Automatic Classification of Purpose of software usages in scholarly articles

Master Thesis

Fakultät für Informatik und Elektrotechnik
Institut für Nachrichtentechnik

Von Bekalue Tadesse

Automatic Classification of Purpose of software usages in scholarly articles

Master Thesis

Universität Rostock

Fakultät für Informatik und Elektrotechnik
Institut für Nachrichtentechnik

Kobro, Bekalue Tadesse

Matriculation Number: 218205220

Supervisors	:	Dr.-Ing. Frank Krüger & David Schindler
Professor	:	Prof. Dr.-Ing. Sascha Spors
Date	:	11. April 2022

SUPERVISORS:
Dr.-Ing. Frank Krüger & David Schindler

DECLARATION

I hereby confirm that I have written the present work independently and have not used any other sources than those given in the bibliography.

All passages that are taken verbatim or correspondingly from published or not yet published sources are marked as such.

The drawings or images in this work were created by myself or provided with a corresponding source reference.

This work has not yet been submitted to any other examination authority in the same or a similar form.

Rostock, 11. April 2022

Kobro, Bekalue Tadesse

ABSTRACT

These days scientific research is increasingly dependent on software as many scientists not only use existing software, but also create one to use in their research process. This strong reliance on software, besides its benefits, has also created issues regarding reliability and quality of research results from a software. Some scientists were forced to retract their scientific work upon a retrospective discovery of an error in a software they used, which rendered their research worthless. In addition, use of software for scientific research entailed difficulty of reproducing research results mainly because of poor software citation practices.

This thesis contributes to an effort to alleviate the above problems by extracting useful information about software tools particularly the purpose of use of software in a scientific paper. Automatic extraction of purpose of software usage creates an opportunity to identify a set of software tools used for the same purpose in a given context of scientific work. This in turn helps to compare results obtained among similar software tools enabling evaluation for quality and reliability of research software. In addition, automatic extraction of software usage purposes enables to semantically browse related research articles based on their use of a specific software for the same purpose.

In this thesis, SoMeSci dataset has been annotated with labels that indicate the purpose of software use which have been identified by extensive research of literature and software ontologies. Further, Sci-BERT classifier model has been selected and trained on SoMeSci dataset after exploring various kinds of models that suit for extraction of software usage purposes. The classifier model has been further optimized by considering various training scenarios such as more context information, inclusion or exclusion of some parts of SoMeSci dataset, simplifying classifier model with fewer classifier modules. Overall, results of model training, indicates the use of classifier models pretrained on scientific corpus, such as Sci-BERT and Bio-BERT, enabled automatic classification of software usage purposes with a reasonable performance despite a limited amount of labeled dataset. Further more, it was observed that consideration of broader context, such as 2 adjacent sentences, improved classifier model's performance.

ACKNOWLEDGMENTS

First and fore most I would like to praise God, the almighty, for His continuous provision of grace and unfailing mercy through out my study and life overall.

Secondly, I would like to express my sincere gratitude towards my thesis supervisors Dr. Ing. Frank Krüger and David Schindler for offering me an opportunity to work under their supervision. I appreciate every precious time, effort and resources they have invested in me. They have been extremely helpful through out my thesis work and I could not imagine doing this project without their immense support and direction.

I would also like to thank my parents, Tadesse Kobro and Debritu Berhanu, for their continuous support and sacrifices. My gratitude further goes to my siblings: Bamlak, Japheth, Hannah and Jerusalem for their understanding and love. I also feel a deep sense of gratitude towards my uncle Netsanet Berhanu for being a continuous source of wisdom and inspiration.

Last but not least, I owe a heartly gratitude to my pastor and friend, Kevan Smith and his wife Pricilla for their empathy, continuous prayers and support. I would love to appreciate pastor Kevan for his profound wisdom and counseling.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

URL Uniform Resource Locator

MRI Magnetic Resonance Imaging

CT Computed Tomography

EHT Event Horizon Telescope

CERN European Council for Nuclear Research

LHC Large Hydron Collider

ReSA Research software Alliance

NTBT Nuclear Test Ban Treaties

SWO SoftWare Ontology

SoMeSci Software Mentions in Scientific Articles

OLS Ontology Search

Bi-LSTM-CRF Bi-directional Long Short Term Memory-Conditional Random Field

BERT Bidirectional Encoder Representations from Transformers

Bio-BERT Biomedical-BERT

Sci-BERT Scientific-BERT

DMRI Diffusion Magnetic Resonance Imaging

Geant4 GEometry ANd Tracking

SPARQL SPARQL Protocol and RDF Query Language

IOB Inside–Outside–Beginning

NASA National Aeronautics and Space Administration

CAD Computer Aided Design

PMC PubMed Central

CSV Comma Separated Value

CGI Common Gateway Interface

NER Named Entity Recognition

NLP Natural Language Processing

POS Part-of-Speech

HMM Hidden Markov Model

MEMM Maximum Entropy Markov Model

CRF Conditional Random Field

LSTM Long Short-Term Memory

RNN Recurrent Neural Network

Bi-LSTM Bi-Long Short-Term Memory

DNA Deoxyribonucleic

MST Mission Simulation Toolkit

GPU Graphics Processing Unit

MLM Masked Language Model

Part I

THESIS

1 INTRODUCTION

Today's scientific research relies on a use as well as production of various types of research artifacts, a.k.a. research resources, ranging from digital to physical [61]. One key digital research artifact, broadly used in a scientific investigations, is software. Many scientists not only *use* already existing software for various purposes during their research, but also *create* a new software as part of their research work [22, 24].

Quality of a research result directly relies on quality of software which in turn depends on how the software is developed as well as used in a research [24]. Typical problems associated with the development and use software in a research are inability to reproduce research results, lack of proper citation for software tools and most importantly the issue of reliability of research results obtained from a software [6, 59, 63].

Automatic extraction of detailed information about a software tools helps to solve these problems. For instance, information about a software such as Uniform Resource Locator (URL), name of developer, version, release, extension, etc. can be used to uniquely identify a software tool used in a given research. Uniquely identifying a software in turn helps to improve reproducibility and transparency of scientific work by avoiding ambiguity regarding what software, along with its specification, has been used to produce a given result. In addition, such information can also be used for automatic citation of software which is also advantageous to attribute proper credits to the developers of a software [32].

Unlike URL, version, name of software, etc. which are often explicitly mentioned in scientific papers, information about software hidden in a textual description such as the purpose of use of a given software can also be extracted automatically using machine learning techniques. Purpose of a software is reason a software is created for. For example, purpose of system software such as operating systems is to manage hardware resources, to provide a platform to run application software, etc. [87].

Extraction of purpose of use of software can be used to discover a set of similar software tools which have been used for the same purpose in a specific context of text. This creates an opportunity to compare results obtained from different software tools, to test reliability of a research result obtained from software tool and contributes to identification of any potential error hidden in a software.

In addition, knowledge about software use and purpose of use in the literature, supports semantic analysis and retrieval of scientific publications based on use of particular software [61]. On the other hand, the same knowledge can be used to suggest list of software tools that might suit for a given task in a research.

Various manual and rule based techniques has been attempted in the past to extract information about software. However, machine/deep learning based techniques have not been exploited to their potential until very recently. The main reason was lack of training data which can support training of a classifier for software information extraction [59].

Producing reliable ground truth data could be accomplished by crowd sourcing for general domains but it is expensive particularly for domain-specific and scientific publications as it requires expert of the domain knowledge [7].

Fortunately, identification of software mentions from scientific articles has drawn more attention over the past years and now various labelled datasets, such as BioNerDs [14] , SoftCite [13], are available. Recently a more comprehensive dataset, SoMeSci, has also been published. SoMeSci contains high quality manually annotated datasets that cover broader range of information about software paving a way for a use of machine learning based approach for the automatic extraction of information about software [59].

1.1 CONTRIBUTIONS

This thesis applies SCI-BERT model, a transformer based neural network architecture, to automatically classify the purpose of software usage from a text in scientific papers using a labeled dataset, SOMESCI.

To accomplish this, first review of literature has been carried out about the role of software in a research. Second, possible list of software usage purposes have been identified via extensive analysis of literature and other sources like software ontologies and repositories like Sci-Crunch. Next, from already existing annotations about a software in the SOMESCI dataset, software usage mentions have been extended with software purpose labels using BRAT annotation tool. After that, extended version of SOMESCI dataset has been cleaned, analyzed, transformed to Inside–Outside–Beginning (IOB) format, and used in the classifier model. Finally, the classifier model has been optimized and evaluated by training in different scenarios.

1.2 OVERVIEW OF THE REPORT

Chapter 1 makes a soft introduction about why it is important to automatically extract information about a software such as purpose of use of software.

Chapter 2 focuses on highlighting the role of software in a research to indicate driving information about software from scientific publications is an important task.

Chapter 3 focuses to identify possible types of software usage purposes from literature and software ontology. This is an important step taken to extend software usage statements in the SOMESCI dataset with software purpose annotations.

Chapter 4 explains how SoMeSci dataset has been extended with software purpose annotations, annotation tool used, and the annotation process. In addition explains about data pre-processing, transformation to suitable format and splitting for classification purpose. At the end, results of analysis of the extended SoMeSci dataset has also been presented.

Chapter 5 explores suitable approaches and models for software purpose classifications. Then training, optimization and evaluation of the selected classifier model has been presented in chapter 6. Finally, results and observations are presented in chapter 7.

2

THE ROLE OF SOFTWARE IN SCIENTIFIC INVESTIGATION

Nowadays scientific research is unthinkable without a use of software and investigations in various areas of science are becoming increasingly reliant on software tools [22, 24, 30, 64].

A software is very important asset for building a scientific knowledge and more discoveries in a research are made possible than ever by a use of software tools that automate processing of huge amount of data [30]. Typically a software is used in a research for data processing tasks such as data analysis, modeling, simulation, control processes, knowledge dissemination, etc. [24, 52].

Software tools can be categorized into scientific software and regular software based on their role in scientific research. Scientific software is typically developed to assist exploration of a given research problem and it is typically not easy to specify requirements upfront. In contrast, specification of requirements for regular software such as accounting software is often known in advance. Broadly, any software can also be regarded as scientific software as long as it is used at least once in a formal research with proper citation. Throughout this thesis scientific software refers to tools that are directly used for exploration of research problems with a proper citation based on the definition [22, 24].

In modern research, a scientific software is as important as any lab-equipment [88]. However, the development of scientific software is fundamentally different from regular software and usually requires active engagement of experts who have profound domain knowledge. For instance, development of a software tool for transformation of Deoxyribonucleic (DNA) into protein crystals requires expert domain knowledge compared to development of accounting software or human resource management software tool which can be done by wide range of software developer community with basic understanding of the domain [62, 88]. Due to this, an increasing number of scientists are developing a software as part of their research work or directly taking part in the development process of a research software [30, 31].

According to surveys conducted in the UK and USA, 2008 and 2017 respectively, most scientists agree that software plays an important role in their research work [26, 48]. Participants of the survey, in UK, were 2000 researchers working in various areas of science in roles ranging from student to senior academic staff whereas participants of the survey, in USA, were members of the US National Postdoctoral Association.

The results of UK survey indicate that [26]:

- 38% of researchers spend at least 20% of their time developing a software.

- Almost half of scientists spend more time creating software as part of their research work than five years ago .
- Over 50% of survey respondents reported that they develop their own software.
- Nearly 70% claim their research-work directly depends on use of a software and
- Over 90% of scientists say software is important for their research.

The results of US survey indicate that [48]:

- Over 90% of scientists use software
- 63% of respondents said their research is impossible with out the use of software.
- 31% of scientists stated they could do their work without using a software but more effort would require.
- Only 6% of survey respondents say that there is would be no significant difference in their task if they do not have to use software.

Overall, results from the two surveys clearly indicate that software is pervasive in scientific investigations and many researchers use as well as develop a software for their research.

Even though software plays an important role in a modern research, usually the contributions of software is understated. This can be seen from the poor citation practice of software in research papers across several fields of research [52, 59, 91]. The Research software Alliance (ReSA)[1] has gathered literature that demonstrates the roles of software in research at the Zetoro group library[2] in an effort to increase acknowledgment of the roles of scientific software in research.

The next section presents more details about the role of software: in general, in specific domains, and in research breakthroughs.

2.1 GENERAL ROLES

Software is playing crucial roles in a research and making a shift in a research culture in terms of enabling automation of analysis pipelines, creation of new ways of analysis via computational models, supporting sophisticated analysis of large volume of data, documentation of a research, etc. [29].

Some of the most general roles of a software in research:

- A software dictates the quality of a research outcome [24]. Outcome of a research becomes unreliable or even useless if there is an error in the software [63]. For example, several scientists retracted their scientific publications up on a retrospective discovery of a bug in their software [45, 46, 88]. A more palpable failure of a research ambition due to an error in the software, for instance, is the failure of Ariane rocket in 1996 [19].

1 https://www.researchsoft.org/
2 https://www.zotero.org/groups/2400609/resa/library

- Software helps to explore and understand a research problem [24].
- Results from a scientific software is presented as an evidence to support a research conclusion [31].
- A software also helps to document a research process and to validate results of a given research [29]. Executable cells in a Jupyter notebook is one real world example where a software can be used to document a research result.
- Software allows experiments to be made beyond constrains of the physical world. This is because experiments that run on a computer are not limited by processes that occur in nature but only by the laws embedded in the computer code [89].

2.2 DOMAIN SPECIFIC EXAMPLES

Software is being extensively used for a research in various areas of science such as physics, chemistry, space science, life science and so on.

The physics research facility, the Large Hydron Collider (LHC) at European Council for Nuclear Research (CERN), for instance uses a software with more than 5 million lines of code which is used for processing of terabytes of data generated from experiments [64]. Being a pioneer research institute, CERN has developed its own open source software frame work to provide large scale repository services to manage digital assets. An example of such software is *Invenio*[3] which is typically used for management of scholarly digital contents [86].

In a nuclear research, a software is being developed increasingly to be used for experiments [90]. For example, testing a modification in a nuclear weapon can not be done on a field, but instead a software that simulate the impact of modification is usually used [31]. This is because testing of a nuclear weapon on a field is banned by regulations like nuclear test ban treaties Nuclear Test Ban Treaties (NTBT) in addition to the potential disaster that testing a nuclear weapon poses to the environment and life [70]. GEometry ANd Tracking (GEANT4)[4] is example of software tool that is typically used in nuclear research to simulate the passage of particles through a matter [2].

In chemistry research, a software can be used to model and simulate chemical processes that are challenging, too complex or expensive to conduct in reality. Karplus and Levitt used computer simulations for their joint-research "the development of multiscale models for complex chemical systems" and won a Nobel prize in 2013 for their work [5, 64]. *Scigress*[5] is an example of software that is used for research in chemistry for 3-dimensional visualization of chemical formulas to avoid wrong interpretation and misunderstanding [43, 82].

In a climate and environmental studies, software is used to make predictions about climate changes. For example historical temperature data can be integrated to make

3 https://github.com/inveniosoftware/invenio
4 ssh://git@gitlab.cern.ch:7999/geant4/geant4.git
5 https://www.fqs.pl/en/chemistry/products/scigress

predictions about future temperature variations [64].

In a space science, space probes heavily rely on software. In this case a software navigates space crafts to other planets, processes and transmits scientific data back to Earth for further processing, helps researchers interpret results, etc.[39]. At National Aeronautics and Space Administration (NASA), Mission Simulation Toolkit (MST) simulation framework, for instance, is used to support development of autonomous planetary vehicles. The MST [6] provides a software environment which includes simulated robotic platforms and sensors.

In medical research and diagnosis, imaging software plays a critical role to assist medical researchers, for instance, for early isolation of cancer cells. The main reason for low chance of survival from cancer is mainly due to late detection of cancer cells in the body. This makes a diagnosis of cancer to be a time critical task and early identification of cancer implies curability of a disease and a higher chance of survival [67]. Especially on the early stages, it is not straight forward to determine which cells are likely to develop a cancer. For this reason, medical scientists use different types of software to identify cancer cell or to decide weather a tumor is malignant or not. Using a software, they could perform various kinds of analysis and processing on imageries obtained from scans such as Magnetic Resonance Imaging (MRI) or Computed Tomography (CT) Scan [4]. An example of software that is used for cancer imaging research is Diffusion Magnetic Resonance Imaging (DMRI). Such software is extensively used by many researchers, more than 75,000 downloads every year [50]. Therefore, it is clear that software plays a critical role in medicine, to diagnose diseases and ultimately to save life.

Software plays an important role in power system planning and operation as well. One of the major activities in power system operation is contingency analysis. During contingency analysis, engineers determine violations of power grid operation conditions, such as overloading, which might occur when outage of a transmission line or a power generation unit happens. Contingency analysis helps to understand power system behavior after outages and gives an opportunity to take preventative actions [47]. Power grids are extremely complex and such kind of analysis tasks are unimaginable with out a use of software. An example of software that is used to perform contingency analysis in the power system operation is *Power World* software [7].

Though it is not possible to mention the role and use of software across all areas of science and research, the above examples serve to be a good sample to see how ubiquitous the impact of software is almost in all research areas.

2.3 THE ROLE OF SOFTWARE IN RESEARCH BREAKTHROUGHS

The impact of software is more pronounced and easy to observe when scientists achieve ground breaking results. The use of software enabled scientists to produce better sci-

6 https://software.nasa.gov/software/ARC-14932-1
7 https://www.powerworld.com/

entific discoveries and achieve research breakthroughs [22].

One good example of research breakthrough made, because of use of software in a scientific investigation, is creation of the very first visual representation of a black hole using an open source software NumFOCUS[8] [10].

To observe a black hole that is 55 million light years away, it would have required to build a huge telescope of size of planet earth. But instead of building one giant telescope, which is not possible any way, hundreds of scientists spent decades of years creating a global network of telescopes, known as EHT [10, 12], synchronized precisely using atomic clocks.

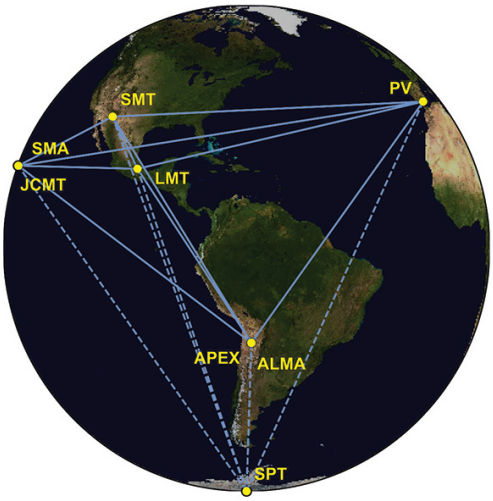


Figure 2.1: Location of EHT Satellites around the world [10]

The EHT gathered a huge amount of data for years. However there was a lot of noise in the collected data because the EHT was a network non-similar telescopes. In addition, the radio signals were coming through attenuated due to atmospheric effects like water vapor, clouds, turbulence, etc. [51].

Therefore the scientists had to use various algorithms and data analysis pipelines. The resulting image from various data processing was compared to ensure the integrity of the result. This huge scientific breakthrough in a space research, can be attributed to the use of powerful data processing software.

8 https://numfocus.org/

3

SOFTWARE USAGE PURPOSES

In scientific investigations broad range of software is being employed for various purposes. In terms of size, software ranges from simple script to extremely complex with millions of lines of code. In terms of task, a software can be used for execution of rudimentary tasks to computation of extremely complex ones. Typical examples of purpose of software use for scientific investigation are simulation, modeling, data analysis, etc. [22].

To be able to automatically classify software usage purposes, a classifier algorithm has to be trained on a manually annotated dateset that indicate software usage purpose. The SoMeSci dataset already has annotations about type of software, type of software mentions, etc. Therefore, the main goal of enumerating a comprehensive list of software usage purposes is to extend and enhance existing annotations in the SoMeSci dataset with software purpose labels, so that the dataset suits training of software purpose classification.

To enumerate possible software usage purposes, manual analysis of literature in the SoMeSci dataset has been carried out. However, manual analysis of all literature is tedious and intractable task and consequently other resources such as software ontologies and repositories like Sci-Cruch have been analyzed to list down categories of software purposes. The process work flow, shown on the figure 3.1., below describes resources that have been analyzed to enumerate categories of software usage purposes.

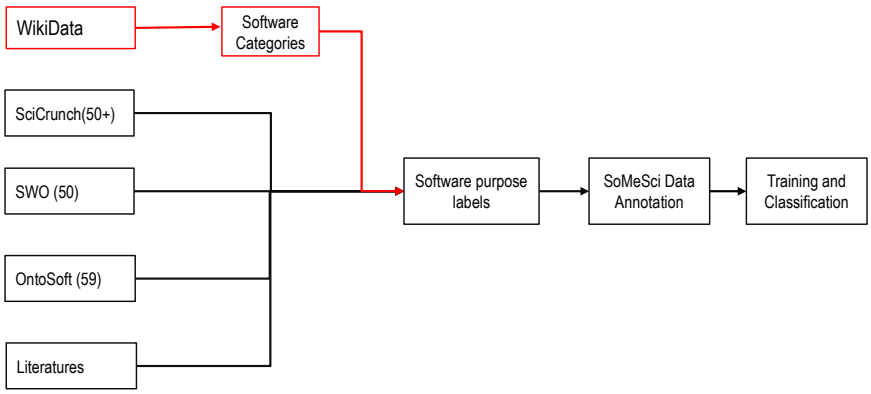


Figure 3.1: Workflow indicates a sequence of steps followed to extract classes of potential software usage purposes to label dataset that will be used to train a classifier.

After identifying a list of potential software usage purposes, the list has been consolidated further to narrow down the list to a more comprehensive list of software usage purposes for convenience during annotation of dataset. This section elaborates the analysis procedure of a list of resources mentioned above to identify possible software usage purposes.

3.1 ANALYSIS OF LITERATURE

In a research, scientists follow scientific method to discover knowledge. Typically, scientists begin with a question and attempt to answer questions through a research and propose hypothetical answers for their questions. Then, they test the proposed hypothesis by conducting various experiments. Although all scientists do not follow the exact same procedure, the over all idea remains the same [80]. This is where a software use comes into play, aid scientists during their experiment.

Therefore, the analysis of literature has been carried out aimed to find out for what purpose scientists have used a given software by asking a rhetorical question *"for what purpose are scientists trying to use the software ?"* in a given context.

Accordingly, key words that reflect potential software usage purposes have been identified from the literature are listed on the following table 3.1 below:

Table 3.1: Key words that have been extracted from analysis of literature that potentially indicate purpose of software use.

- Comparison of experimental groups	- Statistical analysis
- Quantification	- Data analysis
- Measurements	- Densitometric analysis
- Analysis	- Voxel-based Analysis
- Mapping	- Cross-sectional ROI analysis
- Correction of mapping	- Gene analysis
- Generate scaffolds	- Gene assembling
- Generate trees	- Construct contigs
- Search sequences	- Fill gaps
- Map	- Generate assembly
- Predict gene structure	- Calculate
- Align gene	- Draw heat map
- Filter	- Validate
- Evaluate	- Annotation
- Select	- Fit or train a model
- Optimise	- Sketch
- Classify	- Identify

The list of key words in the above table is used to delineate possible software usage purposes, however, it is intractable to enumerate all possible software purposes by manually reading through large number of publications. To augment results obtained from the analysis of literature, various software ontologies and repositories like, SciCrunch, have been analyzed as presented in the following section.

3.2 ANALYSIS OF SOFTWARE ONTOLOGIES

Ontologies are controlled vocabularies that provide formal naming and definition of properties and relation between concepts, entities, data etc. Ontologies are specialized to a specific subject matter and every academic discipline creates ontologies to organize data into useful knowledge [79].

Effective knowledge representation begins with analysis of ontologies with in the domain of interest [9]. Accordingly, analysis of software ontologies have been done to find out possible software usage purposes. The software ontologies, that have been analyzed on this project are: WikiData[1], SoftWare Ontology (SWO)[2], and Ontosoft[3].

3.2.1 *Wikidata*

Wikidata is a multilingual knowledge graph that is curated collaboratively by a Wikimedia community and serves as a freely available common source of structured data for everyone [84, 85].

Wikidata was created by Wikimedia foundation mainly to store metadata that can be used for other Wikimedia projects such as Wikipedia. Interestingly, wikidata is allowed to contain inconsistent and contradicting facts in order to embrace the diversity of knowledge about a given entity [66].

Although wikidata has a tremendous amount of data in it, information that would indicate software usage purposes was not found in it (last checked 25-12-2022). Rather information about software categories was found. Even though wikidata has no information about software purposes, it was desired to proceed with software category information because overall there has been a challenge to find other resources that help to list down software usage purposes. The goal is to list down software usage purposes from software categories, following an indirect approach, assuming each software category has essentially a software purpose associated to it.

To list down software categories from wikidata, a SPARQL query terminal has been used. The query has been formulated to retrieve software categories in a format that supports network analysis. As a result, 460 software categories have been retrieved from wikidata on the date 25-11-2021. The SPARQL query used to retrieve software categories have been listed under *Appendix-A*.

Since the 460 list of software categories is large and does not imply any software purpose, it was desired to carry out network analysis to reduce the software category list by creating potential relation between software categories. To find out potential relation between these categories and to select more general software categories, a network analysis has been done using Gephi software[4] (version 0.9.2) (RRID:SCR_004293)

1 https://wikidata.org/
2 https://www.ebi.ac.uk/ols/ontologies/swo
3 https://www.ontosoft.org/
4 http://gephi.org/

. Using Gephi, clustering of related software categories and filtering has been made to identify a more generalized software categories.

The procedure for network analysis has been described as follows:

- First query result from the SPARQL terminal of wikidata has been downloaded in a Comma Separated Value (CSV) file format.

- Then, the CSV file has been opened with Gephi software as *"undirected graph"*. This renders a network graph with overlapping nodes and edges which is difficult to observe the relation between software categories of the network.

- To unravel the overlapping nodes for visibility, the lay-out of the graph is then changed to *"Fruchterman Reingold"*. This creates automatically ordered network of software categories ordred in a circular manner as shown in the figure 3.2 below.

- To find out software categories that are related to each other, clustering has been done using *"Modularity"* from the list of statistical tools in the Gephi software. This creates an option *"Modularity class"* in the software which helps to partition software categories based on realtion to each other.

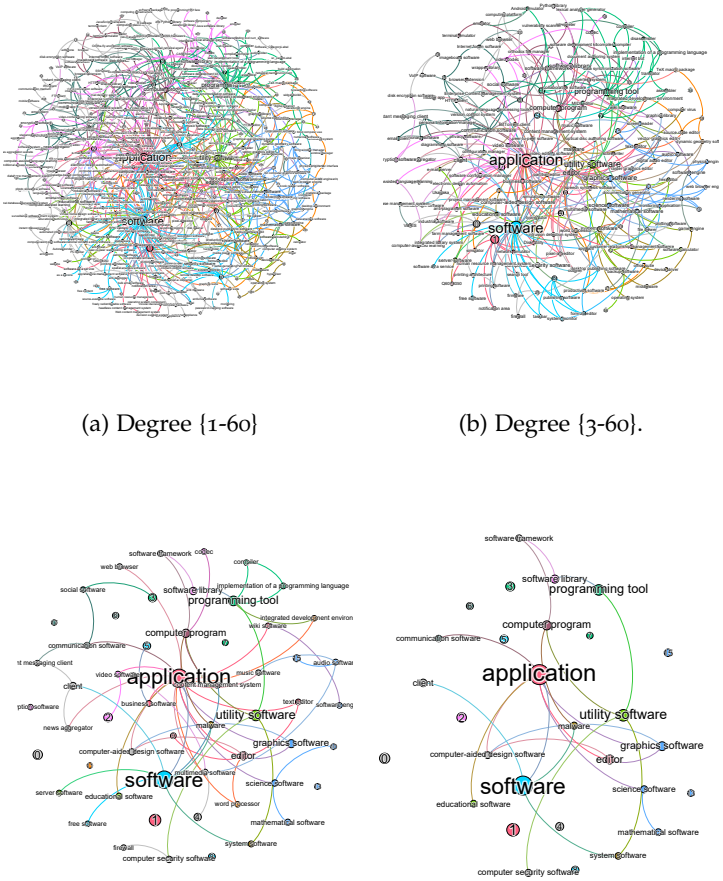
- Then to adjust size of nodes based on importance, node size ranking has been done with a *"Degree"* parameter with {minimum, maximum} size of {20, 80} respectively. These numbers have been chosen after manually tweaking numbers in the Gephi spftware for better visibility of important nodes in the network. Intuitively, big node size corresponds to more general software category.

- After identifying important nodes, filter tool *"Degree range"* in the Gephi software has been used to filter software categories in the based on prominence of nodes in the network. The filter generates values in the range of {1, 60} where the maximum value indicates the most prominent node (most general software category). The numbers in the range of {1, 60} have been generated automatically by the filter. By varying the degree range, various level of detail of the software category network can be obtained as shown on figures 3.2 (a) through (d).

According to the network analysis, major types of software categories identified are listed on the table 3.2 below.

According to a manual analysis of wikidata, software categories listed in table above are related to each other as well. Mathematical software, for instance, is subclass of science software and science software is subclass of application software.

By further analyzing the relation between the software categories in the table above, overall the three main types of software categories identified are: Application software, System software and Software Component. A simplified version of hierarchical relation between these categories is depicted on figure 3.3. Overall, from the analysis of wikidata some software usage purposes that can be inferred from the keyword in the table 3.2 and figure 3.3. are programming, analysis, design, editing, graphical modeling, etc.



(a) Degree {1-60}

(b) Degree {3-60}.

(c) Degree {5-60}

(d) Degree {7-60}

Figure 3.2: Network analysis result indicates the relation between each software category. The Degree filter with a range of 1 to 60, Large value correspondences more general software category.

Table 3.2: Major software categories identified with network analysis of data retrieved from Wikidata (on 25-11-2021) with SPARQL Query.

- Application softwares	- Utility software
- Computer security software	- System software
- Measurements	- Densitometric analysis
- Client	- Programming tool
- Software Library	- Software framework
- Editor	- Science software
- Graphics software	- Computer aided design software
- Mathematical software	- Communication software

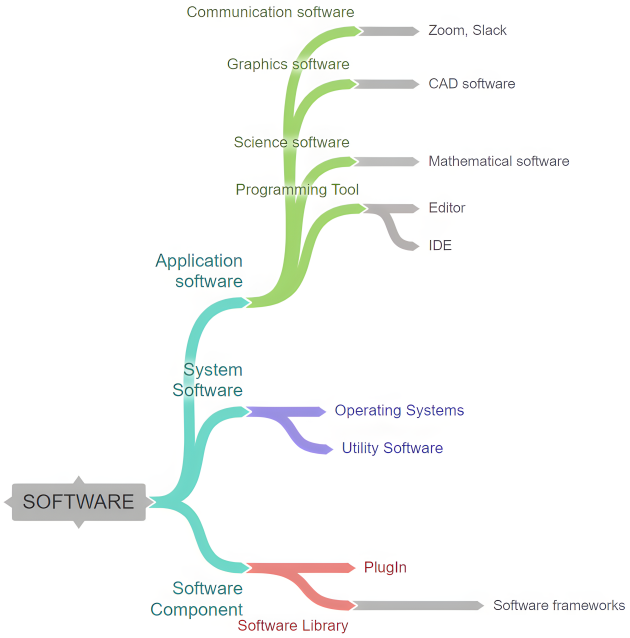


Figure 3.3: Relation between the three main software categories identified from WikiData.

3.2.2 *The software ontology (SWO)*

The SWO, particularly describes software used, for preparation and maintenance of data, within fields of computational biology and bio-informatics. The SWO was primarily developed to improve reproducibility by providing detailed description about software used for biomedical investigations [42].

SWO was found on ontology search Ontology Search (OLS) website[5] and was examined (on the date 13-12-2021) for possible software purposes. Unlike wikidata, a list of possible software purpose were found directly in "browse terms" section of the SWO website.

To navigate to the list of key words that suggest potential software purpose one can follow the following steps: "Browse terms"> "entity">"occurrent"> "planned" >"planned process". The software usage purpose in the SWO has been presented in to two main groups as "data transformation" and "data visualization". Under data transformation, 40 sub-types of potential software purposes are listed as shown on figure 3.4 (a screenshot taken from the SWO wesite on date 13-12-2021) below.

Some examples of software usage purposes retrieved from SWO are shown on the table 3.3 below.

5 https://www.ebi.ac.uk/ols/ontologies/swo

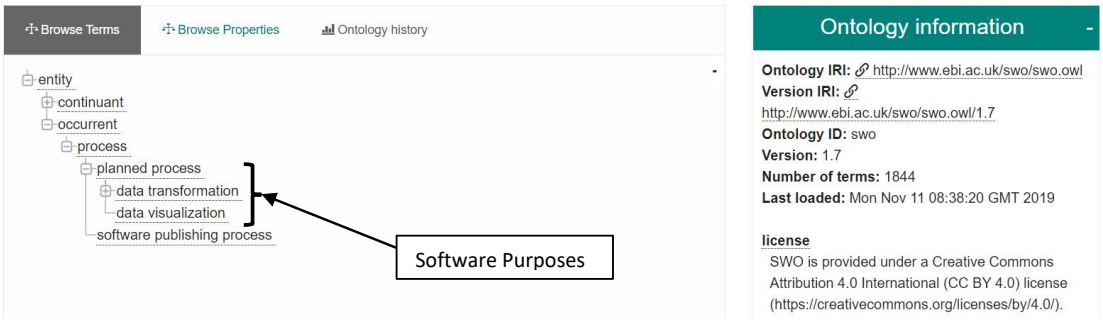


Figure 3.4: List of key words that indicate software usage purposes on the SWO Ontology website. Inside the data transformation section, there are 40 software usage purposes

Table 3.3: Examples of software usage purposes extracted from the SWO Ontology.

- Data transformation	- Calculation
- Genome annotation	- citation management
- Class discovery	- dataset comparison
- Annotation	- Analysis
- Text editing	- Data visualization
- Modeling	- File rendering
- Curve fitting	- Matrix manipulation
- Simulation	- Data mining task
- Query and retrieval	- Clustering task
- Distance calculation	- Image compression
- Molecular sequence analysis	- Data Normalization
- Decision tree induction	- Descriptive statistical analysis

3.2.3 *OntoSoft*

Onosoft[6] is a software registry framework that stores important metadata about software to foster reuse and sharing of software among scientific community. The ontology provides descriptions about a software that would help scientists to identify, understand, execute, and do research with a software. Moreover, it helps scientists get information about update and support for the software. Particularly, Ontosoft focuses on the geo-science because software resources are not being shared adequately in that field [21].

These descriptions are visualized in a pie-chart, on figure 3.5, with each slice indicating the completeness of the description. The grey color in each slice of the pie-chart shows how much level of detailed information has been provided for the software and red color indicates the amount of information missing.

6 https://www.ontosoft.org/portal/

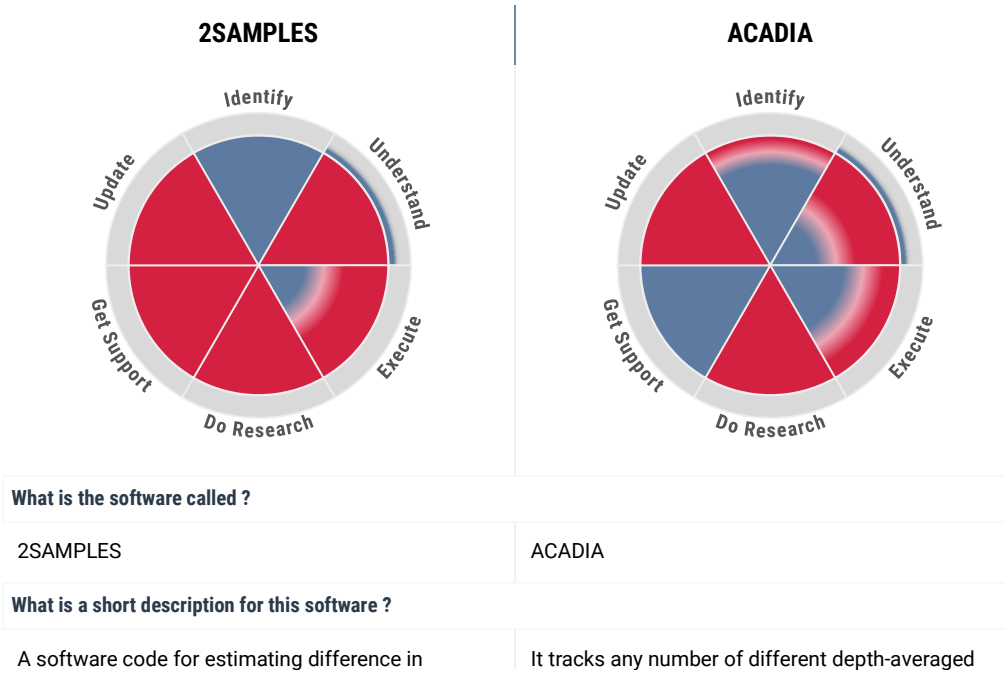


Figure 3.5: Comparison of two software tools using Ontosoft website (taken 15-12-2021), indicates which software has more level of detailed information.

Table 3.4: The six dimensions of information description in the Ontosoft software ontology

DIMENSION	DESCRIPTION
Identify	- Name/abbreviation of software, etc.
Understand	- domain specific key words , software creator, ...etc.
Execute	- URL, license, system requirements . . . etc.
Do Research	- file formats, preferred citation information, . . . etc.
Get support	- Contact details, possible support included, etc.
Update	- Version, developer community, maintenance, etc.

The type of information provided in each dimension of description entries are summarized in the table 3.4.

From the set of information provided among the 6 dimensions of the Ontosoft, particularly the "understand" dimension has nearly 400 domain specific key words that would potentially indicate software usage purposes. Therefore, those domain specific key words have been retrieved (on date 15-12-2021) from the Ontosoft website. After that the list has been manually analyzed and condensed into a more general list of software purposes. Table 3.5 lists samples of domain specific key-words that would potentially indicate software usage purposes.

Table 3.5: Examples of key-words extracted from Ontosoft ontology that indicate potential use of software.

Domain specific Key words	
- Data manipulation	- Numerical model
- Data Mining	- Numerical simulation
- Image processing	- Thermal model
- Machine learning	- Integrated modeling
- Simulation- optimization	- Interactive visualization
- Network analysis	- Wind wave estimation
- 3D printing	- data integration
- geospatial data conversion	- Image Processing
- Computer Vision	- Machine Learning
- Interactive Visualization	- Numerical simulation
- network analysis	- Performance Evaluation
- Visualization	- Wind wave estimation

3.3 ANALYSIS OF SCI-CRUNCH REPOSITORY

The other resource analyzed in addition to software ontologies, to list down software usage purposes, is Sci-crunch[7] repository. Sci-crunch is a data portal that searches through hundreds of community databases, aggregates information resources to create a large collection of data and tools available for access at a single spot [23].

To identify possible software usage purposes, the Sci-crunch repository has been analyzed by selecting software resources. At the time of retrieval (16-12-2021), the sci-crunch registry has listed 7,155 software tools. These software tools have been analyzed using the website's built-in word cloud-generator. The generated word cloud indicates types of software tools as shown on the figure 3.6.

From the word cloud, 200 types of software have been identified and manually analyzed to select important key words that potentially indicate possible software usage purposes. Sample of software types and a corresponding usage purpose is summarized on table 3.6 below.

3.4 TYPES OF SOFTWARE USAGE PURPOSES

Based on a through analysis of resources in key words that indicate software purposes has been listed in tables 3.1, 3.2, 3.3, 3.4 and 3.5 respectively. These list of key words have been further analyzed manually and 8 types of software usage purposes identified are Data collection, Data pre-processing, Analysis, Visualization, Simulation, stimulation, Modelling, and Programming.

7 https://scicrunch.org/

Figure 3.6: Word cloud retrieved from Sci-crunch repository indicating 200 different types of software that indicate potential purpose of software usage

Table 3.6: Software types in Sci-Crunch repository that suggest potential software usage purposes.

Software Type	**Purpose**
- Data acquisition software, Image acquisition software	- Data collection, Recording
- Data / Image / Sequence / Network Analysis software	- Data Analysis
- Text mining software	- Data Analysis
- Signal processing software	- Data Analysis
- Data / 3D Visualization software	- Data Visualization
- Simulation software	- Simulation
- Alignment software	- Data pre/post-processing
- Rendering software	- Modeling
- Code testing framework	- programming

To establish a clear boundary and avoid ambiguity during the annotation process of software usage statements, in SoMeSci dataset, each software usage purpose has been clearly defined based on literature in the next section as follows.

3.4.1 *Data collection*

According to Wikipedia, data collection is a process of collecting, recording or measuring information on targeted variables which enables answering of questions. Regardless of the type of data, quantitative or qualitative, data collection is one of the most important steps in a scientific investigation [74].

Scientists collect data for their research using various data collection software tools and gadgets. In one research, for instance, scientists used an Actigraph Reader Interface Unit (RIU-41A) with its software to measure the level of activity of more than 5000 children of age 12 to characterize the relation between physical activity and obesity [28, 49].

Often times data pre-processing involves several steps such as data cleaning, integration, transformation, reduction, etc. [41]. Data cleaning, for instance, involves dropping of data, replacing, or imputation of missing values in order to improve performance of algorithms and reliability of analysis results, especially in data mining applications [73, 77]. In a scientific investigation, scientists usually carry out data pre-processing using a custom script or using an existing application software or programming library.

3.4.2 *Data pre-processing*

Data collection processes produce inconsistent data and analysis of such noisy data might yield misleading results because of the "garbage in, garbage out" problem [75]. To avoid this problem, scientists usually carry out data pre-processing using a software. Data pre-processing generally refers to the addition, deletion, or transformation of raw data into a clean and tidy form to improve performance and reliability of analysis results, especially in data mining applications [33, 56].

Often times data pre-processing involves several steps such as data cleaning, integration, transformation, reduction, etc. [41]. Data cleaning, for instance, involves dropping of data, replacing, or imputation of missing values in order to improve performance of algorithms and reliability of analysis results, especially in data mining applications [73, 77]. In a scientific investigation, scientists usually carry out data pre-processing using a custom script or using an existing application software or programming library.

3.4.3 *Data Analysis*

Data analysis refers to processing, transforming, modeling, etc. of data with a goal of discovering a new insight that would support conclusions or decision making. Data analysis involves diverse techniques with different names in various domains [72]. In their research, scientists employ wide variety of software tools to carry out tasks of data analysis such as curve fitting, spectral smoothing using a software [55].

3.4.4 *Data visualization*

Data visualization refers to techniques that are used to communicate data or information effectively in the form of visual objects such as points, lines, bars, etc. in a graphic representation [76]. Many types of data visualization software such as MatLab, Tableau, etc. and programming libraries such as Plotly, Matplotlib, Seaborn, are commonly used in research as well as business analytics.

3.4.5 *Simulation*

Computer simulations mimic operation of real-world process or system using models that represent key-behaviors of the system. By varying variables of the simulation, predictions about behavior of systems can be made. Simulations have a wide range of application in scientific modeling of natural systems in physics, chemistry and biology [83]. Simulations are run to improve understanding of a problem [62].

3.4.6 *Stimulation*

Stimulation is the act of evoking the development of activity or response. Living organisms have sensory receptors that generate impulses that travel through nerve to the brain upon a reception of excitation by means of various agents, energy, collectively known as stimuli. Examples of sensory receptors in the human body are photoreceptors in the retina, touch receptors on the skin, chemical receptors in mouth, etc. [69]. In the scientific investigations, scientists use various mechanisms to provide a stimulation to their research object. One of the ways to provide stimulation is using a software. In neurological research, for example, scientists use various brain stimulation techniques and software to study neurological disorders. Deep Brain Stimulation (DBS), for instance, is one of brain stimulation techniques used to treat diseases like Parkinson's, essential tremor, dystonia etc. [58].

3.4.7 *Modeling*

Modeling refers to scientific activities that aim to facilitate understanding of a particular feature or phenomena in the world. It is a process of identifying and selecting relevant aspects of a situation or phenomenon under consideration. Different types of models with more specific aim exist. For instance, conceptual modeling provide better understanding, mathematical models help to quantify, computational models are used for simulation, etc. [81].

Modeling is a broad term that refers to a wide range of activities. It might also refer to 3D modeling and graphical representation of a real world physical objects like vehicles, buildings, ... etc. using Computer Aided Design (CAD) software. For instance, some scientists use graphical modeling software, for instance for digitally documenting historical sites such as castles [15]. On the other hand modeling can also refer to mathematical representation of a non physical abstract entity. In one research paper, for instance, the researchers were trying to model the occurrence of letters and word's initials mathematically [53].

Regardless of the wide meaning and techniques of modeling, inherently all models serve to represent an object or a system to facilitate the representation, or understanding of particular feature or phenomena [78, 81].

Table 3.7: Brief summary of software usage purposes and examples.

USAGE PURPOSE	Examples
1. DATA COLLECTION	Surveying, Data acquisition, Data recording, Text extraction, Measurement, Constructing an artificial data set, Importing a file or data of specific format into a software, ... etc.
2. DATA Pre-processing	Data cleaning, Data normalization, Data encoding, Data reduction, Text editing, Error correction, Data type conversion , Missing data handling, Data transformation, Tabulating, merging data, File formatting, Aligning gene, ... etc.
3. ANALYSIS	Data manipulation, Tabulating, merging data, Testing hypothesis, Data mining, clustering, Prediction, prediction, quantification,calculation, computation, comparing, testing, searching, assessing, evaluating , ...etc.
4. VISUALIZATION	Creating figures, Plotting, Graph/Figure generation, ... etc.
5. SIMULATION	Flight/ Event/Numerical/ simulation, Flood dynamics simulation, vehicle schedule simulation, etc.
6. STIMULATION	Stimulate behavior
7. MODELING	Scientific/Mathematical/ modeling, Machine learning / Model fitting, Predicting a behavior, Estimating, Inference, ... etc.
8. PROGRAMMING	Programming, implementing, ...etc.

3.4.8 *Programing*

Programming refers to the process of designing and building executable computer programs that performs a specific task. Computer programs are written in a human readable format mainly to automate execution of complex tasks and for solving problems [71]. Summary of software usage purposes and corresponding examples are summarized in the table 3.7.

4

DATASET

Training and evaluation of automatic information extraction approaches requires availability of reliable ground truth data of sufficient size. Following a growth of interest for extraction of information about software tools from scientific publications labeled datasets with limited scope such as BioNerDs, SoftCite, SoSciSoSci have came into existence. More recently, SoMeSci dataset, a more comprehensive corpus that covers a wide range of information about software tools has also been introduced [59].

This section describes the dataset used in this project – SoMeSci, the enhancement of the dataset with software purpose annotations, issues observed during annotation, preprocessing of the dataset, analysis results of the data and transformation to a suitable format for training purpose.

4.1 SOMESCI DATASET

SoMeSci dataset contains high quality, hand annotated articles collated from PubMed Central (PMC). The articles and annotations included in the dataset are summarized below.

4.1.1 *SoMeSci Articles*

The corpus is composed of four group of files, namely*PLoS methods, PLoS sentences, PubMed full text* and *Creation sentences*. Facts about the articles in the SoMeSci corpus is summarized in the table 4.1.

Table 4.1: Article composition of SoMeSci dataset.

SoMeSci article	Description
1. PLoS Methods	- 480 articles from PLoS journal that contain only methods sections
2. PubMed full text	- 100 randomly selected full-articles from PMC Open Access
3. PLoS sentences	- 677 files extracted from PLoS articles which contain sentences that has software names mentioned in them.
4. Creation sentences	- 110 files that contain sentences indicating creation of a software files out of which 50 are extracted from PMC OA, where as the rest 60 are from PLoS.
Total	- 1367 files

Table 4.2: Annotation composition of SoMeSci dataset before its extension with software usage purpose labels.

Annotation category	Types of Annotations
4 software-types	- Application , Plugin , Operating System and Programming Environment
4 mention types	- Mention, Usage, Creation, and Deposition
9 additional info.	- Developer, Version, URL, Extension, Release, License, Abbreviation, & Alternative name.

4.1.2 *SoMeSci Annotations*

SoMeSci corpus has three main types of annotations that correspond to a type of information related with software tools. These annotations indicate the type of software, type of mention and additional information about the software as summarized on the table 4.2.

4.2 ANNOTATION TOOL

The dataset has been annotated using BRAT rapid annotation tool (version 1.3) (RRID:SCR_008769) in a Linux 20.4 environment. The annotation tool has been run in a local machine as a Common Gateway Interface (CGI) application using a browser.

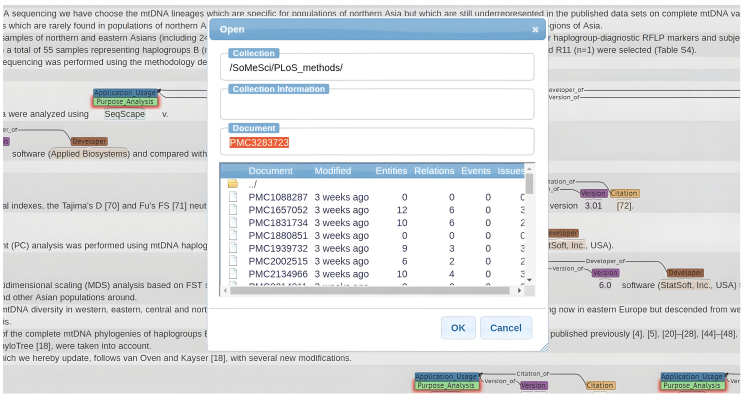


Figure 4.1: BRAT annotation tool

4.2.1 *Annotation of SoMeSci with software purpose labels*

SoMeSci corpus has been enhanced with annotations of eight classes of purpose of software usage labels identified in the earlier section (summarized in table 3.7). Since using software for a particular purpose only refers to the usage of a software, only usage annotations among the 4 types of annotations described in the table 4.2), have been further labeled with software purpose. The figures below show SoMeSci dataset before and after software purpose annotations.

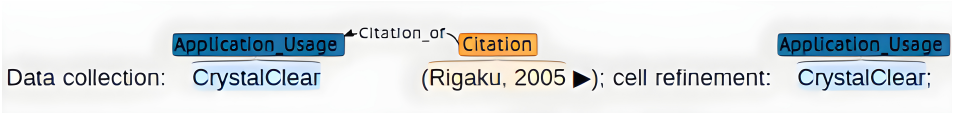


Figure 4.2: Original annotations in SoMeSci dataset before extension with software usage purpose annotations

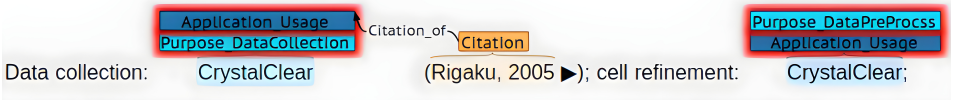


Figure 4.3: SoMeSci annotations after extension with software usage purpose labels

4.2.2 *Top-level annotation*

For the sake of simplicity, certain types of software usages have been assigned the same class of software purpose annotation. For example, modeling might refer to graphical modeling of an object using CAD software or it might also refer to mathematical representation of a given problem. All such variants of modeling tasks have been assigned "modeling" , a top-level class, as a label without differentiating specific variants.

4.2.3 *Challenges during Annotation*

There were cases where a given software had multiple types of software usage purpose where throughout the annotation process only a single software usage purpose was being assigned. In such cases, annotations has been carried out in a such way by deciding on each context which software purpose annotation is more important or based on the general goal of the software usage.

For example, FlexArray software on the figure below, has been annotated with software purpose analysis even though the same software was used for visualization purpose as well. This is because on this context analysis is more important than visualization and essentially visualization could also be interpreted as one kind of analysis. In addition, specific definition of each of software usage purposes has been also taken into account.

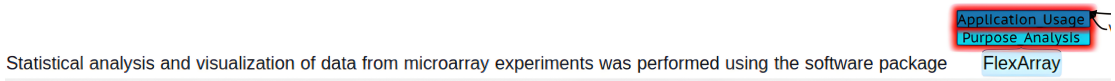


Figure 4.4: Selection of priority for annotations based on context: Visualization is one form of analysis hence label analysis is selected over visualization

However, in some instances, annotation of software-purposes appears to be ambiguous. For example a software used for counting or quantification can be assigned a software purpose label of *data collection* but also *analysis*. Such cases of annotation with software usage purpose are subjective and can affect the over all annotation quality.

The other challenge of annotation was difficulty arising from limited domain knowledge. In some cases, labeling of purpose of use of a software was quite difficult because of lack of understanding what the scientists were trying to do using the software.

4.3 DATA PRE-PROCESSING

Pre-processing of the dataset has been carried out to ensure the integrity of our dataset before using it in the classifier. The data pre-processing tasks handled annotation errors, merging annotations , transforming and splitting of dataset.

4.3.1 *Automatic Integrity Testing*

As described in the above table, the four types of software mentions in the SoMeSci are mention, usage, creation and deposition. The main goal of annotating the dataset was to assign corresponding purpose of software usage label to each instance of software usage but not for mention, creation and deposition. However, due to an error there were some instances of software mention that has been annotated with software purpose. In addition to this, there were also some instances of usage, that has not been annotated and intentionally skipped because of the purpose of software usage did not seem to be clear.

Therefore, all instances of wrong or missing annotations have been identified automatically to ensure the integrity of training dataset.The python code for automatic identification of annotation errors has been listed under *Appendix-B*.

4.3.2 *Merging Annotations*

After handing all annotation errors and missing labels, annotations of software usage has been merged with annotations of software purpose. Merging of the annotations has been carried out to unify and integrate software usage purpose annotations with the original annotations. The python code for merging annotations has been listed on the *Appendix-C*. Figures 4.5 and 4.6 below shows, software purpose labels before and after integration into the original annotation respectively.

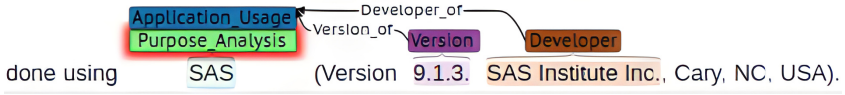


Figure 4.5: Software usage annotations before merging with software purpose annotation.

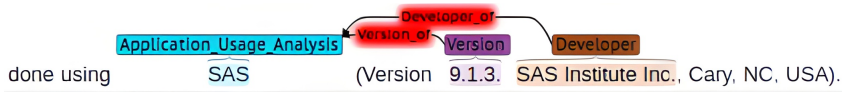
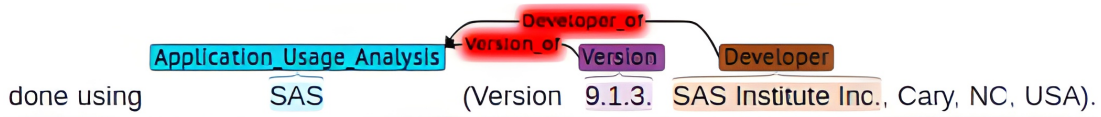


Figure 4.6: Software usage annotations after merging with software purpose annotation.

4.3.3 *Transformation to IOB format*

After merging software usage and purpose labels, transformation of data into IOB format has been carried out using articlenizer (link to articlenizer). Figure 4.7 below shows the data format before and after transformation.




```
O O B-Application_Usage_Analysis O O B-Version I-Version B-Developer I-Developer I-Developer I-Developer
```

Figure 4.7: Merged software usage and purpose annotation & its corresponding IOB representation.

4.4 ANALYSIS OF ANNOTATED DATA

Analysis of cleaned SoMeSci dataset has been carried out to find a deeper insight about the training data.

4.4.1 *Co-reference resolution of software entities*

The base line for the analysis of the dataset was to carry out disambiguation of software names. This was particularly important because there is large degree of variation in software names. Using list of software name with corresponding URL, software mention instances have been disambiguated from each other and all software name variations that refer to the same entity have been given the same name.

Figure 4.8 below shows name variations for MATLAB software, in which all instances resolve to the same URL i.e. entities referring to the same software. All variations of names has been replaced by the first "Matlab" in this case.

```
# example result for Matlab

for lnk in list(sw_nameVarations):

    if lnk == "https://www.wikidata.org/wiki/Q169478":
        print(sw_nameVarations.get(lnk)," resolves to ->",lnk)
    #break
```

```
['Matlab', 'MATLAB', 'MatLab', 'MATLB']  resolves to -> https://www.wikidata.org/wiki/Q169478
```

Figure 4.8: Example of Co-reference resolution for Matlab software. All name variations have been given the same name.

4.4.2 *Analysis results*

According to the analysis results, the top 3 software by number of software name mention count through out the list of articles in PubMed and PLoS dataset are: PASW, GNU-R and STATA as shown on the figure 4.9 below.

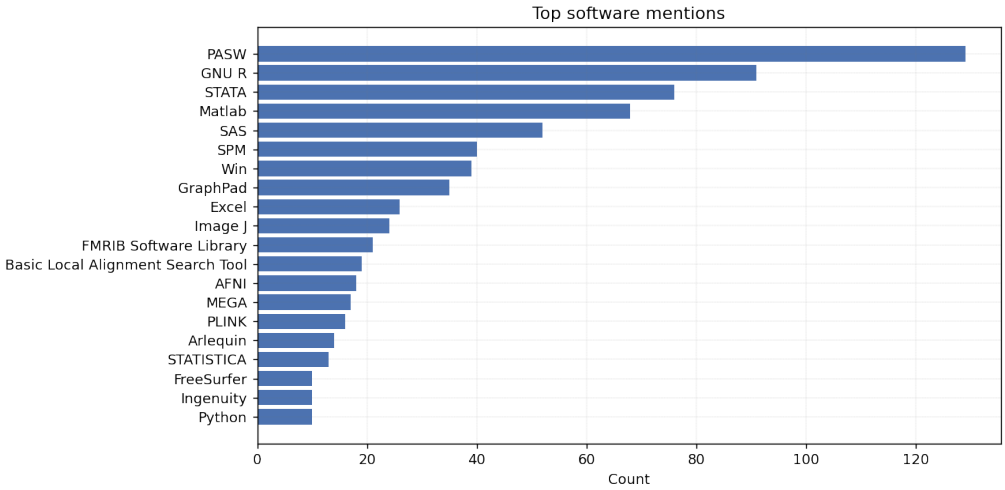
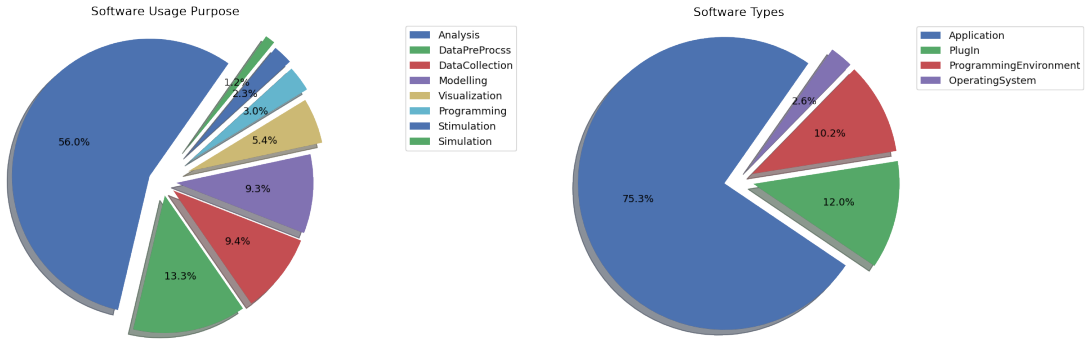


Figure 4.9: Top software mentions

Dataset analysis result from the perspective of purpose of software usage indicates that the most common purpose of software usage are: Analysis, Data pre-processing, Data collection and modeling where as the least common are simulation and stimulation. The result is shown on figure 4.10 below.

The other insight form from the software-type perspective, is that the most commonly used type of software in the research articles in the SoMeSci dataset is Application software as depicted on figure 4.11.



(a) software purpose

(b) Types of software

Figure 4.10: Software usage purposes and types of software in the SoMeSci dataset.

When it comes to share of each purpose of software usage among the 4 types of software, the pattern once again clearly indicates most of the time a software has been used for the purpose of analysis and data collection in all of the four software types.

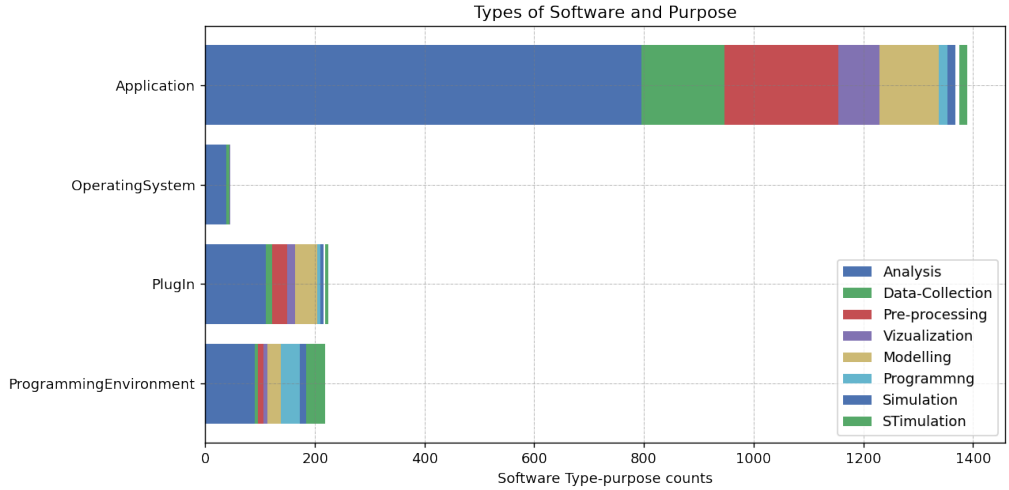


Figure 4.11: Share of software usage purpose from each type of software in the SoMeSci dataset.

Lastly, the other interesting insight observed from analysis of the dataset was determining: " For how many different purposes has a given software been used ?". The analysis reveals that from 657 unique software, nearly 76% have been used only for a single purpose as shown on the bar graph 4.12 below. Over all, this result indicates that most of the time software tools have been used only for a specific purposes.

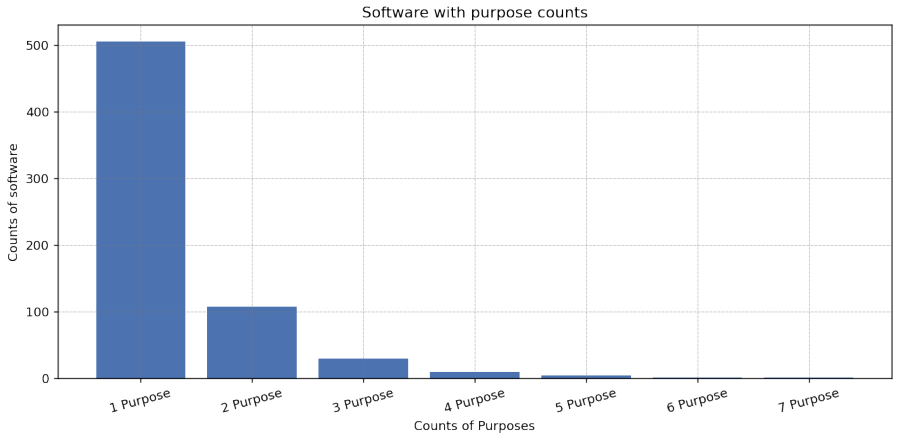


Figure 4.12: Number of purposes a software is used for, most software tools in SoMeSci are used for just a specific purpose.

4.5 DATA SPLITTING AND OPTIMIZATION

After the data has been transformed into the IOB format, the dataset, SoMeSci, has been split into train, test and development set with 60, 20, 20 ratio respectively. To ensure the distribution of enough samples of class labels, for all 8 types software usage purpose class labels, the dataset has been iteratively split until each of software usage purpose class label lies within +/- 5% range for all train, test and development datasets.

Initially the iterative data splitting was carried out separately for PLoS methods and PubMed-full text. However, due to lack of enough class labels in PubMed-full text, especially for stimulation class, the iterative splitting did not converge. For this reason PLoS and PubMed articles have been combined to make the splitting converge.

The most optimal data split obtained from the iterative splitting is shown on the table 4.3 below, where there is still class imbalance for stimulation in the development dataset. Finally, few articles from the test set have been manually moved into the development set to make the data split comply 100% within +/- 5%.

Table 4.3: The optimal datasplit for class labels before further manual balancing.

Software Purpose	Train (%)	Dev.(%)	Test (%)	Total
Analysis	890 (62%)	263 (18%)	283 (20%)	1436
Data Collection	150 (61%)	48 (19%)	48 (20%)	246
Pre-processing	205 (61%)	64 (19%)	66 (20%)	335
Modeling	141 (62%)	35 (15%)	51 (23%)	227
Programming	51 (64%)	14 (17%)	15(19%)	80
Stimulation	16 (57%)	6 (21.5%)	6 (21.5%)	28
Simulation	45 (63%)	9 (13%)	17 (20%)	71
Visualization	70 (60%)	27 (23%)	20 (17%)	117

Part of a python code for optimizing the data split has been listed on *appendix D*. The notebook for dataset splitting and optimization can be found on a git-hub repository[1].

1 https://github.com/BeTKH/SoMeNLP/tree/contxt2Sentcs_wo/bin

5 CLASSIFIER MODELS

Extraction of information about a software can be done following various approaches. In the past, rudimentary approaches such as searching for a term in paper, manual content analysis using human readers as well as rule based approaches have been employed [32]. Recently, a deep learning model have also been employed for automatic extraction of information about software such as mention types, software type, etc. [60].

Software is a real world entity and extraction of information about software can be approached as a named entity recognition problem. Named Entity Recognition (NER) tasks are one of classical sequence labeling tasks of NLP among others such as Part-of-Speech (POS) tagging and text chunking [3, 25]. Therefore, automatic classification of software usage purpose can be approached as a sequence labeling task in which a class label, from a fixed list of class labels, is assigned to each token in a sequence.

Various types of sequence labeling models can be broadly categorized as statistical machine learning based or neural network based models [25]. Classical machine learning based models for sequence labeling are HMM [34], MEMM [44], and CRF [35]. These models are statistical and fundamentally depend on manually crafted features, external resources such as gazetteers and fail to adapt to new domains [40].

In contrast, there are various neural network architectures and models that suit sequence labeling. Recurrent Neural Network (RNN), based on LSTM, had been well established models for sequence labeling. Recently, transformer based neural network architectures, such as BERT, have pushed state-of-the-art performance further not only in sequence labeling tasks but for most Natural Language Processing (NLP) tasks [65].

This section presents and compares various types of sequence labeling models suitable for sequence labeling task of automatic classification of software usage purposes.

5.1 MACHINE LEARNING MODELS

5.1.1 *Hidden Markov models (HMMs)*

One of basic machine learning models that are used for sequence labeling are Hidden Markov Models. HMMs are based on Markov processes which describes a sequence of hidden finite states (Yi) in which a given state in a sequence depends only on the state prior to it [1, 18].

The Markov chain of hidden states Yi generate observations Xi based on its current state using a joint probability distribution P(Y, X). The joint probability P(Y, X) is a product of conditional probability P(X | Y) and a prior probability P(Y).

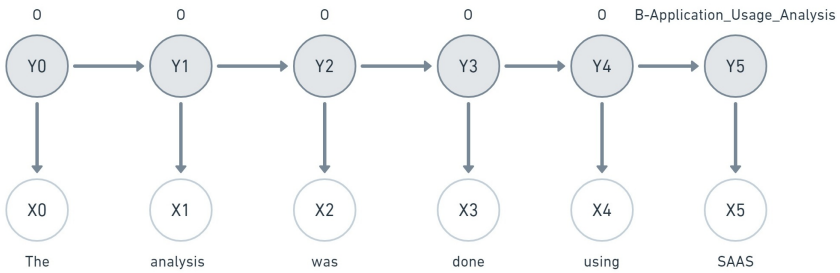


Figure 5.1: HMM Model: a directed graph indicates generation of tokens depends on exactly the state before.

As the model transitions through hidden states of Yi, it generates an observation Xi which corresponds to a token in a sentence, hence HMMs are also referred to as generative models [1].

The challenge with Hidden Markov Models is that, it could be difficult to define the joint probability for some real world applications where the number of hidden states is infinite. In addition, Hidden Markov Models fail to capture long range dependencies because of their Markov assumption which considers only prior states [8, 68].

5.1.2 *Maximum Entropy Markov Models (MEMMs)*

Unlike hidden Markov models which generate sequences of tokens based on hidden states, maximum entropy models are discriminative models i.e., they directly model the probability of each label Yi based on the current observation Xi and the prior hidden state Yi-1 [44].

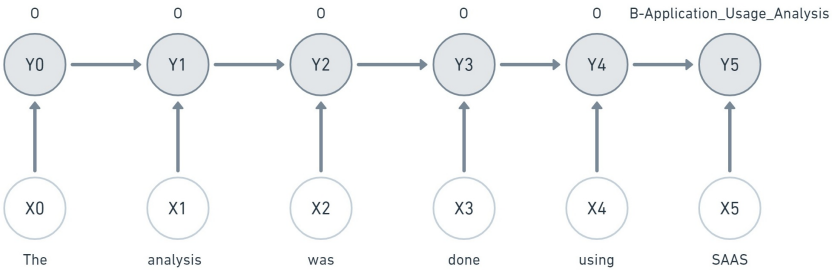


Figure 5.2: MEMMs classify each token based on current observation and previous label

The biggest drawback with MEMMs and other discriminative directed graphical models based on Markov, tend to be biased in favor of states with fewer successor states i.e. [35].

5.1.3 *(Linear) Conditional Random Fields (CRFs)*

Conditional Random Fields are similar with MEMMs in that both are probabilistic and discriminative models that can perform sequence labeling based on conditional probability P(Y | X) unlike joint probability of HMMs [68]. A sequence of tokens in a sentence corresponds to X and Y refers to a sequence of class labels [35].

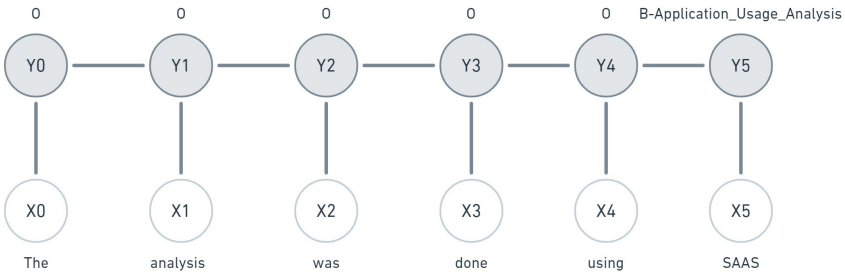


Figure 5.3: CRF Model: un-directed graph indicates that classification at a given instance depends on labels occurring before and after.

CRFs differ from MEMMs in that they are undirected graphs, i.e., the inference of Yi depends on all the labels occurring before it as well as after it. This makes training of CRFs computationally expensive especially when a wider window of tokens is chosen to handle long range dependencies [1].

5.2 DEEP LEARNING MODELS

Deep learning based models are state-of-the-art for many machine learning and NLP tasks, including sequence labeling. There are various types of deep learning architectures that suit for sequence labeling. Popular neural network architectures for sequence labeling tasks are recurrent neural networks and transformer networks.

Recurrent neural network based models such as LSTM, Bi-LSTM, Bi-LSTM-CRF had been state-of-the-art sequence labeling models, before the inception of transformer-encoder based models like BERT which produced a new state-of-the-art performance for many NLP tasks including sequence labeling.

5.2.1 *Long Short-Term Memory (LSTM)*

RNNs are specific type of neural network architectures that take a sequence of inputs features to yield a sequence of labels. However, plain RNNs fail to capture long-range dependencies because of vanishing and exploding gradient problems [17, 54].

Fortunately, other variants of RNNs such as LSTM networks are capable overcoming unstable gradient problems [3, 36, 40]. This is because, unlike RNNs, LSTM networks have memory cells that are composed of three gateways which determine what amount of information to forget at a given instant of time [40].

LSTM networks are typically composed of input layer, hidden layer(s), and output layer. Number of inputs to the input layer corresponds to input features and the number of outputs on the output layer corresponds to number of classes of the classification task. The memory cells of LSTM are found in the hidden layers of the network [17]. A memory cell structure is depicted on the figure 5.4 below.

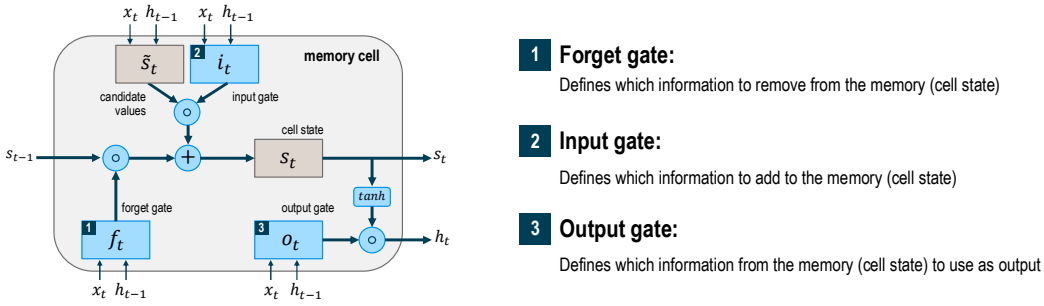


Figure 5.4: LSTM Cell [17]

5.2.2 *Bi-Long Short-Term Memory (Bi-LSTM)*

For sequence labeling tasks, it is desirable to access left as well as right input features to capture context information. However, LSTM models are capable of remembering only past (left) sequence of features.

To solve this problem, Bi-LSTM combines two LSTM networks where the first LSTM unit captures input features in a forward direction while the other captures the information in the reverse direction. Finally the two input features get concatenated to create a feature representation that encapsulates context information in both directions [40].

5.2.3 Bi-LSTM-CRF

In sequence labeling, it is also important to capture correlations among adjacent input features at a sentence level. Since CRFs are capable of incorporating correlations among neighboring input features in a seqence, outputs of Bi-LSTM can be fed into CRF to create Bi-LSTM-CRF model which is aware of correlation between neighboring features in addition to context in both directions [40].

The use of past and future inputs using Bi-LSTM combined with CRF to capture correlations among features at a sentence level boosts sequence labeling accuracy. Because of this the Bi-LSTM-CRF model produce state-of-the-art results among LSTM based models. For example, an F1 score of 97.55%, 94.46% and 88.83% is achieved on classical sequence labeling tasks of POS, chunking (CoNLL 2000) and NER (CoNLL-2003) respectively [27].

5.3 TRANSFORMER BASED MODELS

Although Bi-LSTM-CRF models have superior performance for sequence labeling, over other LSTM based models, they have inherent weaknesses. Firstly, Bi-LSTM-CRFs are not truly bi-directional context readers, rather pseudo-directionality is achieved learning right and left contexts separately and concatenating them. Because of this the true context of words could be lost slightly.

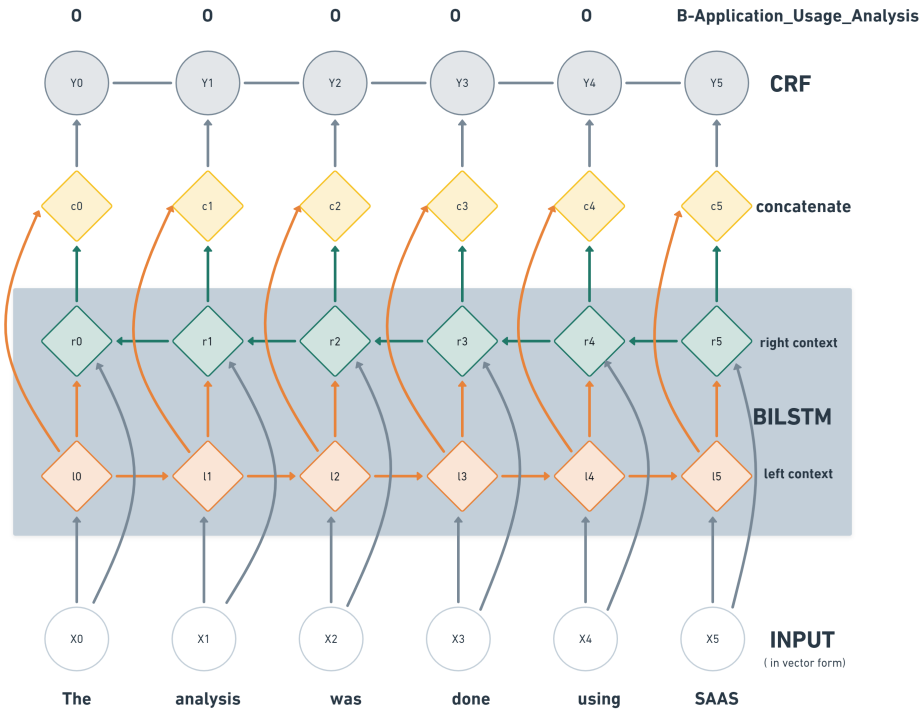


Figure 5.5: A Bi-LSTM-CRF model: the Bi-LSTM layer captures context information in two directions from left to right and vice versa.

Secondly, Bi-LSTM-CRFs are very slow to train because of their sequential nature which does not allow to take advantage of parallelization using Graphics Processing Unit (GPU)s. Third, training of Bi-LSTM-CRFs, like any other deep learning models, requires a lot of data and training models from scratch is computationally expensive.

The latest deep learning models based on transformer architectures are capable of overcoming shortcomings of Bi-LSTM-CRFs. First, transformer based models employ transfer learning where knowledge gained from pre-training can be re-used and fine tuned to solve specific problems [16]. In addition, transformer architectures have the ability to read past and future context simultaneously unlike Bi-LSTM-CRFs [11]. Moreover, unlike LSTM networks, transformer models can handle sequential data simultaneously which gives the opportunity to expediate model training by parallelizing with the help of modern GPUs [16].

5.3.1 *Bidirectional Encoder Representations from Transformers (BERT)*

BERT is state-of-the-art transformer based model with transfer-learning capability i.e. it can be pre-trained with unlabeled data and fine-tuned further for downstream applications just by adding one output layer [11, 16].

BERT is also referred to as Masked Language Model (MLM) because, the model arbitrarily masks and predicts a hidden word from a given sentence during pre-training. This is why BERT is capable of capturing right and left context simultaneously [11].

Originally BERT is pre-trained on BooksCorpus (0.8 Billion words) and English Wikipedia (2.5 Billion words). However, there are also other varieties of BERT, such as SCI-BERT, BIO-BERT, DistilBERT, etc. [7, 37, 57]. These models differ only by a corpus used during the pre-training step. Comparison of corpuses of BERT, Sci-BERT and Bio-BERT is summarized on the table 5.1 below.

Table 5.1: Transformer models BERT, SCI-BERT and BIO-BERT corpus

BERT	**SciBERT**	**BioBERT**
- 3.3 billion Tokens - trained on general domain corpora as news articles and Wikipedia.	- 3.17 Billion Tokens - trained on scientific text (1.14M papers from semantic scholar)	- 4.5B words PubMed (abstracts) - 13.5B words PMC (full text)

For the task of software purpose classification, BERT model has been chosen because of the its merit over LSTM based models. Among verities of BERT particularly Sci-BERT is chosen because its training corpus is based on scientific-text which is more relevant for software purpose classification. In addition, Bio-BERT pre-trained model is also trained to compare performance with Sci-BERT.

The implementation of the software purpose classifier follows a previous project with SoMeSci dataset, SoMeNLP, which is a cascade of classifier modules for software, software-type and mention-type. Therefore, another module for software purpose classification is added on the top of previous cascade of classifier modules. A fully connected multi-class BERT classifier model implemented on this project is shown on the figure below.

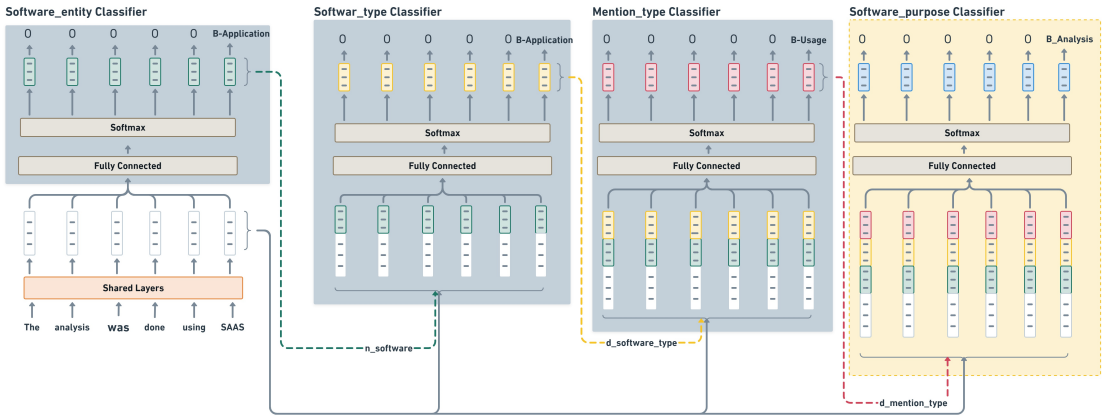


Figure 5.6: Fully connected multi class 4-cascade classifier model for software, software-type, mention-type and software-purpose classification respectively.

6

MODEL TRAINING AND OPTIMIZATION

To automatically classify software usage purposes, SCI-BERT multi-class model has been trained and tested using SOMESCI dataset. The model has been trained in various training scenarios to determine conditions that leads to improved performance. The training scenarios considered the effect of context information and the impact of including or excluding parts of SOMESCI dataset on the classifier's performance.

In addition, *software-purpose* classification performance has also been investigated by removing *mention-type* and *software-type* classifiers from cascades of classifier modules shown on the figure 5.6. Further more, the impact of using another variant of BERT model, BIO-BERT, on classifier's performance has also been investigated.

Finally, results from all training scenarios have been discussed and best performing model has been selected based on the results of investigation.

6.1 THE IMPACT OF LABELED DATA

As described in the table 4.1 before, the SOMESCI dataset is composed of 4 different sets of articles: *PLoS-methods*, *PubMed-full text*, *PLoS-sentences* and *creation-sentences*. Since only articles in the *PLoS-methods* and *PubMed-full text* are annotated with software usage purpose labels, it was desired to evaluate weather including *PLoS/Creation* sentences, would result in improved performance of *software-purpose* classification.

The results of evaluation indicates that including *PLoS-sentences* and *creation-sentences* in the training dataset, definitely improved the overall performance for classification of software, but not for software usage purpose. Figure 6.1 below shows total F-score for *software classification* over test and development sets is improved with the addition of *PLoS/Creation* sentences in the training dataset.

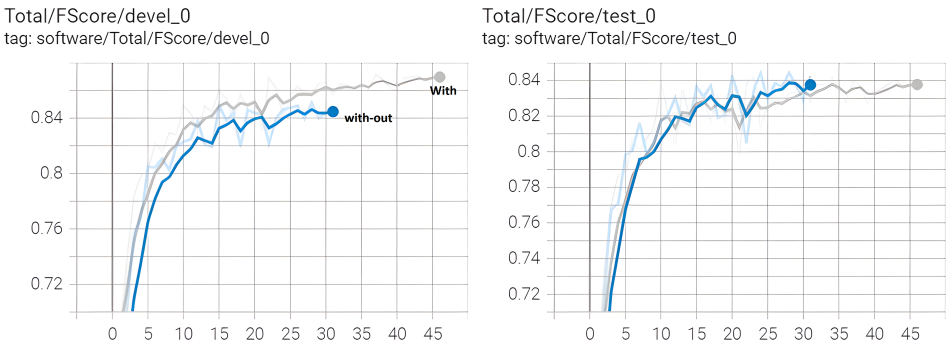


Figure 6.1: Software classification (Total) F1-score over devl.(left) and test(right) sets shows improvement when *Creation/PLoS* sentences are included in the training dataset.

In contrast, the total F-score performance for *software-purpose* classification is diminished when trained with *PLoS* and *Creation* sentences as shown on the Figure 6.2.

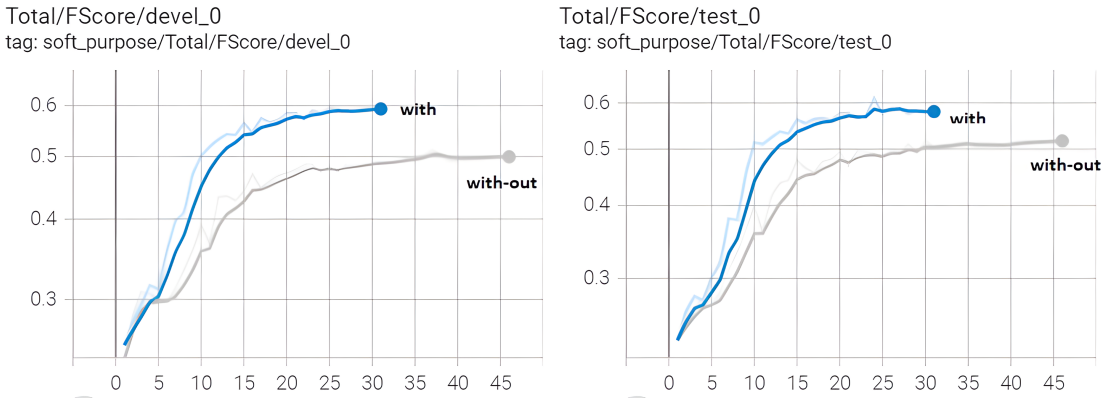


Figure 6.2: Software purpose classification (Total) F1-score degrades for devl. set (left) and test set (right) when *Creation/PLoS* sentences are included in the training dataset.

Software purpose classifier's performance when trained with *Creation/PLoS* sentences versus without is summarized on the table 6.1. Overall, as shown in table 6.1, it is observed that including a dataset that lacks software purpose annotation harms *software-purpose* classifier's performance. Since the main objective of this thesis is software purpose classification, for subsequent steps of analysis the Sci-BERT model has been trained only with datasets of *PLoS-methods* and *PubMed-full text* by excluding *Creation/-PLoS* sentences.

6.2 THE IMPACT OF LARGER CONTEXT

Scientific papers like any other well written documents, have a sequential structure that form abstraction at various levels such as sentence, paragraphs, sections, chapters, etc. These levels of abstractions often determine the meaning of words because each level of abstraction or context conveys a valuable information [20].

Likewise, contextual information helps to determine the correct purpose of use of a software. Therefore for each sentence in a text of scientific articles of training dataset, various range of neighboring sentences have been incorporated to provide a contextual information. Typically 2 adjacent sentences before a sentence, after a sentence, and before & after a sentence have been considered. Further more, a context as broad as the whole paragraph has also been considered.

Excerpt of the python code for reading neighboring sentences for context is listed on the *appendix-E* and the complete source code can be found on the git-hub.[1]

1 https://github.com/BeTKH/SoMeNLP/blob/contxt2Sentcs_wo/somenlp/NER/data_handler.py

Evaluation of a Sci-BERT model with a broader context of two adjacent sentences, from left and right, improved software purpose classification F-score as shown on the figure 6.3 below.

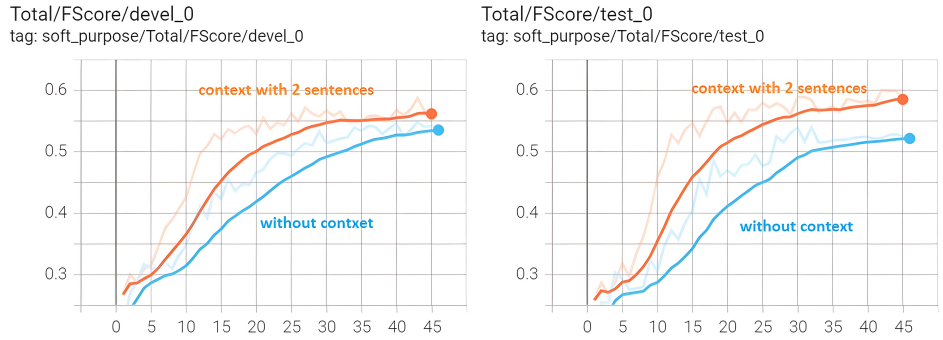


Figure 6.3: Software purpose classification (Total) F1-score improves for devl. set (left) and test set (right) when trained with a broader context of two adjacent sentences compared to without context (blue).

6.2.1 *Left vs Right Context within a paragraph*

In addition to a broader context, it was also desired to determine which part of context, left versus right, is more important for the software purpose classification. Left context refers to sentences prior to a given sentence with in a paragraph and right context refers to sentences that lie right after a given sentence. To determine the impact of neighboring context on the software-purpose classification performance, two adjacent sentences from the left and right has been used to train the Sci-BERT model.

According to the classification's F1-Score, sentences from the left are more important than the sentences on the right side. In addition, the 2-sentences from the left (2,0) has more significance over 2-sentences from left-and-right (2,2) as shown on the table 6.2. This indicates that the Sci-BERT classifier is not benefited any more by including the right context(0,2) or both contexts(2,2) as portrayed on the figure 6.4. below.

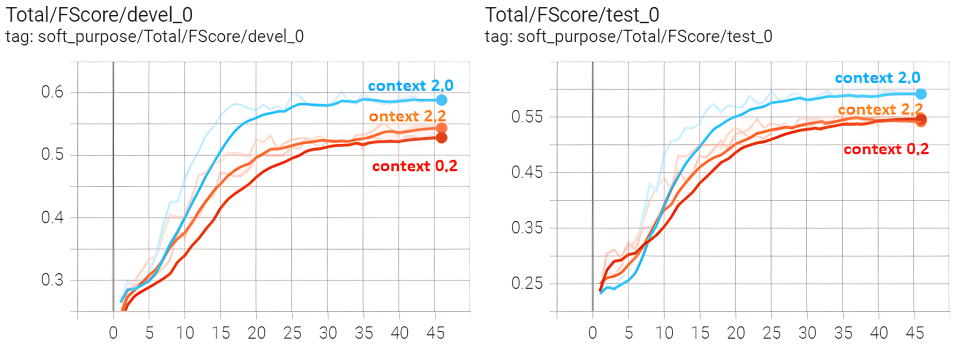


Figure 6.4: Software purpose classification (Total) F1-score performance for devl. set (left) and test set (right) when trained with left-context (2,0) has superior performance over right-context (0,2) as well as combined left-and-right-context (2,2).

6.2.2 *Context as whole paragraph*

In a common sense, a use of broader context such as more than two adjacent sentences for model training is supposed to improve the classification performance. However, the result of analysis, as indicated on the figure 6.4, reveals that a larger context such as 2 sentences from the left and right actually yields less F-score performance for the classifier model. To further validate this result, a context as large as whole paragraph has been considered.

According to the total F-score, software purpose classification actually worsened when entire sentences in the paragraph are used for context information.

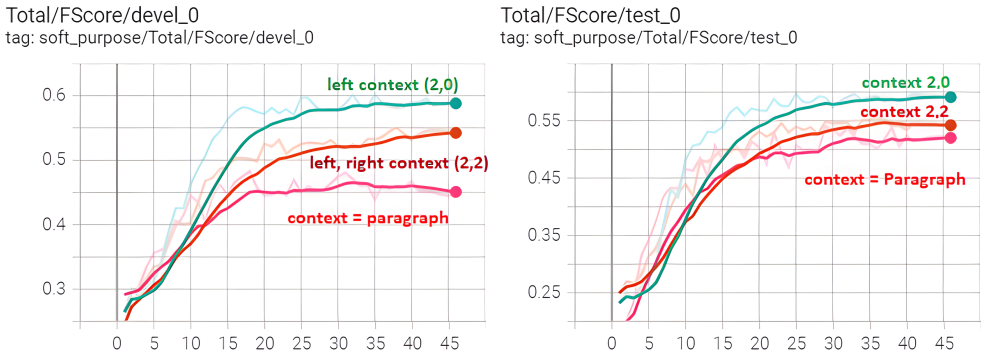


Figure 6.5: Software purpose classification (Total) F1-score performance for devl. set (left) and test set (right) deteriorates when very large context, paragraph, is considered.

6.2.3 *Context Outside a Paragraph*

The other scenario considered was evaluation of software purpose classification performance when context is not limited within a paragraph. Accordingly, classification performance with 2 adjacent sentences with in a paragraph is compared with context information of 2-adjacent sentences regardless of their position in the paragraph.

According to the classifier's F-score, software purpose classification degraded when a context is not limited within a paragraph as shown on the figure 6.6. This agrees with the fact that each paragraph of a scientific publication conveys a specific information and a contextual information outside a paragraph is not useful for the classifier. The Sci-BERT classifier model which considers context outside a paragraph is listed at the git-hub[2].

Based on evaluation of software purpose classifier's performance, overall it was observed that left context is more important than the right context. In addition, it was observed that context from both left and right as well as a broader context outside a paragraph did not improve the classification performance.

2 https://github.com/BeTKH/SoMeNLP/tree/contxt_out_parg

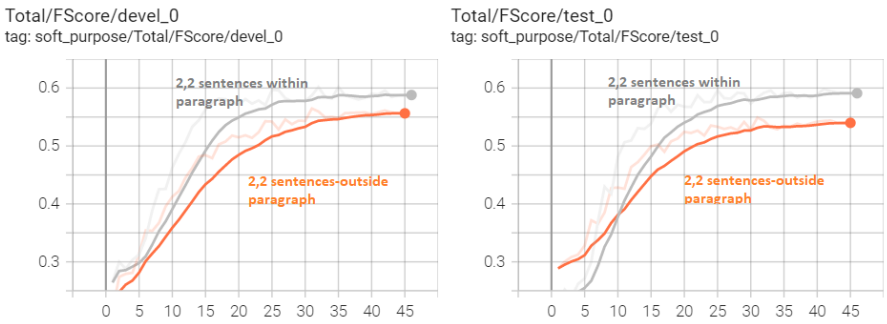


Figure 6.6: Software purpose classification (Total) F1-score performance for devel. set (left) and test set (right) did not benefit from context outside a paragraph.

Based on results, shown on figure 6.6, for all subsequent steps of software-purpose classification model evaluation, only two adjacent sentences from the left context are considered.

6.3 CLASSIFICATION WITH 2-CASCADE SCI-BERT

The 4-cascade SCI-BERT multi-class classifier model, shown on the figure 5.6, has been used for the software purpose classification until this point. This model, not only classifies software usage purposes but also other information about software such as software as an application, software types, and mention-types.

Overall the 4-cascade SCI-BERT classifier's performance has been poor regardless of consideration of various factors discussed above. For this reason, it was desired to study weather the removal of intermediate classifier modules would improve software purpose classification. Consequently, *software-type* and *mention-type* classifier modules, have been removed resulting into a 2-cascade classifier-module as shown on the figure 6.7. The *software-entity* classifier module, however, is kept because identification of software would assist software purpose classification.

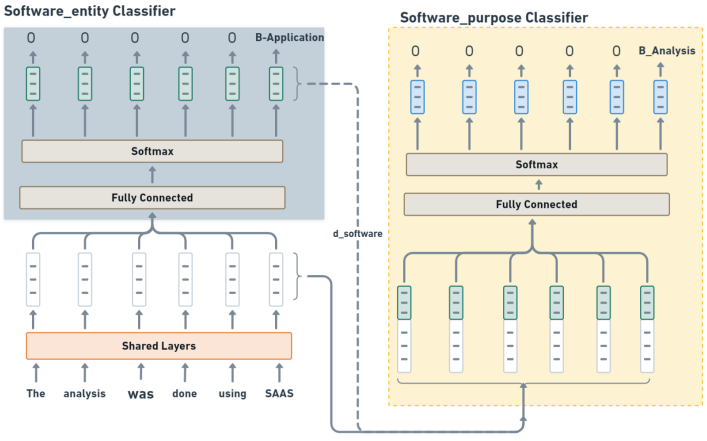


Figure 6.7: Fully connected, 2-cascade *software-entity* and *software-purpose* classifier model.

Evaluation of the 2-cascade Sci-BERT classifier model reveals that overall *software purpose* classification F-score has shown small improvement compared to the original 4-cascade classifier SCI-BERT model, where as the *software-entity* classifier's performance does not indicate any performance improvement as shown in figures 6.8 and 6.9 respectively. Comparison of software purpose classification performance in terms of F-score (F), Precision (P) and Recall (R) for 2-Cascade Sci-BERT classifier versus 4-cascade Sci-BERT classifier is shown on the table 6.3.

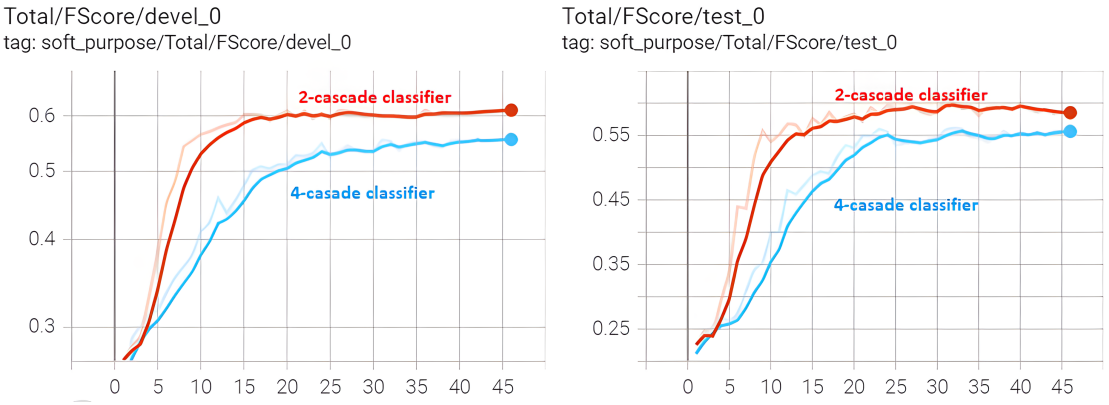


Figure 6.8: 2-cascade classifier has slightly better (Total) F-score over devl. set(left) and test set(right) for software *purpose classification*.

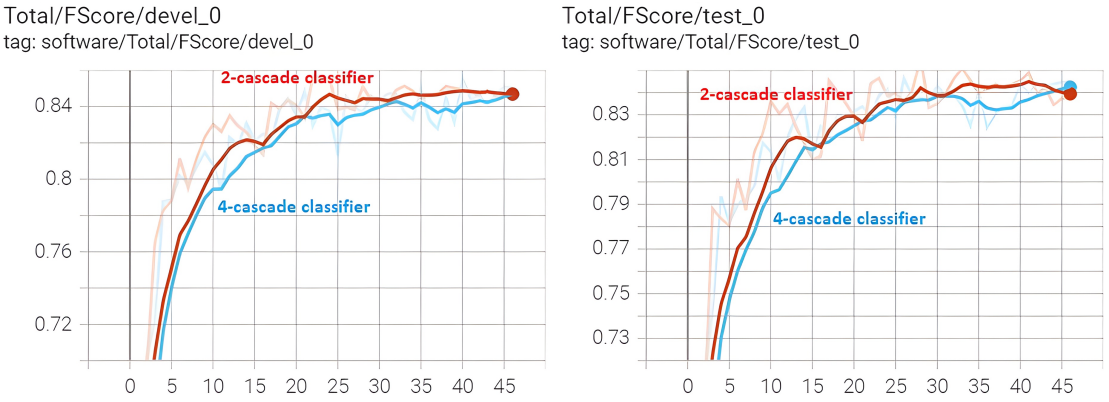


Figure 6.9: 2-cascade classifier has no significant performance improvement in terms of (Total) F-score over devl. set(left) and test set(right) for *software classification*.

The evaluation of classifier's performance by removing the intermediate classifier modules from the 4-cascade model gives a marginal improvement for *software purpose* classification, but not for *software-entity* classification. This might point out, probably flawed classifications of intermediate layers might have an impact on the software purpose classifier which lies at the end of the classification cascade.

Since a 2-cascade Sci-BERT *software-purpose* classifier has slightly better performance, for the next steps this model has been used.

6.4 BIO-BERT VS SCI-BERT

As mentioned before, on section 5.3.1, the other variant of BERT model which is trained with SoMeSci dataset, in this thesis , is Sci-BERT. Even though both BIO-BERT and SCI-BERT give a contextualized representation of a word in a sentence, representations of a word would differ due to the inherent difference of corpora used for pre-trained models of Bio-BERT and Sci-BERT [7, 38]. For this reason, a 2-cascade classifier model has been trained with Bio-BERT and results are compared with Sci-BERT model.

Among various varieties of Bio-BERT models, Bio-BERT(base) and Bio-BERT(large) models have been trained with SoMeSci dataset to compare performance with Sci-BERT model.

The results of evaluation indicate that Bio-BERT-large[3] model performed slightly better than the Sci-BERT[4] model according to the total F-score. However, Sci-BERT has superior performance compared to the Bio-BERT-base[5] model.

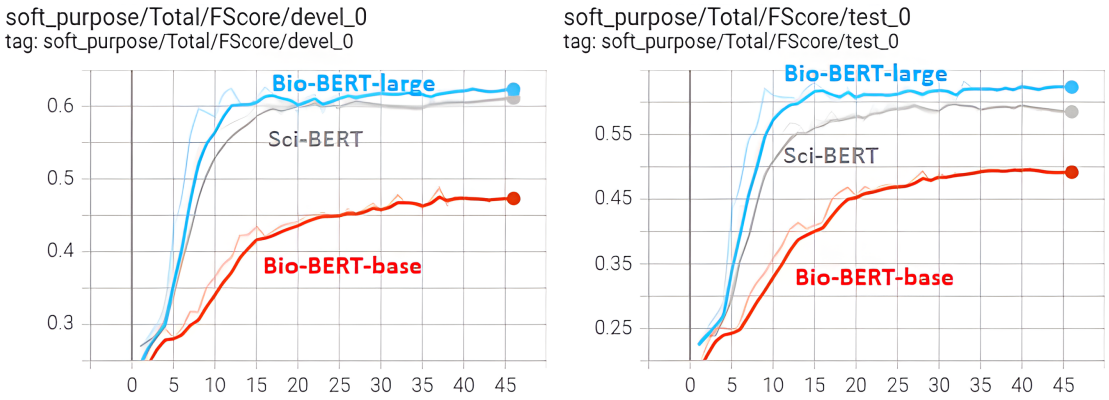


Figure 6.10: 2-Cascade *software-purpose* classifier, trained with Bio-BERT-large, Sci-BERT and Bio-BERT-base indicates that the Bio-BERT-large has superior (Total) F-score over Sci-BERT as well as Bio-BERT-base when tested with devl-set(left) and test-set(right).

Even though Bio-BERT-large software purpose classifier has slightly better total F-score, the model was too slow during a training. The Sci-BERT 2-cascade *software-entity* and *software-purpose* classifier model gives reasonable classification performance with a relatively fast training.

3 https://huggingface.co/dmis-lab/biobert-large-cased-v1.1
4 https://huggingface.co/allenai/scibert_scivocab_cased
5 https://huggingface.co/dmis-lab/biobert-base-cased-v1.2

Table 6.1: Evaluation of software purpose classifier's performance with(+) and without(-) *Creation/PLoS*sentences in the training dataset.

Software Purpose	Metric	Dev+	Dev-	Test+	Test-
Analysis	F	0.64	0.71	0.65	0.69
	P	0.61	0.71	0.60	0.66
	R	0.68	0.71	0.72	0.73
Data-Collection	F	0.26	0.32	0.26	0.34
	P	0.25	0.33	0.22	0.30
	R	0.28	0.31	0.30	0.31
Pre-Processing	F	0.41	0.45	0.49	0.57
	P	0.36	0.37	0.45	0.49
	R	0.48	0.56	0.58	0.67
Modeling	F	0.25	0.54	0.30	0.44
	P	0.27	0.54	0.28	0.42
	R	0.24	0.55	0.30	0.46
Programming	F	0.27	0.42	0.23	0.42
	P	0.28	0.34	0.24	0.36
	R	0.27	0.55	0.23	0.51
Simulation	F	0.00	0.38	0.00	0.00
	P	0.00	0.88	0.00	0.00
	R	0.00	0.25	0.00	0.00
Stimulation	F	0.46	0.59	0.24	0.26
	P	0.40	0.76	0.21	0.24
	R	0.48	0.45	0.28	0.22
Visualization	F	0.48	0.59	0.52	0.58
	P	0.46	0.59	0.46	0.54
	R	0.51	0.61	0.56	0.62
Total (software Purpose)	F	0.27	0.58*	0.30	0.54*
	P	0.26	0.62*	0.26	0.56*
	R	0.31	0.55*	0.32	0.65*
Total (software)	F	0.86*	0.84	0.83	0.83
	P	0.83*	0.82	0.77	0.79*
	R	0.90*	0.86	0.90*	0.87

Table 6.2: Comparison of 4-cascade multi-class Sci-BERT software purpose classifier's F-score, Precision and Recall performance for various levels of context information.

Software Purpose	Metric	Dv.(2,0)	Dv.(0,2)	Dv.(2,2)	Tst(2,0)	Tst(0,2)	Tst(2,2)
Analysis	F	0.72	0.66	0.69	0.70	0.68	0.69
	P	0.72	0.64	0.68	0.65	0.63	0.64
	R	0.73	0.68	0.71	0.76	0.74	0.74
Data-Collection	F	0.29	0.31	0.25	0.38	0.20	0.31
	P	0.29	0.29	0.24	0.38	0.22	0.32
	R	0.29	0.32	0.25	0.39	0.20	0.29
Pre-Processing	F	0.45	0.45	0.41	0.58	0.58	0.52
	P	0.36	0.36	0.37	0.49	0.48	0.44
	R	0.59	0.59	0.52	0.73	0.73	0.64
Modeling	F	0.50	0.50	0.38	0.42	0.42	0.31
	P	0.42	0.42	0.41	0.37	0.37	0.30
	R	0.62	0.62	0.35	0.48	0.48	0.31
Programming	F	0.42	0.41	0.44	0.41	0.33	0.42
	P	0.32	0.32	0.34	0.36	0.28	0.37
	R	0.61	0.57	0.64	0.48	0.42	0.51
Simulation	F	0.24	0.00	0.25	0.00	0.00	0.00
	P	0.33	0.00	0.39	0.00	0.00	0.00
	R	0.19	0.00	0.20	0.00	0.00	0.00
Stimulation	F	0.47	0.34	0.39	0.34	0.41	0.24
	P	0.56	0.42	0.51	0.26	0.37	0.20
	R	0.41	0.32	0.31	0.50	0.47	0.32
Visualization	F	0.58	0.36	0.54	0.60	0.40	0.50
	P	0.60	0.35	0.54	0.57	0.41	0.46
	R	0.56	0.38	0.53	0.64	0.41	0.54
Total	F	0.58	0.53	0.55	0.59	0.54	0.55
	P	0.57	0.51	0.54	0.54	0.51	0.51
	R	0.62	0.55	0.57	0.66	0.59	0.60

Table 6.3: Evaluation of software purpose classifier's performance with 2-cascade Sci-BERT model (2C) versus 4-cascade Sci-BERT model (4C) shows that the 2C-model has slightly better performance compared to 4C-model.

Software Purpose	Metric	Dev.(2C)	Dev.(4C)	Test.(2C)	Test(4C)
Analysis	F	0.72	0.69	0.69	0.68
	P	0.73	0.68	0.65	0.64
	R	0.71	0.70	0.74	0.73
Data-Collection	F	0.34	0.24	0.35	0.25
	P	0.36	0.23	0.41	0.20
	R	0.32	0.25	0.31	0.25
Pre-Processing	F	0.49	0.39	0.57	0.55
	P	0.40	0.32	0.48	0.46
	R	0.61	0.50	0.68	0.73
Modeling	F	0.64	0.47	0.43	0.38
	P	0.57	0.48	0.42	0.38
	R	0.71	0.46	0.44	0.38
Programming	F	0.41	0.46	0.32	0.37
	P	0.32	0.36	0.28	0.32
	R	0.55	0.55	0.38	0.44
Simulation	F	0.28	0.28	0.00	0.00
	P	0.66	0.64	0.00	0.00
	R	0.18	0.18	0.00	0.00
Stimulation	F	0.50	0.48	0.25	0.23
	P	0.56	0.52	0.21	0.29
	R	0.44	0.45	0.31	0.20
Visualization	F	0.60	0.45	0.69	0.57
	P	0.59	0.43	0.62	0.56
	R	0.61	0.47	0.78	0.59
Total	F	0.61*	0.55	0.56*	0.55
	P	0.61*	0.51	0.54*	0.52
	R	0.62*	0.58	0.64*	0.60

7 CONCLUSION AND FUTURE WORK

7.1 SUMMARY

Evaluation of BERT models used in this thesis, Sci-BERT and Bio-BERT, revealed important observations. Firstly, the analysis of Sci-BERT *software-purpose* classification indicates that composition of training dataset has significant impact on the software purpose classification. Overall, when Sci-BERT classifier model is trained by including parts of the SoMeSci dataset that do not have software purpose annotations, software purpose classification performance is deteriorated. However, classification performance for software purpose is increased when the Sci-BERT model is trained with parts of SoMeSci dataset that have software-purpose annotations. This result clearly indicates, including unlabeled dataset in the training data degrades software-purpose classification performance.

Secondly, it has been observed that various levels of context can affect software purpose classification performance. Training of a classifier model based on neighboring sentences, in a scientific articles of SoMeSci dataset, has been seen to improve software purpose classification performance. Typically, a context information with two sentences at the left-side (2,0) of a given sentence appears to give more important clue for the Sci-BERT *software-purpose* classifier. However, unbounded context information such as the whole paragraph or a context not limited within a paragraph did not provide better classification performance compared to context information limited to only two sentences from the left-side(2,0) as shown on the figure 6.4.

Thirdly, evaluation of the Sci-BERT model's performance by removing the intermediate modules of *software-type* and *mention-type* classifiers from the 4-cascade classifier shown on fig. 5.6, produced a slightly better performance for *software purpose* classification. This indicates that flawed classifications, from the middle classifier modules of the 4-cascade, potentially impact *software-purpose* classifier's performance which lies at the end of cascade of classifier modules.

Further more, it was observed that training other variants of BERT model such as Bio-BERT-base and Bio-BERT-large model results in a different software-purpose classification performance. The Bio-BERT-large model has slightly better performance compared with Sci-BERT but it is too slow for training. On the other hand Bio-BERT-small trains fast but its software purpose classification performance is lower than Sci-BERT.

Generally, it is also observed that *software purpose* classification performance is greater for software purposes with larger number of class labels such as *analysis* compared to other software usage purposes like *stimulation or simulation* which has smaller number of class labels.

7.2 CONCLUSION

The role of software in the modern scientific research is paramount. However, the use of software in a research entailed problems such as research result reliability and reproducibility. Automatically extracted information about software, such as software-purpose, can help to mitigate these problems by identifying a set of related software tools in terms of purpose of usage. Results from related software tools can be compared to cross-validate and ensure reliability of research outcomes.

Automatic classification of software-purposes have been possible with availability of labeled datasets and state-of-the-art deep learning models, such as BERT. BERT models are efficient for software purpose classification because of their capability for transfer-learning and does not require a lot of training data, unlike LSTM networks.

Results of software-purpose classification indicates reasonable classification performance despite limited availability of labeled datasets of SoMeSci and unbalanced class labels for software purposes. Particularly, the classifier model performed significantly better for software purposes whith larger number of class labels such as Analysis.

The analysis of the SoMeSci dataset also, revealed most scientists use software to carry out specific tasks in a research, fewer software-purposes, as shown on the figure 4.12. This concurs with the fact that most scientists develop software for their scientific work.

7.3 LIMITATIONS AND FUTURE WORK

In this thesis, for automatic classification of software-purposes only part of the SoMeSci dataset was annotated because of time constraint. Because of this for most software-purposes, such as simulation and stimulation, there were no enough examples for training of classifier model.

To increase software purpose classification performance, in the future, the remaining part of SoMeSci dataset can also be annotated to increase software purpose classification performance. In addition, the SoMeSci corpus size could also be increased slightly to increase software-purpose classification. Further more, annotation of software-usage purposes can also be done by domain experts to increase correct classification of software-purpose classification.

Based on extracted information about software purposes, a set of similar software tools can also be identified from scientific articles to cross-validate results of a research using software tools that are used for the same purpose. Further more, semantic browsing of scientific articles based on use of software for the same purpose can also be implemented.

Part II

APPENDIX

APPENDIX - A : WIKI DATA SPARQL QUERIES

Listing A.1: SARQL Query for software categories

```
SELECT

?Software_Category
?sc_parent
?Software_CategoryLabel
?sc_parentLabel

(COUNT(DISTINCT ?software) AS ?num_software)

WHERE {
        ?Software_Category (wdt:P279*) wd:Q7397;
        wdt:P31 wd:Q17155032;
        wdt:P279 ?sc_parent.

        OPTIONAL { ?software wdt:P31 ?Software_Category. }
        SERVICE wikibase:label { bd:serviceParam wikibase:language "[
            AUTO_LANGUAGE],en". }
}
GROUP BY ?Software_Category ?Software_CategoryLabel ?sc_parent ?
    sc_parentLabel
```

Listing A.2: SPARQL query of software counts in each category

```
SELECT

?Software_Category
?Software_CategoryLabel

(COUNT(DISTINCT ?software) AS ?num_software)

WHERE {
        ?Software_Category (wdt:P279*) wd:Q7397; wdt:P31 wd:Q17155032.
        OPTIONAL { ?software wdt:P31 ?Software_Category. }
        SERVICE wikibase:label { bd:serviceParam wikibase:language "[
            AUTO_LANGUAGE],en". }
}
GROUP BY ?Software_Category ?Software_CategoryLabel
ORDER BY DESC (?num_software)
```

APPENDIX - B : FINDING FILES WITH ANNOTATION ERROR

Listing B.1: Finding file names that have errors with software usage purpose labels in PLoS files

```
Usage_labels = ['Application_Usage','ProgrammingEnvironment_Usage', ...]
Purpose_lables = ['Analysis','Modelling','Stimulation','DataCollection', ...]

path = 'SoMeSci/PLoS_methods/'
for file_name in os.listdir(path):

        if file_name.endswith('.ann'):
                file_path = path + file_name
                with open(file_path, "r") as a_file:
                        anno_usage_strstp, anno_purpose_strstp = [], []
                        for line in a_file:
                                annotataion = line.split()

                                if (annotataion[1] in Usage_):
                                        startstop = annotataion[2:4]
                                        strstp = '_'.join(startstop)
                                        anno_usage_strstp.append(strstp)

                                elif (annotataion[1] in Purpose_):
                                        startstop = annotataion[2:4]
                                        strstp = '_'.join(startstop)
                                        anno_purpose_strstp.append(strstp)

                Usage_ann_set = set(anno_usage_strstp)
                Purpose_ann_set = set(anno_purpose_strstp)

                # Usage_ann_set - Purpose_ann_set = Usage with no purpose ann
                missingPurposeANN = Usage_ann_set.difference(Purpose_ann_set)
                missingUsageANN = Purpose_ann_set.difference(Usage_ann_set)

                # Error-1: lack PURPOSE ANNOTATION
                if(len(missingPurposeANN)!= 0):
                print (file_name, missingPurposeANN)
                # Error-2: have undesired Purpose ANNOTATION
                if(len(missingUsageANN)!= 0):
                print (file_name, missingUsageANN)
```

APPENDIX - C : MERGING SOFTWARE USAGE AND PURPOSE LABELS

Listing C.1: Function that merges software usage and purpose labels

```
def mergeList(list_1, list_2):

        #stores merged annotations
        result_list = []
        checked_list = []

        for x, y in [(x,y) for x in list_1[1].split()[1:] for y in list_2[1].split
            ()[1:]]:
                if( x == y):
                        #  avoid duplicate merging
                        if x not in checked_list:

                                #get id of the firt annotation
                                result_list.append(list_1[0]

                                # get annotation starting_number stop_number
                                annotationStrStp = []
                                #merge annotations by _
                                annotationStrStp.append(list_1[1].split()[0]+'_'+
                                    list_2[1].split()[0].split('_')[1])

                                # start number
                                annotationStrStp.append(list_1[1].split()[1])

                                #end number
                                annotationStrStp.append(list_1[1].split()[2])

                                # Join annotation ,
                                annStrStpJOin = ' '.join(annotationStrStp)
                                result_list.append(annStrStpJOin)

                                # get name of the software
                                software_name = list_1[2]
                                result_list.append(software_name)
                                checked_list.extend(list_1[1].split()[1:])
                else:
                        pass
        return result_list
```

APPENDIX - D : DATA SPLIT OPTIMIZATION

Listing D.1: Function to retrieve counts of class labels in each data split

```
import os
from os import listdir
from collections import Counter

def purposeLabel_counter(path):

        def list_file_names(path, file_ext='.labels.txt'):
                file_names_list = []
                for file_name in os.listdir(path):
                        if not file_name.endswith(file_ext): continue
                        file_names_list.append(file_name)
                file_names_list.sort()
                return file_names_list

        file_name_list = list_file_names(path)

        interest_list = ["Analysis", "Modelling", "Stimulation", "DataCollection",
             "DataPreProcss","Simulation", "Visualization", "Programming"]

        all_purpose_labels = []

        for file_name in file_name_list[:]:
                file_path = path + file_name

                with open(file_path, 'r') as f:
                        list_lines = f.readlines()
                        list_of_tokens = ' '.join(list_lines).split()

                        for tok in list_of_tokens:
                                if tok != 'O' and (len(tok.split('-')[1].split('_'
                                    )) == 3 ):
                                        if tok.split('-')[1].split('_')[2] in
                                            interest_list:
                                                purpose = tok.split('-')[1].split(
                                                    '_')[2]
                                                    all_purpose_labels.append(
                                                        purpose)
        return  dict(Counter(all_purpose_labels))
```

Listing D.2: Function to retrieve software usage purpose class labels in whole-document, train, test, and development list

```
def train_test_dev(whole_path, train_path, test_path, dev_path):

        _whole_sorted = dict(sorted(purposeLabel_counter(whole_path).items()))
        _train_sorted = dict(sorted(purposeLabel_counter(train_path).items()))
        _test_sorted  = dict(sorted(purposeLabel_counter(test_path).items()))
        _dev_sorted   = dict(sorted(purposeLabel_counter(dev_path).items()))

        return _whole_sorted, _train_sorted, _test_sorted, _dev_sorted
```

Listing D.3: Function to determine the proportion of each class label in the split data sets of train, test and development

```
def __proportion_calculator(whole_dict, train_dict, test_dict, dev_dict ):

        train_purpose_props = []
        for (k,v), (k2,v2) in zip(whole_dict.items(), train_dict.items()):
                if k2 == k:
                        train_percentage = int(v2 / v * 100)
                        train_purpose_props.append(train_percentage)

        dev_purpose_props = []
        for (k,v), (k2,v2) in zip(whole_dict.items(), dev_dict.items()):
                if k2 == k:
                        dev_percentage = int(v2 / v * 100)
                        dev_purpose_props.append(dev_percentage)

        test_purpose_props = []
        for (k,v), (k2,v2) in zip(whole_dict.items(), test_dict.items()):
                if k2 == k:
                        test_percentage = int(v2 / v * 100)
                        test_purpose_props.append(test_percentage)

        return train_purpose_props, dev_purpose_props, test_purpose_props
```

Listing D.4: Check if the data split is within +/- 5% for training set

```
def _within_range(list_):

        bool_list = []
        for val in list_:
                if val in range(55,65):
                        bool_list.append(True)
                else:
                        bool_list.append(False)

        return all(bool_list), bool_list
```

Listing D.5: Check if the data split is within +/- 5% for test and developments set

```
def _within_range_dev_test(list_):
        bool_list = []
        for val in list_:
                if val in range(15,25):
                        bool_list.append(True)
                else:
                        bool_list.append(False)
                return all(bool_list), bool_list
```

APPENDIX - E : CONTEXT READING

Listing E.1: Truncated python code for reading left and right sentences for context limited within a paragrapgh

```
def _read_text_file(path, bef, aft, read_empty=False):

        def add_neighbours(*sentcs):
                lis_sents = list(sentcs)
                joined_tokens = []
                for i in range(len(lis_sents)):
                        joined_tokens.extend(lis_sents[i].split())
                        joined_sent = ' '.join(joined_tokens)

        return joined_sent

        def contextWindow(text, bef, aft):

                if (bef >= len(text)-1) or (aft >= len(text)-1):
                        bef = len(text)- 1
                        aft = len(text)- 1
                        contxt_txt = []

                        for i in range(len(text)):
                                contxt_txt.append( ' '.join(text))
                        return contxt_txt

                 elif (bef == 0) and (aft == 2):
                        contxt_txt = []
                        for i in range(len(text)):
                                #OB, 2A
                                if (i >= 0) and (i <len(text)-2):
                                        contxt_txt.append( add_neighbours(text[i],
                                             text[i+1], text[i+2]))

                                elif i == (len(text)-2):
                                        contxt_txt.append( add_neighbours(text[i],
                                             text[i+1] ))

                                elif i == (len(text)-1):
                                        contxt_txt.append(text[i])

                return contxt_txt

                # context 2 sentences before and after
                elif (bef == 2) and (aft == 2):
```

```
                contxt_txt = []
                for i in range(len(text)):
                        if i == 0:
                                contxt_txt.append(add_neighbours(text[i] ,
                                     text[i+1] , text[i+2]))
                        elif i == 1:
                                contxt_txt.append(add_neighbours(text[i-1]
                                     , text[i] , text[i+1] , text[i+2]))

                        elif (i >=1) and (i < len(text)-2):
                                contxt_txt.append(add_neighbours(text[i
                                    -2], text[i-1], text[i], text[i+1],
                                    text[i+2]))

                        elif (i == len(text)-2):
                                contxt_txt.append(add_neighbours(text[i
                                    -2], text[i-1], text[i], text[i+1]))

                        elif i == len(text)-1:
                                contxt_txt.append(add_neighbours(text[i
                                    -2], text[i-1], text[i]))

                return contxt_txt

        txt_with_context = []
        with path.open(mode='r') as in_f:

        # each paragraphs separated by new line
        cont = in_f.read()

        #list of paragraphs
        par_list = cont.split('\n\n')

        for par_ in par_list:
                snt_lst_ = par_.split('\n')
                snt_lst_ = list(filter(None, snt_lst_))

                # sentence with context
                sen_contkx = contextWindow(snt_lst_, bef, aft)
                txt_with_context.extend(sen_contkx)
                return txt_with_context
```

BIBLIOGRAPHY

[1] Charu C Aggarwal. *Machine learning for text*. Vol. 848. Springer, 2018.

[2] Sea Agostinelli, John Allison, K al Amako, John Apostolakis, H Araujo, Pedro Arce, Makoto Asai, D Axen, Swagato Banerjee, GJNI Barrand, et al. "GEANT4—a simulation toolkit." In: *Nuclear instruments and methods in physics research section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 506.3 (2003), pp. 250–303.

[3] Adnan Akhundov, Dietrich Trautmann, and Georg Groh. "Sequence labeling: A practical approach." In: *arXiv preprint arXiv:1808.03926* (2018).

[4] Mokhled S Al-Tarawneh. "Lung cancer detection using image processing techniques." In: *Leonardo Electronic Journal of Practices and Technologies* 11.21 (2012), pp. 147–58.

[5] Jean-Marie André. "The nobel prize in chemistry 2013." In: *Chemistry International* 36.2 (2014), pp. 2–7.

[6] Monya Baker. "Reproducibility crisis." In: *Nature* 533.26 (2016), pp. 353–66.

[7] Iz Beltagy, Kyle Lo, and Arman Cohan. "SciBERT: A pretrained language model for scientific text." In: *arXiv preprint arXiv:1903.10676* (2019).

[8] Jan Bulla. "Application of hidden Markov models and hidden semi-Markov models to financial time series." In: (2006).

[9] Balakrishnan Chandrasekaran, John R Josephson, and V Richard Benjamins. "What are ontologies, and why do we need them?" In: *IEEE Intelligent Systems and their applications* 14.1 (1999), pp. 20–26.

[10] Event Horizon Telescope Collaboration et al. "First M87 event horizon telescope results. I. The shadow of the supermassive black hole." In: *arXiv preprint arXiv:1906.11238* (2019).

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding." In: *arXiv preprint arXiv:1810.04805* (2018).

[12] Sheperd Doeleman, Eric Agol, Don Backer, Fred Baganoff, Geoffrey C Bower, Avery Broderick, Andrew Fabian, Vincent Fish, Charles Gammie, Paul Ho, et al. "Imaging an event horizon: submm-VLBI of a super massive black hole." In: *arXiv preprint arXiv:0906.3899* (2009).

[13] 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.

[14] 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), pp. 1–13.

[15] Sabry El-Hakim, Lorenzo Gonzo, Francesca Voltolini, Stefano Girardi, Alessandro Rizzi, Fabio Remondino, and Emily Whiting. "Detailed 3D modelling of castles." In: *International journal of architectural computing* 5.2 (2007), pp. 199–220.

[16] Aysu Ezen-Can. "A Comparison of LSTM and BERT for Small Corpus." In: *arXiv preprint arXiv:2009.05451* (2020).

[17] Thomas Fischer and Christopher Krauss. "Deep learning with long short-term memory networks for financial market predictions." In: *European Journal of Operational Research* 270.2 (2018), pp. 654–669.

[18] Paul A Gagniuc. *Markov chains: from theory to implementation and experimentation*. John Wiley & Sons, 2017.

[19] Simson Garfinkel. "History's worst software bugs." In: *Wired News, Nov* (2005).

[20] Shalini Ghosh, Oriol Vinyals, Brian Strope, Scott Roy, Tom Dean, and Larry Heck. "Contextual lstm (clstm) models for large scale nlp tasks." In: *arXiv preprint arXiv:1602.06291* (2016).

[21] Yolanda Gil, Varun Ratnakar, and Daniel Garijo. "OntoSoft: Capturing scientific software metadata." In: *Proceedings of the 8th International Conference on Knowledge Capture*. 2015, pp. 1–4.

[22] Carole Goble. "Better software, better research." In: *IEEE Internet Computing* 18.5 (2014), pp. 4–8.

[23] JS Grethe, A Bandrowski, M Chiu, T Gillespie, J Go, Y Li, IB Ozyurt, and ME Martone. "SciCrunch: A Cooperative And Collaborative Data, Information, And Resource Discovery Portal For Scientific Communities." In: *Front. Neuroinform. Conference Abstract: Neuroinformatics*. 2016.

[24] 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, pp. 1–8.

[25] Zhiyong He, Zanbo Wang, Wei Wei, Shanshan Feng, Xianling Mao, and Sheng Jiang. "A survey on recent advances in sequence labeling from deep learning models." In: *arXiv preprint arXiv:2011.06727* (2020).

[26] Simon Hettrick, Mario Antonioletti, Leslie Carr, Neil Chue Hong, Stephen Crouch, David De Roure, Iain Emsley, Carole Goble, Alexander Hay, Devasena Inupakutika, et al. "Uk research software survey 2014." In: (2014).

[27] Zhiheng Huang, Wei Xu, and Kai Yu. "Bidirectional LSTM-CRF models for sequence tagging." In: *arXiv preprint arXiv:1508.01991* (2015).

[28] Vanessa Ibáñez, Josep Silva, and Omar Cauli. "A survey on sleep assessment methods." In: *PeerJ* 6 (2018), e4849.

[29] Caroline Jay, Robert Haines, and Daniel S Katz. "Software must be recognised as an important output of scholarly research." In: *arXiv preprint arXiv:2011.07571* (2020).

[30] Rafael C Jiménez, Mateusz Kuzak, Monther Alhamdoosh, Michelle Barker, Bérénice Batut, Mikael Borg, Salvador Capella-Gutierrez, Neil Chue Hong, Martin Cook, Manuel Corpas, et al. "Four simple recommendations to encourage best practices in research software." In: *F1000Research* 6 (2017).

[31] Upulee Kanewala and James M Bieman. "Testing scientific software: A systematic literature review." In: *Information and software technology* 56.10 (2014), pp. 1219–1232.

[32] F Krüger and D Schindler. *A literature review on methods for the extraction of usage statements of software and data. IEEE Computing in Science and Engineering (2019)*. 2019.

[33] Max Kuhn and Kjell Johnson. "Data pre-processing." In: *Applied predictive modeling*. Springer, 2013, pp. 27–59.

[34] Julian Kupiec. "Robust part-of-speech tagging using a hidden Markov model." In: *Computer speech & language* 6.3 (1992), pp. 225–242.

[35] John Lafferty, Andrew McCallum, and Fernando CN Pereira. "Conditional random fields: Probabilistic models for segmenting and labeling sequence data." In: (2001).

[36] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. "Neural architectures for named entity recognition." In: *arXiv preprint arXiv:1603.01360* (2016).

[37] 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 (2020), pp. 1234–1240.

[38] Fei Li, Yonghao Jin, Weisong Liu, Bhanu Pratap Singh Rawat, Pengshan Cai, Hong Yu, et al. "Fine-tuning bidirectional encoder representations from transformers (BERT)–based models on large-scale electronic health record notes: an empirical study." In: *JMIR medical informatics* 7.3 (2019), e14830.

[39] Robyn Lutz. "Software engineering for space exploration." In: *Computer* 44.10 (2011), pp. 41–46.

[40] Xuezhe Ma and Eduard Hovy. "End-to-end sequence labeling via bi-directional lstm-cnns-crf." In: *arXiv preprint arXiv:1603.01354* (2016).

[41] Brian Malley, Daniele Ramazzotti, and Joy Tzung-yu Wu. "Data pre-processing." In: *Secondary analysis of electronic health records* (2016), pp. 115–141.

[42] 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* 5.1 (2014), pp. 1–13.

[43] Nicolas Marchand, Philippe Lienard, Hans-Ullrich Siehl, and Harunobu Izato. "Applications of molecular simulation software SCIGRESS in industry and university." In: *FUJITSU Sci Tech J* 50 (2014), pp. 46–51.

[44] Andrew McCallum, Dayne Freitag, and Fernando CN Pereira. "Maximum entropy Markov models for information extraction and segmentation." In: *Icml*. Vol. 17. 2000. 2000, pp. 591–598.

[45] Zeeya Merali. "Computational science:... error." In: *Nature* 467.7317 (2010), pp. 775–777.

[46] Greg Miller. *A scientist's nightmare: software problem leads to five retractions*. 2006.

[47] Veenavati Jagadishprasad Mishra and Manisha D Khardenvis. "Contingency analysis of power system." In: *2012 IEEE Students' Conference on Electrical, Electronics and Computer Science*. IEEE. 2012, pp. 1–4.

[48] Udit Nangia, Daniel S Katz, et al. "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.

[49] Andy R Ness, Sam D Leary, Calum Mattocks, Steven N Blair, John J Reilly, Jonathan Wells, Sue Ingle, Kate Tilling, George Davey Smith, and Chris Riddoch. "Objectively measured physical activity and fat mass in a large cohort of children." In: *PLoS medicine* 4.3 (2007), e97.

[50] Isaiah Norton, Walid Ibn Essayed, Fan Zhang, Sonia Pujol, Alex Yarmarkovich, Alexandra J Golby, Gordon Kindlmann, Demian Wassermann, Raul San Jose Estepar, Yogesh Rathi, et al. "SlicerDMRI: open source diffusion MRI software for brain cancer research." In: *Cancer research* 77.21 (2017), e101–e103.

[51] NumFOCUS. *NumFOCUS tools help create the first ever image of a black hole*. [Online; accessed 08-December-2021]. URL: https://numfocus.org/case-studies/first-photograph-black-hole.

[52] Xuelian Pan, Erjia Yan, and Weina Hua. "Disciplinary differences of software use and impact in scientific literature." In: *Scientometrics* 109.3 (2016), pp. 1593–1610.

[53] Hemlata Pande and Hoshiyar S Dhami. "Mathematical Modelling of Occurrence of Letters and Word's Initials in Texts of Hindi Language." In: *SKASE Journal of Theoretical Linguistics* 7.2 (2010).

[54] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks." In: *International conference on machine learning*. PMLR. 2013, pp. 1310–1318.

[55] Andrew Proctor and Peter MA Sherwood. "Data analysis techniques in x-ray photoelectron spectroscopy." In: *Analytical Chemistry* 54.1 (1982), pp. 13–19.

[56] Åsmund Rinnan, Lars Nørgaard, Frans van den Berg, Jonas Thygesen, Rasmus Bro, and Søren Balling Engelsen. "Data pre-processing." In: *Infrared spectroscopy for food quality analysis and control* (2009), pp. 29–50.

[57] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter." In: *arXiv preprint arXiv:1910.01108* (2019).

[58] Maartje Schermer. "Ethical issues in deep brain stimulation." In: *Frontiers in integrative neuroscience* 5 (2011), p. 17.

[59] 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: *arXiv preprint arXiv:2108.09070* (2021).

[60] 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.

[61] 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)*. IEEE. 2019, pp. 52–57.

[62] Judith Segal and Chris Morris. "Developing scientific software." In: *IEEE software* 25.4 (2008), pp. 18–20.

[63] David AW Soergel. "Rampant software errors may undermine scientific results." In: *F1000Research* 3 (2014).

[64] Tim Storer. "Bridging the chasm: A survey of software engineering practice in scientific programming." In: *ACM Computing Surveys (CSUR)* 50.4 (2017), pp. 1–32.

[65] 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* 30 (2017).

[66] Denny Vrandečić. "Wikidata: A new platform for collaborative data collection." In: *Proceedings of the 21st international conference on world wide web*. 2012, pp. 1063–1064.

[67] Paul D Wagner, Mukesh Verma, and Sudhir Srivastava. "Challenges for biomarkers in cancer detection." In: *Annals of the New York academy of sciences* 1022.1 (2004), pp. 9–16.

[68] Hanna M Wallach. "Conditional random fields: An introduction." In: *Technical Reports (CIS)* (2004), p. 22.

[69] Wikipedia contributors. *Stimulation — Wikipedia, The Free Encyclopedia*. [Online; accessed 30-December-2021]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Stimulation&oldid=976395276.

[70] Wikipedia contributors. *Comprehensive Nuclear-Test-Ban Treaty — Wikipedia, The Free Encyclopedia*. [Online; accessed 7-December-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Comprehensive_Nuclear-Test-Ban_Treaty&oldid=1053274189.

[71] Wikipedia contributors. *Computer programming — Wikipedia, The Free Encyclopedia*. [Online; accessed 30-December-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Computer_programming&oldid=1062649903.

[72] Wikipedia contributors. *Data analysis — Wikipedia, The Free Encyclopedia*. [Online; accessed 29-December-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Data_analysis&oldid=1061024140.

[73] Wikipedia contributors. *Data cleansing — Wikipedia, The Free Encyclopedia*. [Online; accessed 29-December-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Data_cleansing&oldid=1051181443.

[74] Wikipedia contributors. *Data collection — Wikipedia, The Free Encyclopedia*. [Online; accessed 29-December-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Data_collection&oldid=1049936190.

[75] Wikipedia contributors. *Data pre-processing — Wikipedia, The Free Encyclopedia*. [Online; accessed 29-December-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Data_pre-processing&oldid=1059558941.

[76] Wikipedia contributors. *Data visualization — Wikipedia, The Free Encyclopedia*. [Online; accessed 29-December-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Data_visualization&oldid=1059912747.

[77] Wikipedia contributors. *Imputation (statistics) — Wikipedia, The Free Encyclopedia*. [Online; accessed 29-December-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Imputation_(statistics)&oldid=1056727993.

[78] Wikipedia contributors. *Model — Wikipedia, The Free Encyclopedia*. [Online; accessed 11-February-2022]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Model&oldid=1058944086.

[79] Wikipedia contributors. *Ontology (information science) — Wikipedia, The Free Encyclopedia*. [Online; accessed 22-December-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Ontology_(information_science)&oldid=1060388948.

[80] Wikipedia contributors. *Scientific method — Wikipedia, The Free Encyclopedia*. [Online; accessed 20-December-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Scientific_method&oldid=1061107378.

[81] Wikipedia contributors. *Scientific modelling — Wikipedia, The Free Encyclopedia*. [Online; accessed 30-December-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Scientific_modelling&oldid=1051627717.

[82] Wikipedia contributors. *Scigress — Wikipedia, The Free Encyclopedia*. [Online; accessed 25-March-2022]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Scigress&oldid=1051856557.

[83] Wikipedia contributors. *Simulation — Wikipedia, The Free Encyclopedia*. [Online; accessed 29-December-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Simulation&oldid=1061669086.

[84] Wikipedia contributors. *Wikidata — Wikipedia, The Free Encyclopedia*. [Online; accessed 20-December-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Wikidata&oldid=1060408581.

[85] Wikipedia contributors. *Wikimedia Foundation — Wikipedia, The Free Encyclopedia*. [Online; accessed 20-December-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Wikimedia_Foundation&oldid=1060114687.

[86] Wikipedia contributors. *Invenio — Wikipedia, The Free Encyclopedia*. [Online; accessed 25-March-2022]. 2022. URL: https://en.wikipedia.org/w/index.php?title=Invenio&oldid=1075310021.

[87] Wikipedia contributors. *Software — Wikipedia, The Free Encyclopedia*. [Online; accessed 18-March-2022]. 2022. URL: https://en.wikipedia.org/w/index.php?title=Software&oldid=1076010620.

[88] Greg Wilson, Dhavide A Aruliah, C Titus Brown, Neil P Chue Hong, Matt Davis, Richard T Guy, Steven HD Haddock, Kathryn D Huff, Ian M Mitchell, Mark D Plumbley, et al. "Best practices for scientific computing." In: *PLoS biology* 12.1 (2014), e1001745.

[89] Stephen Wolfram. "Computer software in science and mathematics." In: *Scientific American* 251.3 (1984), pp. 188–203.

[90] Huiping Yan and Susan Yatabe. "Case studies of nuclear research software development." In: *CNL Nuclear Review* 8.1 (2017), pp. 35–51.

[91] Bo Yang, Ronald Rousseau, Xue Wang, and Shuiqing Huang. "How important is scientific software in bioinformatics research? A comparative study between international and Chinese research communities." In: *Journal of the Association for Information Science and Technology* 69.9 (2018), pp. 1122–1133.