.07	1.03	1.12
Year	1.05	1.05	1.05
Citation Rank	1.00	(1.00	1.00)
# Software	.99	.99	.99
Veterinary	.98	(.95	1.01)
Psychology	.98	(.96	1.01)
Journal Rank	.97	.97	.97
Pharmacology	.96	.94	.97
Health	.92	.90	.95
Immunology	.90	.89	.91
Neuroscience	.84	.83	.85
Social	.84	.81	.87
Physics	.84	.82	.86
Arts	.83	.77	.91
Open	.81	.80	.82
Multidisciplinary	.77	.76	.78
Computer	.74	.72	.76
Economics	.72	.59	.88
Biochemistry	.70	.69	.70
Business	.70	.63	.78
Mathematics	.68	.66	.70
Earth	.60	.53	.68
Free	.45	.45	.46
Engineering	.37	.36	.38
Decision	.33	.28	.40

Table A8: Result of the logistic regression predicting if the version was provided for individual software mentioned in articles considering its publication year (Year), scientific domain, impact of the publishing journal (Journal Rank), the article impact in citation count (Citation Rank), the number of software used within the article (# Software), and whether the software itself is available freely (Free) and open source (Open). OR=Odds Ratio; OR>1 indicates that a variable is associated with a higher amount of version mentions; non significant values are indicated by the CI including 1, which is also highlighted by the use of brackets; the table is ordered by OR.

Variable	OR	95% CI	
Materials	10.88	10.60	11.17
Economics	1.95	1.56	2.43
Neuroscience	1.43	1.42	1.45
Physics	1.29	1.26	1.33
Environmental	1.25	1.23	1.28
Immunology	1.22	1.21	1.23
Veterinary	1.22	1.19	1.26
Dentistry	1.19	1.14	1.24
Nursing	1.16	1.13	1.19
Computer	1.13	1.10	1.17
Biochemistry	1.05	1.05	1.06
Medicine	1.05	1.04	1.06
Pharmacology	1.05	1.03	1.07
Year	1.02	1.02	1.02
Citation Rank	1.01	1.01	1.01
Journal Rank	1.00	(1.00	1.00)
Agricultural	.95	.94	.95
Chemistry	.95	.93	.98
Multidisciplinary	.94	.93	.96
Decision	.93	(.75	1.15)
# Software	.90	.90	.90
Chemical	.88	.85	.91
Psychology	.88	.86	.90
Arts	.84	.76	.92
Open	.84	.82	.85
Health	.80	.78	.83
Social	.58	.56	.60
Energy	.47	.42	.51
Earth	.46	.39	.54
Business	.31	.27	.35
Engineering	.27	.26	.28
Mathematics	.23	.22	.24
Free	.17	.16	.17

Table A9: Result of the logistic regression predicting if the developer was provided for individual software mentioned in articles considering its publication year (Year), scientific domain, impact of the publishing journal (Journal Rank), the article impact in citation count (Citation Rank), the number of software used within the article (# Software), and whether the software itself is available freely (Free) and open source (Open). OR=Odds Ratio; OR>1 indicates that a variable is associated with a higher amount of developer mentions; non significant values are indicated by the CI including 1, which is also highlighted by the use of brackets; the table is ordered by OR.

Variable	OR	95% CI	
Free	10.07	9.94	10.20
Business	2.72	2.37	3.12
Chemistry	1.95	1.90	2.01
Agricultural	1.42	1.41	1.43
Multidisciplinary	1.39	1.37	1.41
Mathematics	1.36	1.32	1.41
Veterinary	1.32	1.27	1.38
Immunology	1.25	1.24	1.26
Earth	1.23	1.05	1.43
Economics	1.15	(.79	1.69)
Biochemistry	1.10	1.09	1.11
# Software	1.09	1.09	1.09
Pharmacology	1.04	1.02	1.07
Environmental	1.04	1.01	1.06
Journal Rank	1.03	1.03	1.03
Psychology	1.01	(.98	1.06)
Citation Rank	.99	.99	.99
Year	.97	.97	.97
Engineering	.97	(.93	1.01)
Chemical	.96	(.93	1.00)
Social	.94	(.88	1.00)
Energy	.91	.84	.99
Computer	.82	.80	.84
Health	.80	.75	.85
Open	.78	.77	.78
Medicine	.76	.75	.77
Decision	.74	.63	.87
Neuroscience	.73	.72	.74
Arts	.71	.62	.82
Nursing	.63	.60	.67
Physics	.55	.53	.57
Dentistry	.46	.41	.51
Materials	.28	.27	.29

Table A10: Result of the logistic regression predicting if a formal citation was provided for individual software mentioned in articles considering its publication year (Year), scientific domain, impact of the publishing journal (Journal Rank), the article impact in citation count (Citation Rank), the number of software used within the article (# Software), and whether the software itself is available freely (Free) and open source (Open). OR=Odds Ratio; OR>1 indicates that a variable is associated with a higher amount of formal software citations; non significant values are indicated by the CI including 1, which is also highlighted by the use of brackets; the table is ordered by OR.

Variable	OR	95% CI	
Free	12.54	12.30	12.78
Energy	2.19	1.93	2.47
Chemical	1.66	1.56	1.76
Neuroscience	1.49	1.46	1.52
Multidisciplinary	1.30	1.27	1.33
Biochemistry	1.27	1.25	1.29
Medicine	1.20	1.18	1.22
Arts	1.19	(.97	1.46)
Computer	1.18	1.13	1.23
Veterinary	1.07	(1.00	1.14)
Pharmacology	1.06	1.03	1.10
Decision	1.04	(.80	1.36)
Physics	1.03	(.97	1.09)
Agricultural	1.02	(1.00	1.03)
Citation Rank	1.01	1.01	1.01
# Software	1.00	(1.00	1.01)
Year	.99	.99	.99
Journal Rank	.99	(.99	1.00)
Psychology	.88	.83	.93
Immunology	.87	.85	.88
Health	.86	.78	.94
Earth	.82	(.60	1.10)
Environmental	.74	.71	.77
Chemistry	.67	.64	.71
Social	.60	.54	.68
Mathematics	.56	.53	.60
Nursing	.53	.48	.59
Source	.53	.52	.54
Engineering	.50	.47	.54
Dentistry	.46	.39	.55
Business	.46	.33	.63
Economics	.36	.13	.98
Materials	.06	.05	.07

Table A11: Result of the logistic regression predicting if the URL was provided for individual software mentioned in articles considering its publication year (Year), scientific domain, impact of the publishing journal (Journal Rank), the article impact in citation count (Citation Rank), the number of software used within the article (# Software), and whether the software itself is available freely (Free) and open source (Open). OR=Odds Ratio; OR>1 indicates that a variable is associated with a higher amount of URL mentions; non significant values are indicated by the CI including 1, which is also highlighted by the use of brackets; the table is ordered by OR.

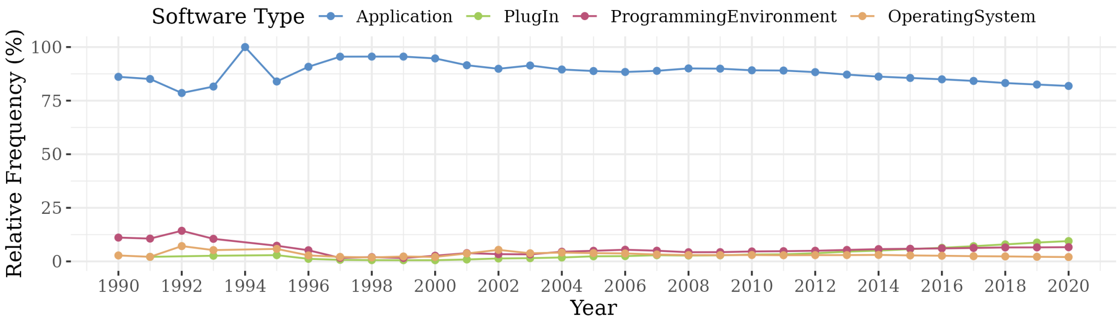


Figure A3: Relative amount of software types mentioned over time.

Variable	OR	95% CI	
# Software	1.51	1.45	1.58
not retracted	1.35	1.15	1.57
not retracted : #	.98	(.94	1.03)

Table A12: Result of the logistic regression predicting whether mentione software is freely available for retracted papers considering the number of software per article (# Software), the retraction status (not retracted), and their interaction. OR=Odds Ratio; OR>1 indicates that a variable is associated with a higher likelihood of the mention of freely available software; non significant values are indicated by the CI including 1, which is also highlighted by the use of brackets; the table is ordered by OR.

Variable	OR	95% CI	
not retracted	1.55	1.32	1.81
# Software	1.38	1.33	1.44
not retracted : #	.97	(.93	1.01)

Table A13: Result of the logistic regression predicting whether mentioned software is open source for retracted papers considering the number of software per article (# Software), the retraction status (not retracted), and their interaction. OR=Odds Ratio; OR>1 indicates that a variable is associated with a higher likelihood of the mention of open source software; non significant values are indicated by the CI including 1, which is also highlighted by the use of brackets; the table is ordered by OR.

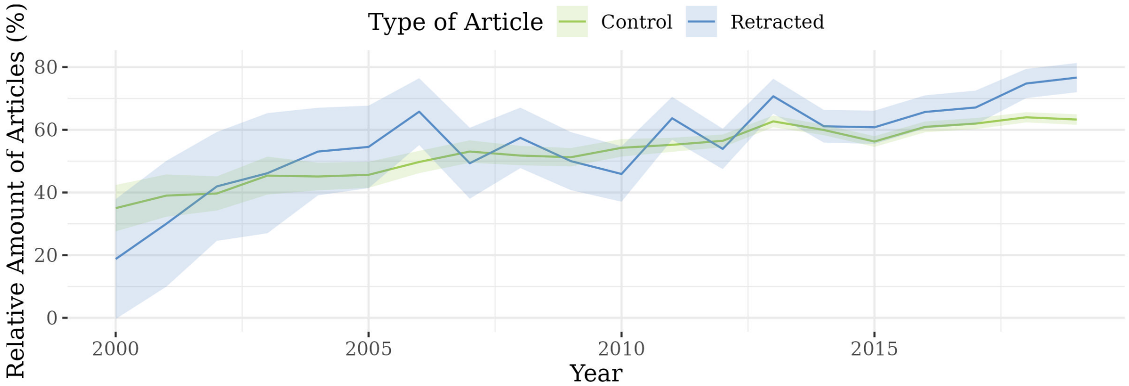


Figure A4: Relative number of articles containing at least one software over time for retracted and control articles. 95% CIs are indicated by lighter colored areas.

Top-level Reason	RW Reasons
Error	Error by Third Party; Error in Materials (General); Unreliable Image; Error by Journal/Publisher; Error in Image; Error in Text; Original Data not Provided; Results Not Reproducible; Concerns/Issues About Image; Concerns/Issues About Results; Unreliable Data; Error in Methods; Error in Analyses; Error in Results and/or Conclusions; Error in Data; Unreliable Results; Concerns/Issues About Data
Investigation	Investigation by ORI; Investigation by Third Party; Investigation by Company/Institution; Investigation by Journal/Publisher
Plagiarism	Plagiarism of Image; Plagiarism of Data; Plagiarism of Article; Plagiarism of Text; Euphemisms for Plagiarism
SelfPlagiarism	Duplication of Text; Euphemisms for Duplication; Duplication of Data; Duplication of Image; Duplication of Article
Misconduct	Misconduct by Company/Institution; Euphemisms for Misconduct; Manipulation of Results; Misconduct by Third Party; Falsification/Fabrication of Results; Falsification/Fabrication of Image; Manipulation of Images; Misconduct - Official Investigation/Finding; Falsification/Fabrication of Data; Misconduct by Author
PaperMill	Hoax Paper; Randomly Generated Content; Paper Mill
other	Sabotage of Materials; Updated to Correction; Complaints about Company/Institution; Breach of Policy by Third Party; No Further Action; Nonpayment of Fees/Refusal to Pay; Complaints about Third Party; Miscommunication by Company/Institution; Updated to Retraction; Not Presented at Conference; Miscommunication by Third Party; Taken via Peer Review; Contamination of Reagents; Miscommunication by Journal/Publisher; Salami Slicing; Civil Proceedings; Objections by Company/Institution; Ethical Violations by Third Party; Publishing Ban; Contamination of Materials (General); Criminal Proceedings; Error in Cell Lines/Tissues; Contamination of Cell Lines/Tissues; Bias Issues or Lack of Balance; Complaints about Author; Legal Reasons/Legal Threats; Miscommunication by Author; Doing the Right Thing; False Affiliation; Cites Retracted Work; Concerns/Issues about Third Party Involvement; Notice - Unable to Access via current resources; Informed/Patient Consent - None/Withdrawn; Temporary Removal; Lack of Approval from Company/Institution; Conflict of Interest; Lack of Approval from Third Party; Taken from Dissertation/Thesis; Notice - Lack of; Objections by Author(s); Withdrawn (out of date); Lack of Approval from Author; Rogue Editor; Lack of IRB/IACUC Approval; Copyright Claims; Withdrawn to Publish in Different Journal; False/Forged Authorship; Concerns/Issues about Referencing/Attributions; Duplicate Publication through Error by Journal/Publisher; Objections by Third Party; Ethical Violations by Author; Author Unresponsive; Concerns/Issues About Authorship; Retract and Replace; Upgrade/Update of Prior Notice; Fake Peer Review; Notice - No/Limited Information; Date of Retraction/Other Unknown; Breach of Policy by Author; Withdrawal; Notice - Limited or No Information

Table A14: Summary of top level retraction reasons based on the fine-grained retraction reasons obtained from RW.

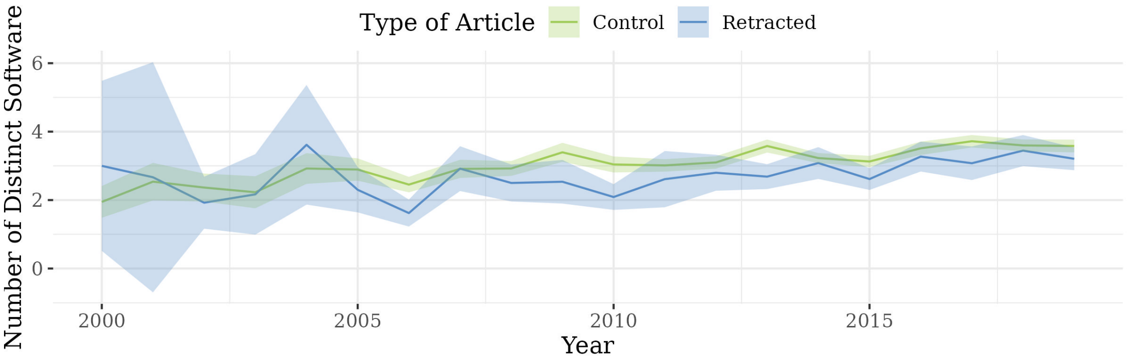


Figure A5: Mean number of distinct software mentioned in articles that contain at least one software, depicted over time for retracted and control articles. 95% CIs are indicated by lighter colored areas.

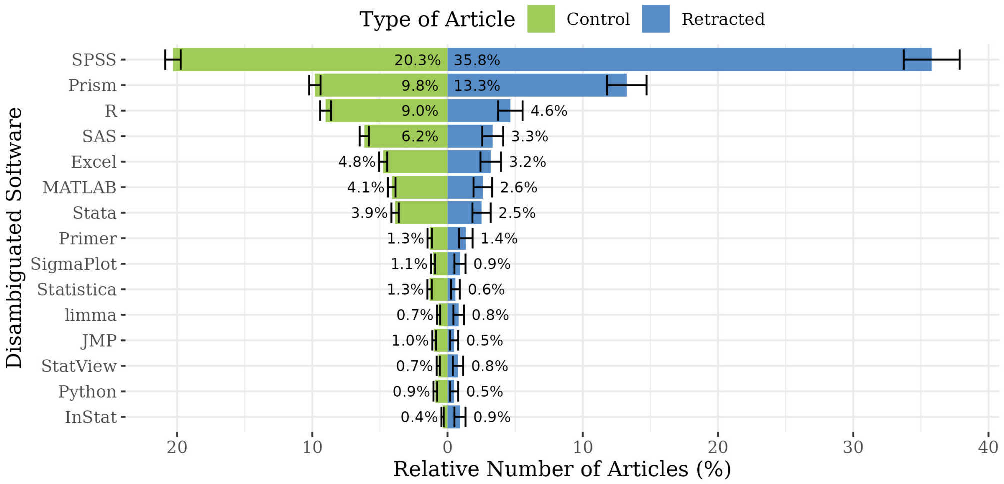


Figure A6: Proportion of retracted and control articles mentioning software out of the top 15 most used statistical software. Error bars indicate 95% CIs.

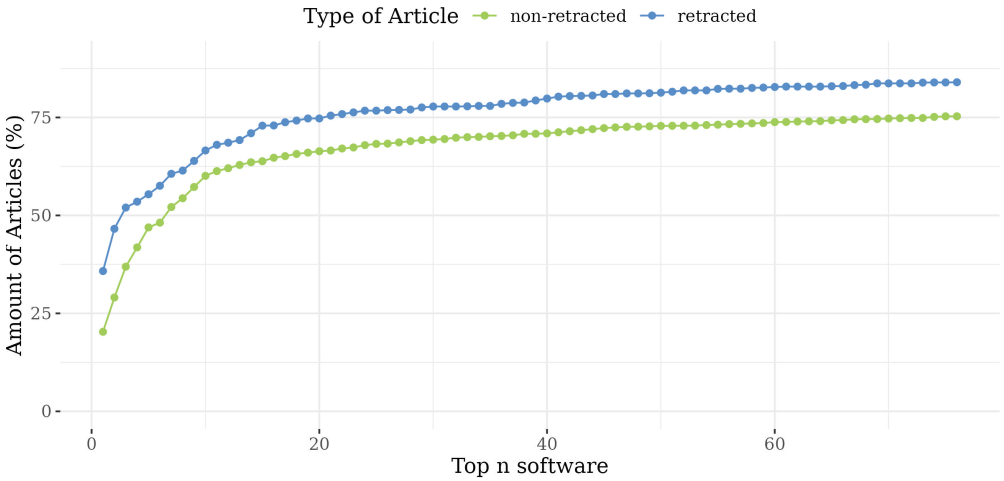


Figure A7: Relative amount of articles mentioning at least one of the top n software out of all articles that mention software.

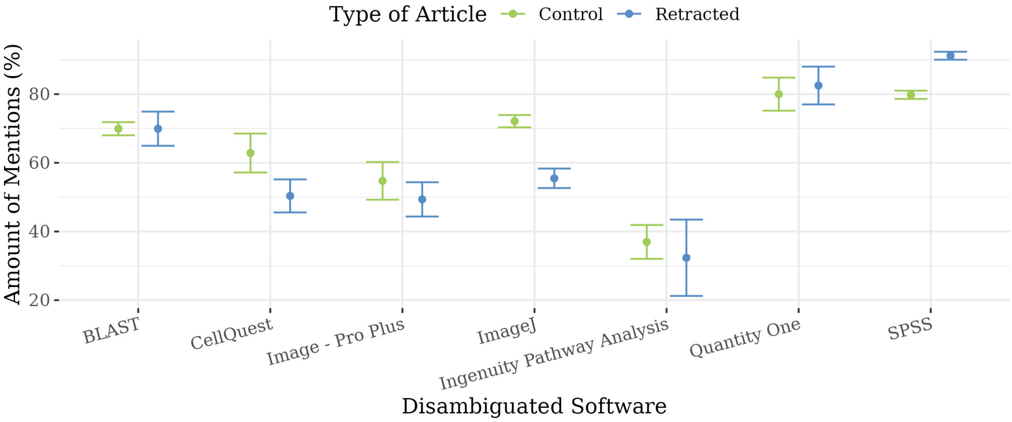


Figure A8: Relative amount of articles mentioning a specific software and referring to it by its most commonly used name.

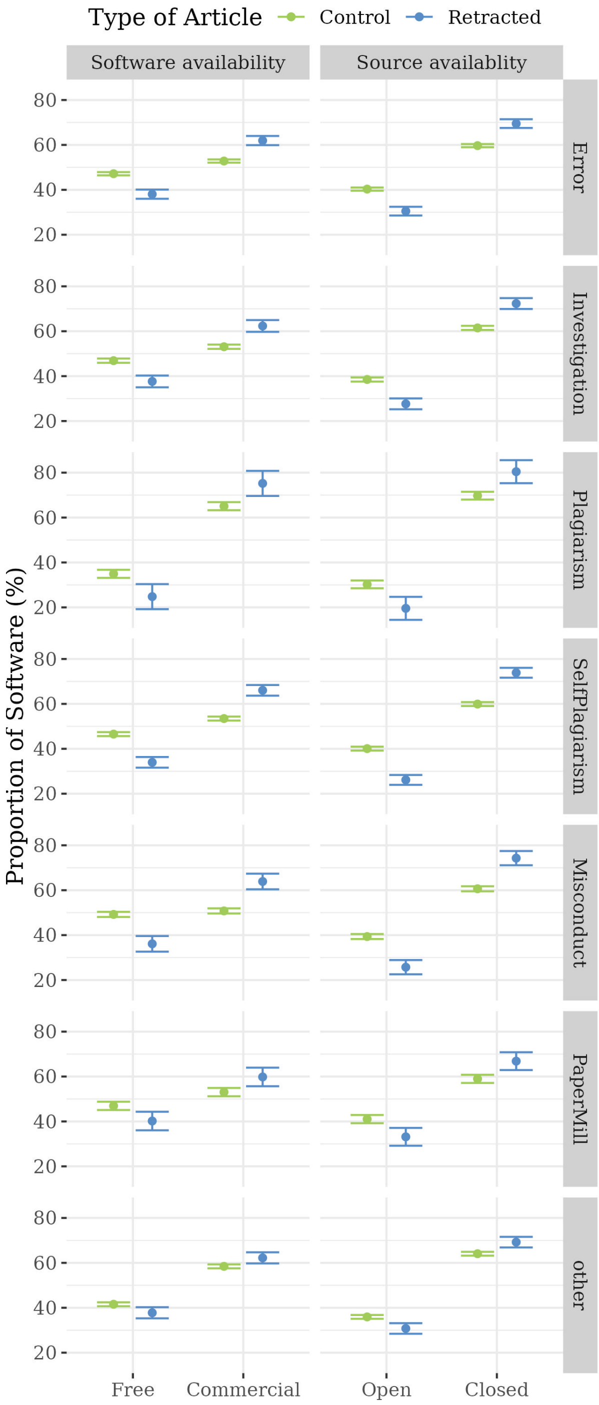


Figure A9: Proportion of free or open source software across retracted and control articles per retraction reason. A separate control set is constructed for each retraction reasons by selecting the ten corresponding articles for each retracted paper. Error bars indicate 95% CIs.

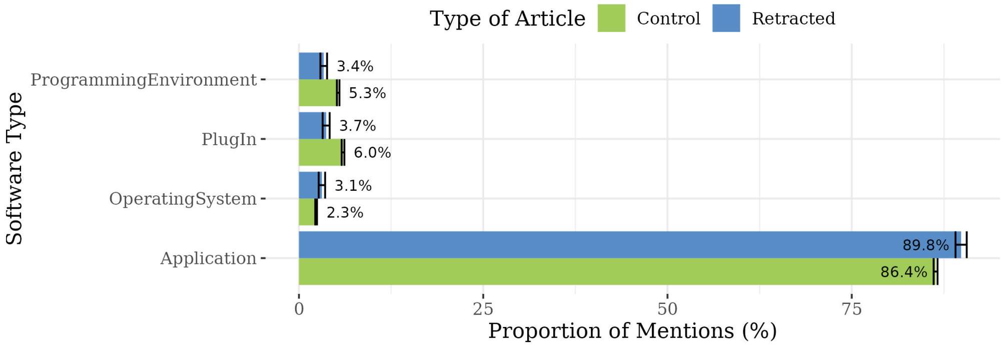


Figure A10: Proportion of software types on overall software mentions between retracted and control articles.

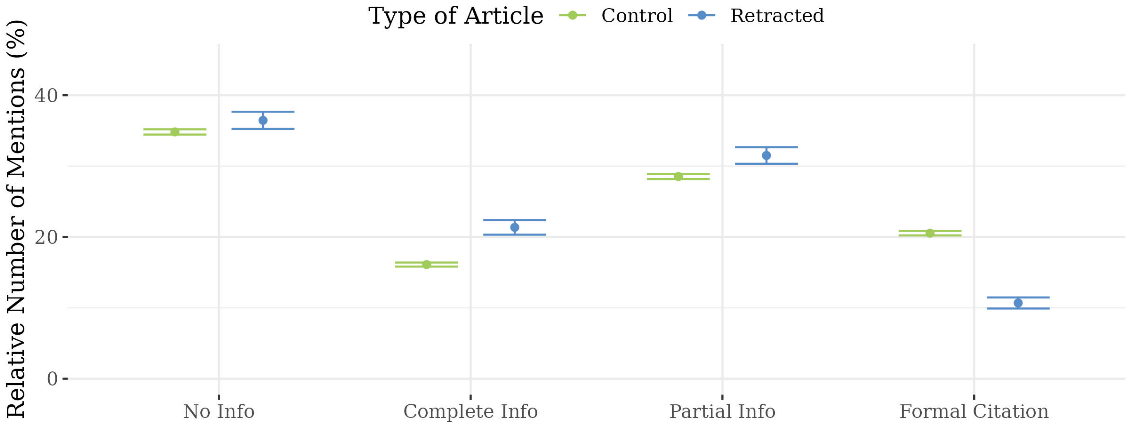


Figure A11: Proportion of software mentions across different levels of citation completeness, separated by retracted and control articles. No Info: Neither the version, nor the developer of a software are provided; Incomplete Info: Either version or developer is provided; Informal Citation: Version and developer are provided; Formal citation: software mention is accompanied by bibliographic citation. Error bars indicate 95% CIs.

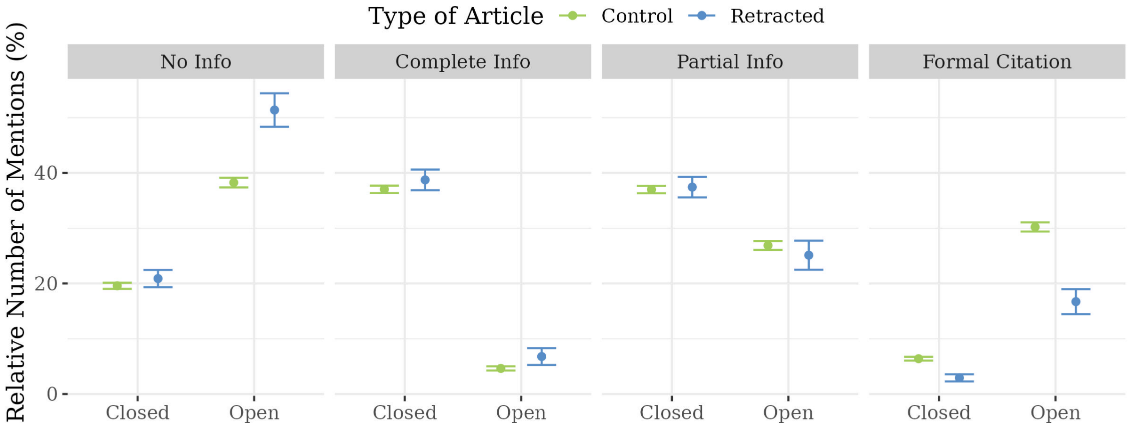


Figure A12: Proportion of software mentions across different levels of citation completeness, separated by retracted and control articles and between open source and closed source software. No Info: Neither the version, nor the developer of a software are provided; Incomplete Info: Either version or developer is provided; Informal Citation: Version and developer are provided; Formal citation: software mention is accompanied by bibliographic citation. Error bars indicate 95% CIs.

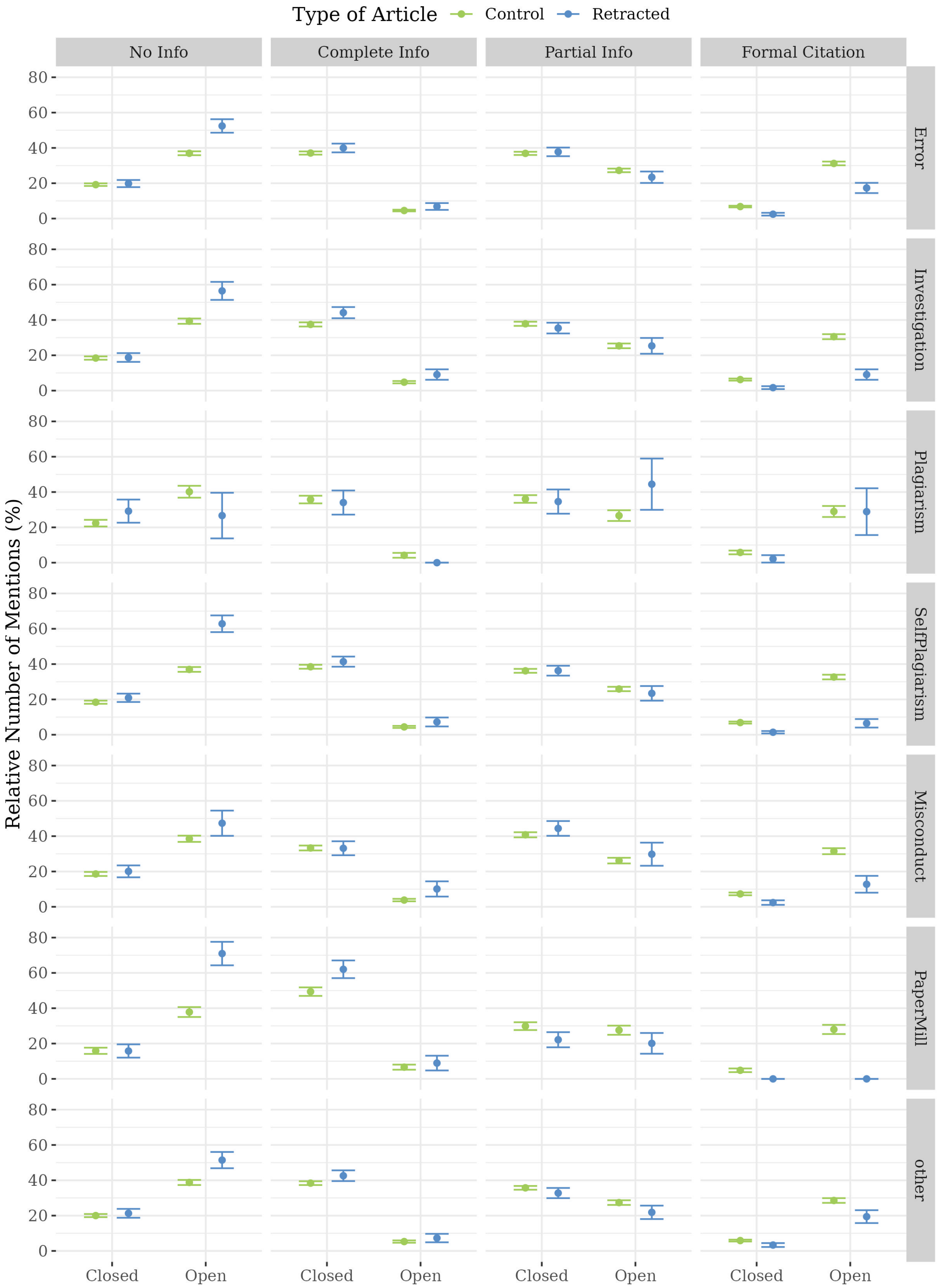


Figure A13: Proportion of software mentions across different levels of citation completeness per retraction reason, separated by retracted and control articles, considering whether software is commercial or freely available.. No Info: Neither the version, nor the developer of a software is provided; Incomplete Info: Either version or developer are provided; Informal Citation: Version and developer are provided; Formal citation: software mention is accompanied by bibliographic citation (independent from any associated information). Error bars indicate 95% CIs.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die eingereichte Promotionsschrift selbstständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Die vorgelegte Promotionsschrift wurde bisher weder im Ausland noch im Inland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt.

____
Ort, Datum

____
David Schindler