OULUN YLIOPISTO
UNIVERSITY of OULU

Using Multiclass Classification Algorithms to Improve Text Categorization Tool - NLoN

University of Oulu
Department of Information Processing Science
Master’s Thesis
Jianwen Xu
2021-07-22

Abstract

Natural language processing (NLP) and machine learning techniques have been widely utilized in the mining software repositories (MSR) field in recent years. Separating natural language from source code is a pre-processing step that is needed in both NLP and the MSR domain for better data quality. This paper presents the design and implementation of a multi-class classification approach that is based on the existing open-source R package Natural Language or Not (NLoN).

This article also reviews the existing literature on MSR and NLP. The review classified the information sources and approaches of MSR in detail, and also focused on the text representation and classification tasks of NLP. In addition, the design and implementation methods of the original paper are briefly introduced.

Regarding the research methodology, since the research goal is technology-oriented, *i.e.*, to improve the design and implementation of existing technologies, this article adopts the design science research methodology and also describes how the methodology was adopted.

This research implements an open-source Python library, namely NLoN-PY. This is an open-source library hosted on GitHub, and users can also directly use the tools published to the PyPI library.

Since NLoN has achieved comparable performance on two-class classification tasks with the Lasso regression model, this study evaluated other multi-class classification algorithms, *i.e.*, Naive Bayes, k-Nearest Neighbours, and Support Vector Machine. Using 10-fold cross-validation, the expanded classifier achieved AUC performance of 0.901 for the 5-class classification task and the AUC performance of 0.92 for the 2-class task.

Although the design of this study did not show a significant performance improvement compared to the original design, the impact of unbalanced data distribution on performance was detected and the category of the classification problem was also refined in the process. These findings on the multi-class classification design can provide a research foundation or direction for future research.

Keywords
Natural language processing, machine learning, multi-class classification, mining software repositories, NLoN, text categorization.

Supervisor
D.Sc. Professor Mika Mäntylä

Foreword

I want first to thank my supervisor, Professor Mika Mäntylä, for both your empirical expertise and theoretical support help me clarify the research direction and classification methodology.

I would also like to thank Maëlick Claes and all the members of the NLP-TD project, communicating with you helped me understand the practical use of existing technology. Although it was only a remote meeting, in the current pandemic, being able to see everyone's faces also gave me a lot of encouragement.

Finally, I want to thank my wife Yixian, who has been with me and supported me all the way. Without your encouragement, it would be difficult for me to face all these challenges.

Jianwen Xu

Oulu, July 22, 2021

Contents

1. Introduction

Software repositories may contain various valuable data, such as instructions on software design and maintenance, solutions to certain stack traces, and comments related to technical debt. But these data are often a mixture of natural language and source code, separating natural language and non-natural language is an essential pre-task before proceeding to a more in-depth natural language processing. At the same time, different software repositories often have different mixing styles, *e.g.*, in StackOverflow, the questioner often mixes natural language with stack traces or log information, while the answerer often mixes natural language and source code. Therefore, this problem may require a multi-class classification solution.

Previous research has given a solid foundation on two-class classification, Mäntylä, Calefato, and Claes (2018) proposed a machine learning approach NLoN (Natural Language or Not) for separating natural language from other text inputs that are common in software engineering. Their results suggested a significantly more lightweight design than previous studies, meanwhile, achieved comparable performance in three different information source types, *i.e.*, issue comments from bug repositories, archived chats from online discussions, and email messages from mailing lists.

However, Mäntylä et al. (2018) also pointed out that there is a limitation in the original study, *i.e.*, the solution can only provide a two-class classification (natural language or not). This may lead to an ambiguous result for a mixed statement that consists of both natural language and code in a line of text. Multi-class classification (natural language or not or mixed) and further processing for mixed lines (*e.g.* parse and separate mixed content) could supplement and enhance the original solution.

Moreover, the original study adopted only one machine learning technique Glmnet[1] which implements a fast algorithm that fits generalized linear and similar models via penalized maximum likelihood (Friedman, Hastie, & Tibshirani, 2010). Considering the diversity and rapid development of machine learning algorithms (Bonaccorso, 2017; Jordan & Mitchell, 2015), it is necessary to explore whether other algorithms would provide better results.

Therefore, this study is aimed at improving the accuracy of classification for software engineering text mining by using multi-class classification algorithms. Based on this, this study needs to address two main research questions:

- RQ1: How to design and implement a multi-class classification approach based on previous work?
- RQ2: Whether other algorithms can get better performance than Glmnet without breaking the original principle of lightweight design?

Since the output of the original study is an open-source R project on GitHub[2] (M3SOulu, 2018), this study also focuses on the implementation and performance of an artefact (open-source R package), thus the design science research (DSR) approach will be

[1] https://glmnet.stanford.edu/

[2] https://github.com/M3SOulu/NLoN

adopted as the methodology in this research. This paper followed the guideline of design science research methodology (DSRM) proposed by Peffers, Tuunanen, Rothenberger, and Chatterjee (2007).

To bridge the gap of academic background, a review of the original paper and the related work will be presented in chapter 2. After that, chapter 3 will give a detailed description of the methodology and explain the adoption of DSRM for this project. Next, in chapter 4, the implementation of the study will be described and illustrated. Chapter 5 will present the findings and discuss the results. At last, a conclusion of the study will be summarized in chapter 6.

2. Prior research

This chapter is divided into three sections, *i.e.*, mining software repositories, natural language processing, and the original paper. Section 2.1 reviews the development of mining software repositories (MSR) research, *i.e.*, the main research field of this thesis. Meanwhile, this study also utilized the natural language processing techniques which are introduced in Section 2.2. Next, Section 2.3 gives a comprehensive overview of the original research conducted by Mäntylä et al. (2018).

2.1 Mining software repositories

The research subjects of mining software repositories (MSR) refer to artefacts that are produced and archived during software evolution (Kagdi, Collard, & Maletic, 2007). The MSR aims to discover interesting and actionable information about software systems and projects by analyzing available data from different software repositories (Hassan, 2008). To provide a systematic perception, the taxonomy of MSR is introduced in Section 2.1.1. Hence, software repositories can be diverse, such as source control systems that store the entire history of the source code, defect management systems that retain bugs, improvements, and other issues, and other communication channels such as mailing lists or forums (Robles, 2010; Nazar, Hu, & Jiang, 2016). Thereby, considering the scope of this study (mining text that consists of both natural language and code), the related software repositories and their respective approaches are listed in Subsection 2.1.2.

2.1.1 Taxonomy of MSR

Regarding the taxonomy of MSR approaches, different dimensions were used to compare and contrast as shown in Table 1. Kagdi, Collard, & Maletic (2005) proposed a taxonomy of MSR approaches with four categories including annotation analysis, heuristic, data mining, and differencing. Then, based on the grouping, their work went further to evaluate the specific dimensions of each category. At the same time, another framework that classifies the mining tools for MSR was proposed by German, Cubranić, & Storey (2005).

Table 1. The dimensions of MSR taxonomy.

	Dimensions
Kagdi et al., 2005	• Entity type and granularity • How changes are expressed and defined • Type of MSR question
German et al., 2005	• Intent: the expected users of the tool, and its main objective • Information: the specific sources that the tool mines and the type of analysis made by the tool • Infrastructure: the environment needed to support the tool
Kagdi et al., 2007	• The software repositories utilized: information sources • The purpose of MSR: why or what for • The methodology: how to achieve the purpose of mining • The evaluation of the undertaken approach: how to assess the quality

Whereas it can be found that two studies have their focus which may not be able to give a comprehensive classification for MSR approaches. Therefore, to form a basis for MSR investigations in the context of software evolution, Kagdi et al. (2007) extended their previous work (Kagdi et al., 2005) and presented a layered taxonomy which was derived from a broader survey of MSR approaches. As shown in Figure 1, MSR approaches can be categorized into four layers, *i.e.*, layer 1 (software evolution) and layer 2 (purpose) define the overall context in which a particular MSR investigation is conducted, or an adopted/invented MSR methodology is evaluated; then, layer 3 (representation) refers to the type, granularity, and expression of the artefacts and their differences; finally, layer 4 (information sources) represents the information sources that are readily available in the software repositories and those that need to be made available to support the MSR investigation (Kagdi et al., 2007). Hence, to provide a more intuitive perspective, I will introduce the related MSR approaches from the dimension of information sources.

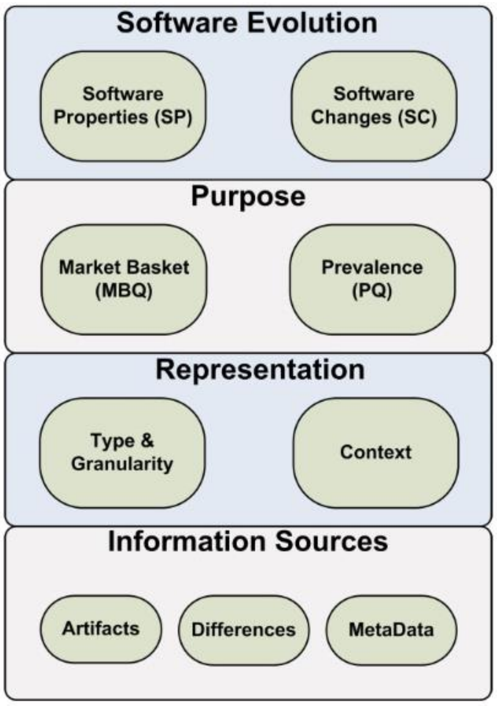


Figure 1. Four-layer taxonomy of MSR approaches (Kagdi et al., 2007, p. 85).

2.1.2 Information sources and approaches

According to Demeyer, Murgia, Wyckmans, and Lamkanfi (2013), the information sources of MSR can be divided into three main categories: source configuration management (SCM), bug tracking systems (BTS), and archived project communications (APC). Additionally, de F. Farias et al. (2016) supplemented some unfamiliar data sources such as binary code, issue tracking system (ITS, similar to BTS but specified to issues reported by customer or user), documents, and logs and subdivided the sources into two subcategories: structured and unstructured repositories (Figure 2).

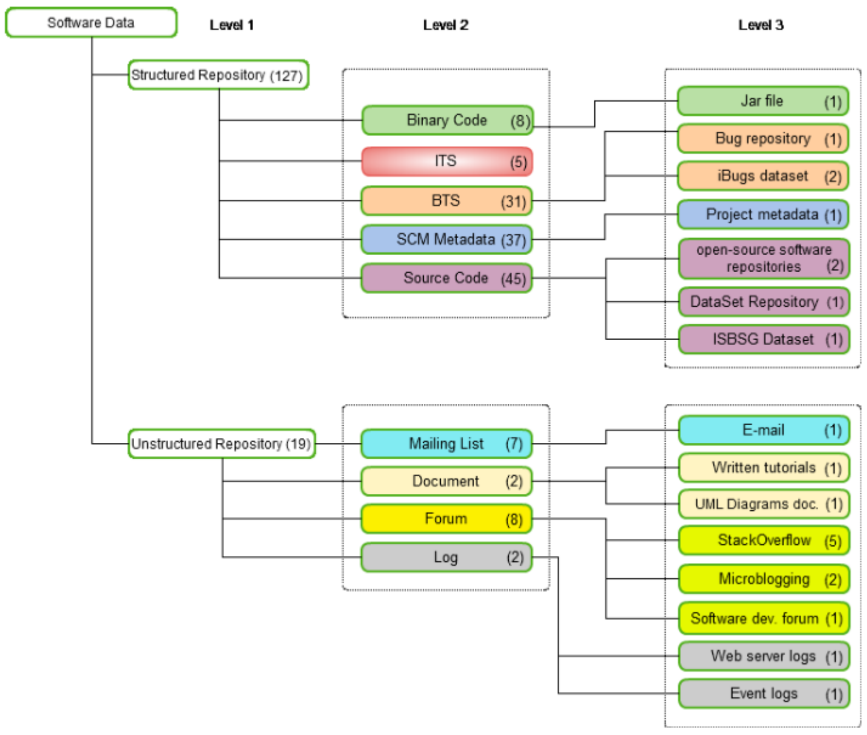


Figure 2. The information source types of MSR (de F. Farias et al., 2016, p. 1477).

Source Code and Version Control Systems

Source code is one of the most important software development artefacts (Kagdi et al., 2007; Hemmati et al., 2013). In general, source code artefacts are stored and managed in source control or version control systems (VCS) (*e.g.* Git[3], Subversion[4]). A VCS can not only store source code but also keep historical versions of source code and project files that are under development (Ruparelia, 2010). Therefore, diverse research purposes can be served due to a source code artefact is a mixed artefact that contains both structured data such as semantics, syntax, and unstructured data such as code comments, identifiers, commit comments (Kagdi et al., 2007; Nazar et al., 2016).

Early researchers have already put effort into searching techniques for source code (Devanbu, 1992; Paul & Prakash, 1994; Chen et al., 2001). Since most programmers started their work by searching for task-relevant code and spent 13% (±6) of time on it (Ko, Aung, & Myers, 2005), more studies have been employed to increase the search effectiveness. Yang and Tan (2012) proposed a general technique to automatically infer semantically related words in software by taking advantage of the context of words in comments and code. They hypothesised that if two words or phrases are used in the same context in comment sentences or identifier names, then they likely have syntactic and semantic relevance. The evaluation proved that their work achieved a reasonable accuracy in seven large and popular code bases written in C and Java. However, their approach identified word pairs related within a given context which may not be commonly related to many applications (Howard, Gupta, Pollock, & Vijay-Shanker, 2013). Thus, a

[3] https://git-scm.com/

[4] https://subversion.apache.org/

complementary study was conducted to automatically mine word pairs that are semantically similar in the software domain by Howard et al. (2013). They leveraged the role of leading comments (a comment before a method signature that provided the reader with the overall summary of a method's actions) and programmer conventions in coding as well as method signatures to automatically mine semantically similar verbs. Their approach contained four main phases of automation as shown in Figure 3:

- Identifying descriptive leading comments.
- Extracting documented actions from leading comments.
- Extracting the main action from the method signature.
- Analyzing comment-code word pairs.

Non-descriptive leading comments are filtered out in the first phase; then, the main action words from the descriptive leading comment and the method signature are extracted for each descriptive leading comment; hence, the candidate semantically similar word pairs are formed by comment-code word pairs; finally, the comment-code pairs are analysed and ranked to automatically produce a list of semantically related word pairs (Howard et al., 2013).

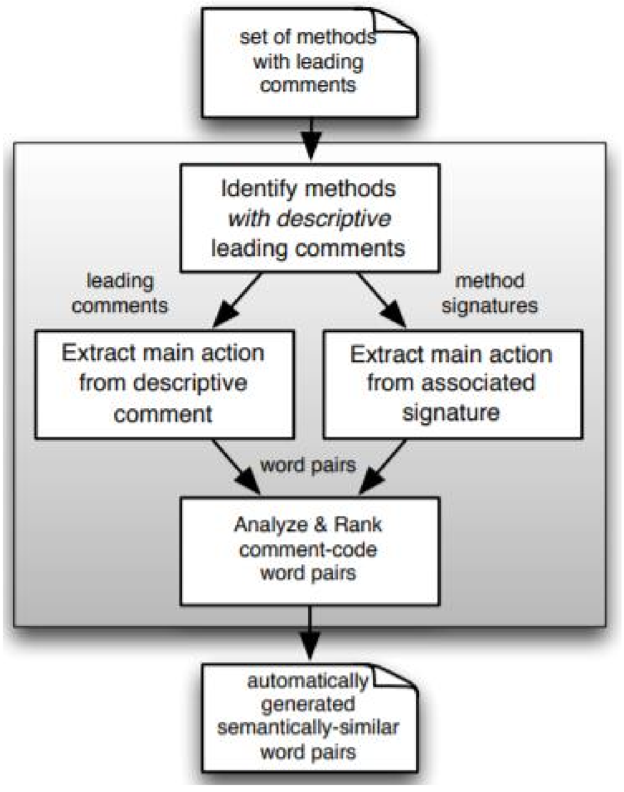

set of methods with leading comments
Identify methods *with descriptive* leading comments
leading comments
method signatures
Extract main action from descriptive comment
Extract main action from associated signature
word pairs
Analyze & Rank comment-code word pairs
automatically generated semantically-similar word pairs

Figure 3. The main process of mining word pairs (Howard et al., 2013, p. 380).

Regarding the version control data, Kim and Notkin (2006) conducted a study on matching techniques that could be used for multi-version program analysing. Their work indicated that matching program elements between two revisions was a basic building block for multi-version program analyses and other software evolution research. They concluded that every matching technique is an implementation of some kind of pseudo equivalence function, and the more heuristics that are used, the better the matching technique will perform. On the other hand, Parnin and Görg (2008) also proposed a technique for expressing changes between software versions. Instead of textual analysis of source code and line-based differencing, they worked on fully annotated bytecode to utilized type information that is harder to obtain with textual representations. They obtained which bytecode statements differed between revisions, emitted a collection of symbol pairs for each statement that had been modified, and identified which pairs were removed and inserted between revisions. The symbols have the type of information and fully qualified source code location annotated on them (Parnin & Görg, 2008).

Bug Repositories

In the form of bug reports, details about software bugs are stored in a bug tracking system (*e.g.* Bugzilla[5], Redmine[6], Jira[7]) which contains information about the creation and the resolution of bugs, feature enhancements, and other maintenance tasks (Nazar et al., 2016). Furthermore, a bug repository can be combined with the version control repository to analyse fix-inducing changes, *i.e.*, changes that lead to problems, indicated by fixes (Śliwerski, Zimmermann, & Zeller, 2005). For their analysis, Bugzilla should be mapped in a local database. Thus, they used the XML export feature of Bugzilla and then imported attachments and activities directly from the web interface of Bugzilla.

In addition to natural language text, bug reports also contained structural elements such as stack traces, source code fragments, and patches (Bettenburg, Premraj, Zimmermann, & Kim, 2008). To make full use of the valuable information of bug reports, a tool called infoZilla that could extract structural elements from bug reports was presented by Bettenburg et al. (2008). Their study has some similarities to my research, *i.e.*, separate natural language and code. However, the objectives of the two studies were different, *i.e.*, my study needs to consider both the extraction of natural language and non-natural language but they focused on only extracting the non-natural language fragments from the context in which natural language is dominant due to the descriptions of bug reports were usually regarded as natural language text (Bettenburg et al., 2008). Moreover, they mainly used heuristics, regular expressions, and island grammars (Moonen, 2001) as the filters for detecting specific elements with remarkably high accuracy. Nevertheless, they indicated the advantages of having additional information such as stack traces and source code separated from natural language, *i.e.*, it could enable researchers to access more structured data, improve machine learning training, and allow research based on specific interests.

Moreover, the life cycle data of issues that are preserved in bug tracking systems can also be regarded as one type of information source for MSR. A typical life cycle of issues is shown in Figure 4. For instance, an early study conducted by Weiss, Premraj, Zimmermann, and Zeller (2007) demonstrated an approach that utilized the issue tracking systems to predict the fixing effort, *i.e.*, the person-hours spent on fixing an issue. They used the Lucene[8] (an open-source search engine software library) framework to search for similar, earlier reports and used their average time as a prediction when given a new

[5] https://www.bugzilla.org/

[6] https://www.redmine.org/

[7] https://www.atlassian.com/software/jira

[8] https://lucene.apache.org/

issue report. Regarding when there were no similar issues, Weiss et al. (2007) also extended the nearest neighbour approach (kNN) to increase the reliability of predictions.

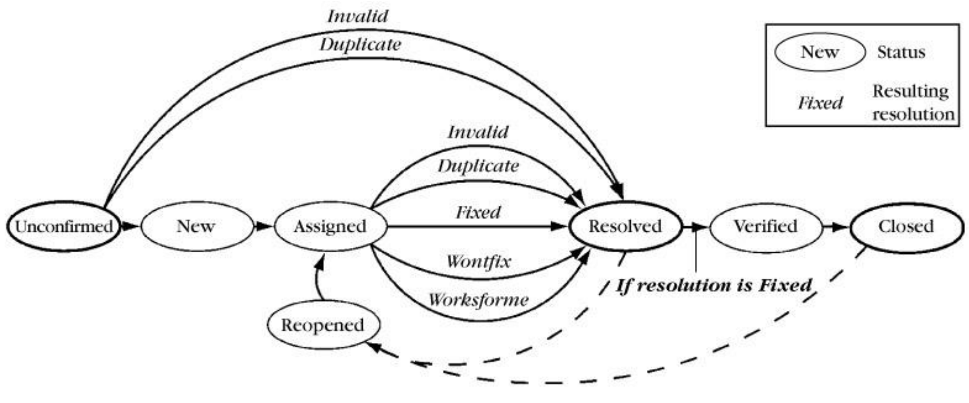


Figure 4. The life cycle of an issue report in Bugzilla (Zeller, 2009)

Mailing Lists

Mailing lists play a central role in communication during Open Source Software (OSS) development process (Ibrahim, Bettenburg, Shihab, Adams, & Hassan, 2010). Typically, mailing lists consist of a series of time-stamped email messages that include a header that contains the sender, receiver(s), and time stamp, a message body, *i.e.*, the email's text content, and a set of attachments, *i.e.*, additional documents sent with the email (Nazar et al., 2016).

Rigby and Hassan (2007) indicated that the unstructured nature of mailing lists increased the difficulty of the information extraction, therefore, they proposed an approach that used a psychometrically-based word count tool to understand Apache httpd[9] (an open-source HTTP server) developer mailing lists. They aimed at discovering the personality and attitude of developers. Similarly, Júnior, Mendonca, Farias, and Henrique (2010) presented a psychometrically-based neuro-linguistic mining tool to analyse the mailing lists of the Apache project. They intended to address the distribution of representational systems among the Apache project developers and explore what is the preferred representational system (PRS) of the Apache project top committers. Likewise, Ibrahim et al. (2010) proposed a composite model that explored the contribution behaviour for the top ten developers in three OSS mailing lists (Apache, PostgreSQL[10] , and Python[11]). To improve the overall performance, they combined two data mining approaches, *i.e.*, using a Naive Bayesian classifier to deal with the body and subject of a thread and a Decision Tree classifier to explain the contribution behaviour of a developer. Hence, they found the most important contributing factors are the message content, developer contribution activity in the last month, and the length of the thread.

However, in recent years, there has been a shift in the communication habits of the OSS projects toward increased usage of issue repositories and other social platforms (Guzzi,

[9] https://httpd.apache.org/

[10] https://www.postgresql.org/

[11] https://www.python.org/

Bacchelli, Lanza, Pinzger, & Van Deursen, 2013). Guzzi et al. (2013) analysed the email threads from the development mailing list of an OSS project, Lucene. Their findings revealed that email threads covered a wide range of topics, with implementation details only appearing in a small percentage of them, those code artefacts were discussed in topics other than implementation, and those project developers did not make up the majority of the participants.

Online Discussions

As another form of APC, online discussions (also known as developer forums) are online forums where engineers, developers, beginners, and the general public can post questions and exchange comments about the same specialized area (Nazar et al., 2016). For instance, Twitter[12], Stack Overflow[13], and Slack[14] are popular for online discussions.

Regarding Twitter, Tian, Achananuparp, Lubis, Lo, and Lim (2012) examined what the software engineering community microblogged on Twitter and classified these microblogs into ten categories as shown in Table 2. They also analysed the types of tweets that drew the most attention and spread the most via retweeting. Thus, they found that job openings, news, Q&A, and new tools and code were the most popular categories of tweets, while events and commercials were the most commonly propagated or shared tweet categories.

Table 2. The categories of tweets (Tian et al., 2012, p. 248).

	Category	**Description**
1.	Commercials	Promotions about a particular company or a commercial product
2.	News	Objective reports about a particular topic
3.	Tools & Code	Sharing of open source tools or code for general use
4.	Q&A	Questions either posed directly or asked via question-answering sites, *e.g.*, StackOverflow.com
5.	Events	Information about various gatherings or activities, *e.g.*, conferences
6.	Personal	Personal messages, *e.g.*, ramblings about daily activities, conversations with friends, etc.
7.	Opinions	Opinions (*e.g.*, likes, dislikes, etc.) about a particular topic
8.	Tips	Advice about a particular topic
9.	Jobs	Job openings
10.	Misc.	Miscellaneous microblogs not belonging to one of the above categories or not related to software engineering. These also include microblogs whose contents are unclear.

[12] https://twitter.com/

[13] https://stackoverflow.com/

[14] https://slack.com/

According to Chowdhury and Hindle (2015), StackOverflow (SO) is an example of an asynchronous long-lived discussion medium, in which questions and responses are posted at various times and the conversation is archived. Wang, Malik, and Godfrey (2015) presented an approach for recommending SO posts that were likely to concern API design issues by combining social network analysis and topic mining. Similarly, Bangash et al. (2019) conducted a study on SO machine learning related posts to discover what machine learning topics were discussed on SO. Moreover, Rahman, Rigby, Palani, and Nguyen (2019) evaluated three different corpus cleaning approaches, *i.e.*, standard NLP, title only, and software task extraction for SO to generate a parallel English-Code corpus from Android related posts. Their results proved the potential of SO as a bilingual machine translation corpus.

Slack is another popular chat platform that hosted many active channels focused on software development technologies, *e.g.*, Python, Rails[15], React[16] (Chatterjee, Damevski, Pollock, Augustine, & Kraft, 2019; Chatterjee, Damevski, Kraft, & Pollock, 2020). Chatterjee et al. (2019) conducted an exploratory study to investigate the potential of developer Q&A chats in Slack as a mining resource for software maintenance and evolution tools. They examined that Q&A chats in Slack provided the same information as can be found in Q&A posts on SO. Furthermore, Chatterjee et al. (2020) presented an approach for automatic data collection, pre-processing, disentanglement and storage of Slack developer chats. The dataset derived from their study could be utilized to identify trending topics of discussion in a software development community and understand common challenges and misconceptions among developers.

Logs

Vaarandi (2003) presented a clustering algorithm for mining patterns from log files. In that study, an experimental clustering tool called SLCT[17] (Simple Logfile Clustering Tool) was implemented. Accordingly, Nagappan and Vouk (2010) presented an approach that was a modification of the SLCT algorithm for log file abstraction to cope with the inherent variability of log files. Furthermore, to build a link between the log messages in the raw log file and the log statements in the source code, Schipper, Aniche, and van Deursen (2019) proposed an approach to implement and evaluate log parsing.

On the other hand, Bajracharya and Lopes (2009) proposed a topic modelling analysis for revealing a usage log of a commercial code search engine. They used Latent Dirichlet Allocation (LDA) as the topic modelling technique and identified the common ways in which users expressed their queries. Similarly, Baysal, Holmes, and Godfrey (2012) demonstrated how usage data from web server logs can be combined with development information to provide insight into user dynamic behaviour and adoption patterns in a variety of deployment environments. Likewise, Fu et al. (2013) proposed an approach for contextual analysis of system logs for understanding the behaviors of a software system.

[15] https://rubyonrails.org/

[16] https://reactjs.org/

[17] https://ristov.github.io/slct/

2.2 Natural language processing

Natural language processing (NLP) is an interdisciplinary research area aimed at getting computers to perform useful tasks involving human language, such as allowing human-machine communication, enhancing human-human communication, or simply processing text or speech for useful purposes (Jurafsky & Martin, 2009). NLP research began in the 1950s, the initial interest was focused on machine translation (Jones, 1994). Since then, diverse research tasks have been studied in the NLP field, such as information retrieval, text summarization, question answering, information extraction, topic modelling, and opinion mining (Cambria & White, 2014). The task that is related to this study - text categorization (classification) is to assign natural language texts to one or more predefined categories based on their content (Dumais, Platt, Heckerman, & Sahami, 1998).

The traditional NLP process may be divided into several steps, reflecting the theoretical linguistic differences between syntax, semantics, and pragmatics (Indurkhya & Damerau, 2010). However, with the development of machine learning techniques, the increasing dominance of statistical approaches in NLP has been acknowledged (Indurkhya & Damerau, 2010). Hence, in this paper, the focus is automatic text categorization with machine learning techniques. Moreover, instead of processing the original text document, text representation and feature selection are used when applying machine learning methods to text categorization for the reasons of both efficiency and efficacy (Dumais et al., 1998).

2.2.1 Text Representations

The "bag of words" was coined by Harris (1954) for describing language in terms of a distributional structure. Naturally, the bag-of-word model does not preserve the order of words. Therefore, based on the assumption that the order of words would not matter, Lang (1995) used the bag-of-words model for text representation to build a Netnews filter system. Likewise, Joachims (1997) provided a more detailed description of bag-of-words representation for text classification tasks, *i.e.*, each word is a feature and the number of times the word occurs in the document is its value as shown in Figure 5. However, the assumption of bag-of-words was challenged by Scott and Matwin (1998, 1999). They tried to represent texts by adding syntactic and semantic relationships between words, although the results did not show significant performance improvements. Nevertheless, their work supported the rising consensus that more sophisticated NLP techniques were needed before better text representations emerged.

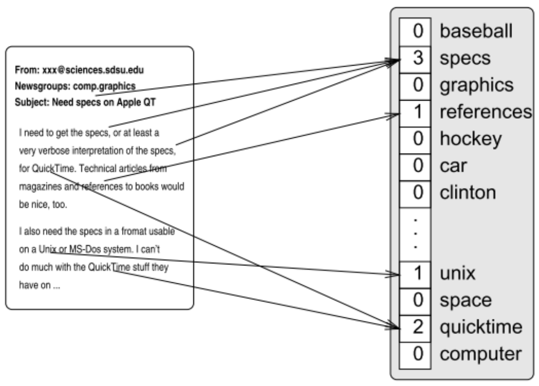

From: xxx@sciences.sdsu.edu
Newsgroups: comp.graphics
Subject: Need specs on Apple QT
I need to get the specs, or at least a very verbose interpretation of the specs, for QuickTime. Technical articles from magazines and references to books would be nice, too.
I also need the specs in a fromat usable on a Unix or MS-Dos system. I can't do much with the QuickTime stuff they have on ...
0 baseball
3 specs
0 graphics
1 references
0 hockey
0 car
0 clinton
1 unix
0 space
2 quicktime
0 computer

Figure 5. Bag-of-words representation in an attribute value style (Joachims, 1997).

On the other hand, the n-gram models were used to address the problem of predicting a word from previous words in a sample of text (Brown, DeSouza, Mercer, Della Pietra, & Lai, 1992). Meanwhile, Cavnar and Trenkle (1994) proposed an N-gram-based approach for text categorization. They used n-grams frequency profiles to represent text categories. Cavnar and Trenkle (1994) provided an example of n-grams for the word "TEXT":

- bi-grams: _T, TE, EX, XT, T_
- tri-grams: _TE, TEX, EXT, XT_, T_ _
- quad-grams: _TEX, TEXT, EXT_, XT_ _, T_ _ _

a string was sliced into a set of overlapping n-grams, and they also appended blanks (underscore character "_") to the beginning and end of the string to help with matching beginning-of-word and end-of-word situations. The original paper (Mäntylä et al., 2018) was also inspired by the character tri-grams approach for identifying languages which was presented by T. Jauhiainen, Linden, and H. Jauhiainen (2017).

Moreover, Bengio, Ducharme, Vincent, and Janvin (2003) proposed a neural network language model to learn jointly the word vector representation and a statistical language model that significantly improved on the n-grams model. Besides, Mikolov, Chen, Corrado, and Dean (2013) introduced word2vec[18] that consisted of two methods for learning representations of words, *i.e.*, continuous Bag-of-Words model and continuous Skip-gram model. They noticed the potential of using simple model architectures to train high-quality word vectors compared to the neural network models.

Similarly, source code could be represented as the language model mentioned above. Collard, Decker, and Maletic (2013) presented srcML[19] which is an XML representation for C/C++/Java source code. They used a lightweight format that allows round-trip transformation with no loss of information or formatting. Apart from this, Alon, Zilberstein, Levy, and Yahav (2019) proposed code2vec[20] which is a neural model for representing snippets of code as distributed vectors (see Figure 6).

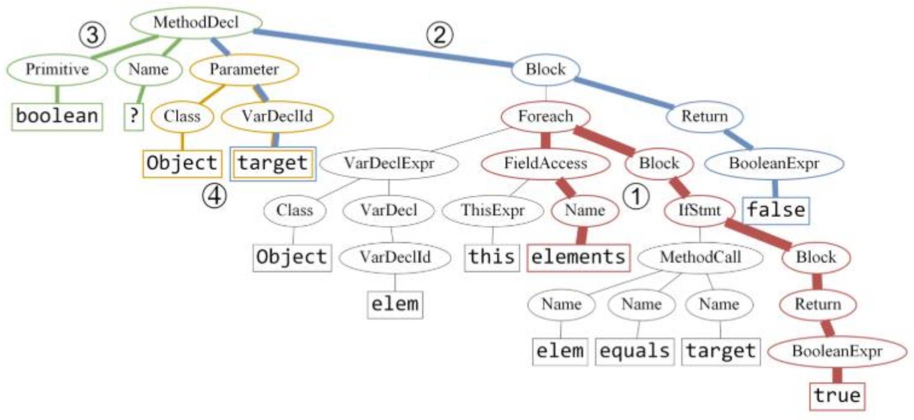


Figure 6. Source code is decomposed into a collection of paths (Alon et al., 2019).

[18] https://code.google.com/archive/p/word2vec/

[19] https://www.srcml.org/

[20] https://code2vec.org/

2.2.2 Classification Algorithms

Kotsiantis (2007) presented a prominent review of supervised machine learning and classification algorithms. Thereby, I will base on that work and refer to other recent studies as well. In general, machine learning algorithms can be divided into supervised learning and unsupervised learning. As shown in Table 3, the supervised learning needs to give the instance in dataset known labels, *i.e.*, the corresponding correct outputs (Kotsiantis, 2007). Therefore, in this paper, the focus is on the supervised learning algorithms due to the study also need to label the dataset as natural language or not for training and testing.

Table 3. Instances with known labels (Kotsiantis, 2007).

	Feature 1	**Feature 2**	**…**	**Feature n**	**Class**
1	xxx	x		xx	Good
2	xxx	x		xx	Good
3	xxx	x		xx	Bad
…					...

The supervised learning techniques for text classification were extensively studied, including Bayesian classifier, Decision Tree, K-nearest neighbour (KNN), Support Vector Machines (SVMs), and Neural Networks (A. Khan, Baharudin, Lee, & K. Khan, 2010). According to Kotsiantis (2007), a general supervised learning process includes several steps:

1. Collecting the dataset.
2. Data preparation and data pre-processing.
3. Algorithm selection (training and testing).
4. Classifier for routine use.

The core step is to choose the appropriate learning algorithm and the process can be repeated multiple times until the prediction accuracy, *i.e.*, the percentage of correct prediction divided by the total number of predictions is satisfactory (Kotsiantis, 2007).

Naive Bayes

Based on the strong independence assumptions of Bayes' Theorem (Good, 1950), the Naive Bayes classifier is a simple probabilistic classifier that can be trained very efficiently by requiring a relatively small amount of training data to estimate the parameters necessary for classification (Khan et al., 2010). McCallum and Nigam (1998) indicated that although the naive assumption was false in most real-world tasks, the Naive Bayes often performs very well in the text classification domain. Hence, the major advantage of the Naive Bayes classifier is its short computational time for training (Kotsiantis, 2007). However, Khan et al. (2010) argued that Naive Bayes performs very poorly when features are highly correlated and does not consider the frequency of word occurrences.

Support Vector Machine (SVM)

SVM classification is based on the Structural Risk Minimization principle proposed by Vapnik (1995) from computational learning theory. The idea of this principle is to find a hypothesis to guarantee the lowest true error (Joachims, 1998). Once the optimal separating hyperplane is found for linearly separable data (see Figure 7), data points on its margin are known as support vector points, and the solution is defined as a linear combination of only these points (Kotsiantis, 2007).

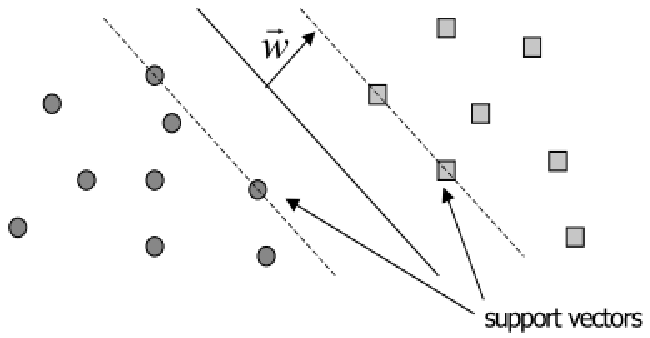


Figure 7. Linear Support Vector Machine (Dumais et al., 1998).

LIBSVM[21] is one of the most widely used SVM software which had been developed since the year 2000 (Chang & Lin, 2011). Hence, Karatzoglou, Meyer, and Hornik (2006) introduced an implementation of SVM in the R package E1071[22] based on LIBSVM. They provided a rigid interface to LIBSVM along with visualization and parameter tuning methods. The Python implementation of LIBSVM is used in this study.

Regarding the multi-class classification, SVM implemented the "one-against-one" approach, *i.e.*, if k is the number of classes, then to construct $k(k-1)/2$ classifiers and each one trains data from two classes (Chang & Lin, 2011).

$$\begin{aligned} \underset{\omega^{ij}, b^{ij}, \xi^{ij}}{min} \quad & \frac{1}{2}\left(\omega^{ij}\right)^T \omega^{ij} + C\sum_t \left(\xi^{ij}\right)_t \\ \text{subject to} \quad & \left(\omega^{ij}\right)^T \phi(x_t) + b^{ij} \geq 1 - \xi_t^{ij}, \text{ if } x_t \text{ in the } i\text{th class}, \\ & \left(\omega^{ij}\right)^T \phi(x_t) + b^{ij} \leq -1 + \xi_t^{ij}, \text{ if } x_t \text{ in the } j\text{th class}, \\ & \xi_t^{ij} \geq 0. \end{aligned} \tag{1}$$

Formula (1) shows how to solve a two-class classification problem for training data from the *i*th and the *j*th classes, Chang and Lin (2011) used a voting strategy, *i.e.*, each binary classification was considered to be voting and a point was selected to be in a class with the maximum number of votes.

[21] https://www.csie.ntu.edu.tw/~cjlin/libsvm/

[22] https://cran.r-project.org/package=e1071

k-Nearest Neighbours (k-NN)

The nearest neighbour algorithm is one of the most straightforward instance-based learning algorithms which is also lazy-learning, *i.e.*, the induction or generalization process is delayed until classification is performed (Kotsiantis, 2007). Cover and Hart (1967) proposed the basic principle for k-Nearest Neighbours (k-NN), *i.e.*, the instances within a dataset will generally exist close to other instances that have similar properties. Additionally, Han, Karypis, and Kumar (1999) presented a k-NN scheme for text categorization in which the importance of discriminating words was learned using mutual information and weight adjustment techniques. The training sets are mapped into a multi-dimensional feature space, and a point in the feature space is assigned to a specific category when it is the most frequent among the *k* closest training data (Khan et al., 2010). Figure 8 illustrated the k-NN algorithm and indicated that the choice of *k* value would affect the result (Zhu, Spachos, Pensini, & Plataniotis, 2021).

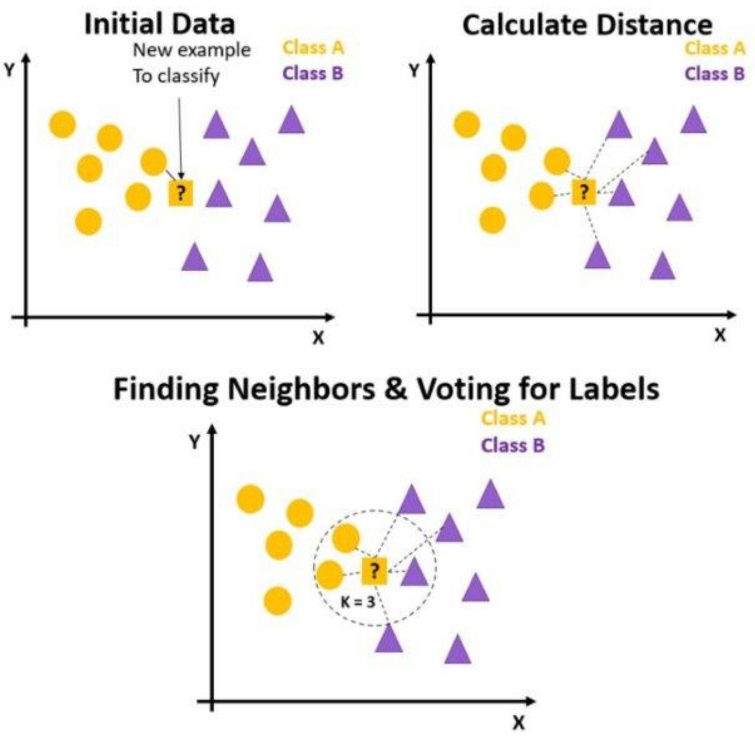


Figure 8. An example of the k-NN algorithm (Zhu et al., 2021).

Decision Trees

Decision trees are hierarchical, sequential structures that recursively partition data to represent rules underlying data (Murthy, 1998). In a decision tree (see Figure 9), each node represents a feature in a classification case, and each branch represents a value that the node can assume (Kotsiantis, 2007). Hence, the key benefit of a decision tree is its ease of comprehension and interpretation; the description of a given result can be easily replicated using simple mathematics algorithms (Khan et al., 2010). Besides, Ho (1995) proposed Random Forest that is a method to construct tree-based classifiers by building multiple trees in randomly selected subspaces of the feature space. Another advanced approach called Gradient Boosting was proposed to produce competitive, highly robust, interpretable procedures for both regression and classification by Friedman (2001).

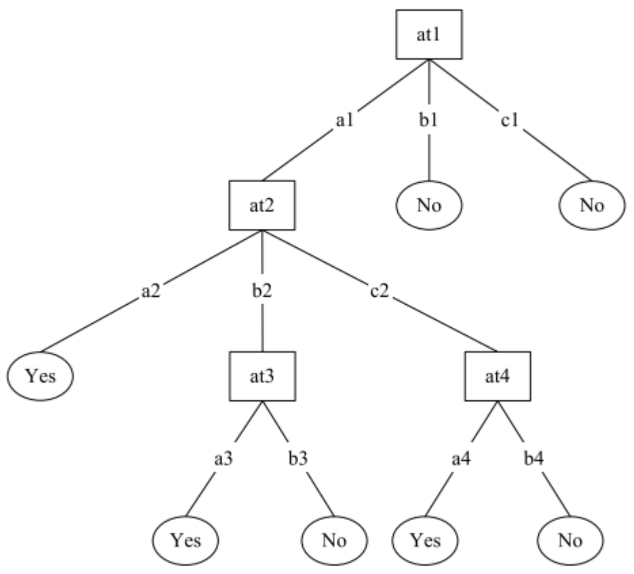


Figure 9. A decision tree (Kotsiantis, 2007).

Artificial Neural network (ANN)

Rumelhart, Hinton, and Williams (1985) proposed a machine learning procedure with a multi-layered perceptron (Artificial Neural Networks). As shown in Figure 10, a multi-layer neural network uses a pattern of connections to join a large number of units, including input units for receiving information, output units for the results, and hidden units in between (Kotsiantis, 2007). ANN was suggested to be an important tool in automatic text categorization (Ruiz & Srinivasan, 1997). However, the training cost of ANN needs to be considered due to training ANN is generally much more time consuming than the other classifiers (Yang & Liu, 1999). Hochreiter and Schmidhuber (1997) presented long short-term memory (LSTM) which is an architecture of recurrent neural network (RNN) to solve complex, artificial long-time-lag tasks. Moreover, regarding another form of ANN, the convolutional neural network (CNN) was used for handwritten digit recognition (LeCun et al., 1990).

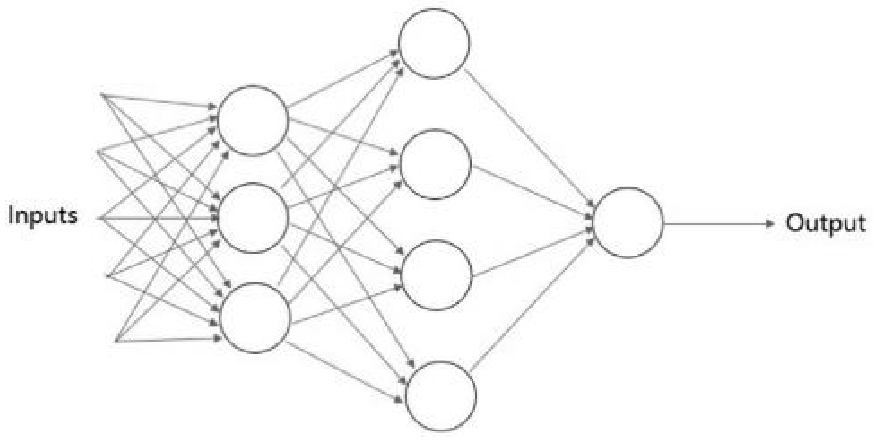


Figure 10. An example structure of ANN (Zhu et al., 2021).

2.3 Natural Language or Not (NLoN)

Mäntylä et al. (2018) presented the NLoN (a lightweight classifier tool that would take a line of text as input and predict whether it is **N**atural **L**anguage **o**r **N**ot) for text classification of MSR information sources. Bacchelli, Dal Sasso, D'Ambros, and Lanza (2012) presented a similar study for classifying content from software development emails, however, the complexity of design is too excessive compared to NLoN.

Mäntylä et al. (2018) used three different types of MSR information sources, including:

- Issue comments of Mozilla Firefox project.
- Archived chats of Slack channel for Kubernetes.
- Mailing lists of Apache Lucene project.

After the mining of those information sources, they manually and independently labelled the data as the ground truth for training and testing. Regarding the text representation, Mäntylä et al. (2018) utilised two approaches, *i.e.*, feature engineering and character tri-grams. They created eleven feature engineering predictors of language such as the ratio of non-letter characters and the ratio of capital letters (see Table 4).

Table 4. Feature engineering predictors (Mäntylä et al., 2018).

	Abbreviation	**Explanation**
1	r.caps	Ratio of capital letters
2	r.specials	Ratio of chars not alphanumeric or whitespace
3	r.numbers	Ratio of numbers
4	l.words	Length of words
5	n.sw	Number of stop-words split with white space
6	n.sw2	Number of stop-words split with tokenize words
7	last.c.code	Is last character typical in code, *e.g.*, { or ;
8	n.c1-3.letters	Number of letters in first three characters
9	last.c.nl	Is last character typical in NL, *e.g.*, ? or .
10	n.emoticons	Number of emoticons
11	first.c.at	Is first character of line @-sign

Additionally, they also used character tri-grams (one variant of the n-grams) and applied some adaptations such as change all numbers to zeros and convert all characters to lower case. Regarding the learning model, they selected Glmnet for performing Lasso regression (Tibshirani, 1996) and optimized its 10-fold cross-validation performance concerning the area under the ROC curve (AUC). As for the result, Mäntylä et al. (2018) achieved comparable performance with an area under the ROC curve performances from 0.976 to 0.987 and cross-source prediction performance from 0.913 to 0.980. Nevertheless, it was shown that different machine learning algorithms could be applied in the MSR field (Pascarella, 2018; Fucci, Mollaalizadehbahnemiri, & Maalej, 2019; Zhai et al., 2020). Pascarella (2018) used Naive Bayes Multinominal and Random Forest to classify code comments in Java mobile applications. Fucci et al. (2019) compared k-NN and SVM with RNN trained on manually annotated Java and .NET API documentation. Zhai et al. (2020) leveraged decision trees, random forests, and CNN to automatically categorize comments.

3. Research methods

Before introducing the methodology, it is necessary to clarify the research object of this study. NLoN is a text categorization tool that can be regarded as an artificial creation. According to Simon (1996), the definition of “artificial” is “produced by art rather than by nature; not genuine or natural; affected; not pertaining to the essence of the matter” (p. 4). Therefore, this study aims at improving the performance of an artefact (NLoN). Likewise, the objective of design science research (DSR) is to develop technology-based solutions to important and relevant business problems (Hevner, March, Park, & Ram, 2004). Moreover, Hevner and Chatterjee (2010) defined DSR as follows:

> Design science research is a research paradigm in which a designer answers questions relevant to human problems via the creation of innovative artifacts, thereby contributing new knowledge to the body of scientific evidence. The designed artifacts are both useful and fundamental in understanding that problem. (p. 5)

The definition of DSR fits our research entirely due to the artefact of this study was created to solve the actual problem that would be met in the research process of mining software repositories, *i.e.*, the natural language and source code need to be separated for further analysis. Hence, DSR is selected as the methodology of this research.

3.1 Design science research methodology (DSRM)

Hevner et al. (2004) introduced the conceptual framework and practice rules of DSR which provided an understanding of how to conduct, evaluate, and present DSR. However, Peffers et al. (2007) indicated that principles and practice rules were only part characteristics of a methodology, a generally accepted process was missing. Therefore, Peffers et al. (2007) presented the design science research methodology (DSRM) for conducting DSR in information systems, they provided a commonly accepted framework for carrying out DSR and a mental model for its presentation. Besides, DSRM may also help with the recognition and legitimization of DSR and its objectives, processes, and outputs (Hevner & Chatterjee, 2010). Hence, a nominal process for the conduct of DSR was suggested by Peffers et al. (2007) as a roadmap for researchers who want to use design as a research mechanism for IS research (see Figure 11).

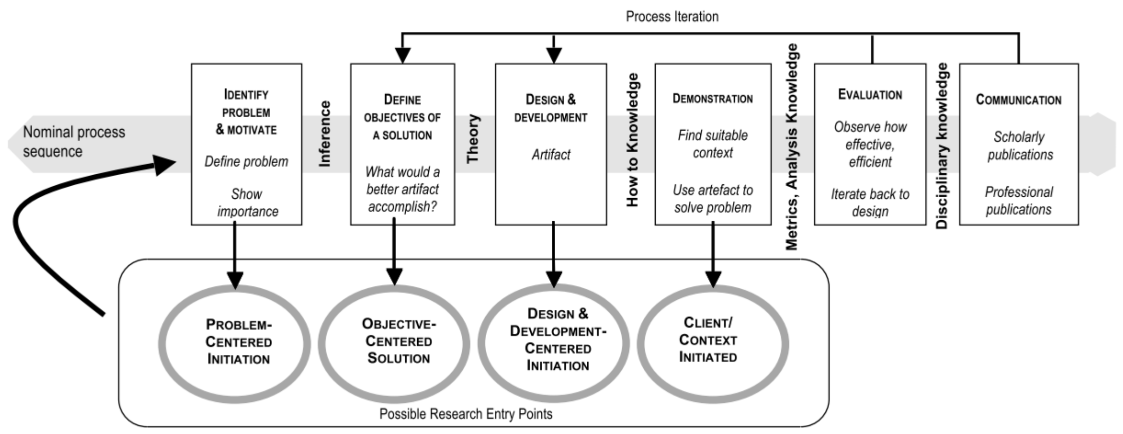

Process Iteration
Nominal process sequence
Identify Problem & Motivate
Define problem
Show importance
Inference
Define Objectives of a Solution
What would a better artifact accomplish?
Theory
Design & Development
Artifact
How to Knowledge
Demonstration
Find suitable context
Use artefact to solve problem
Metrics, Analysis Knowledge
Evaluation
Observe how effective, efficient
Iterate back to design
Disciplinary knowledge
Communication
Scholarly publications
Professional publications
Problem-Centered Initiation
Objective-Centered Solution
Design & Development-Centered Initiation
Client/Context Initiated
Possible Research Entry Points

Figure 11. DSRM Process Model (Peffers et al., 2007).

As shown in Figure 11, the DSRM process model consists of six activities (*e.g.* problem identification and motivation; definition of the objectives; design and development; demonstration; evaluation; and communication) and four entry points (*i.e.* problem-centred initiation; objective-centred solution; design & development-centred initiation; and client/context initiated). I will explain my understanding of this DSRM process model in the following subsections.

3.1.1 Problem Identification and Motivation

According to Peffers et al. (2007), the first activity is to define the specific research problem and justify the value of a solution. In my opinion, defining a problem is a process of refinement and decomposition in where the context of the problem should be constantly clarified, *i.e.*, under what specific conditions the problem is valid. Moreover, based on a clear scope and context, the complexity of the problem could have a measurable premise. Peffers et al. (2007) summarized it as atomize the problem and capture its complexity.

Regarding motivation, the general assumption for why researchers conduct a study is that there is no solution to the defined problem or the existing solution has some deficiencies. At first, my recognition of motivation was why I wanted to study the problem, but after two years' academic training, the subject of motivation no longer included only myself, *i.e.*, the readers of the paper or other researchers can be the subject of motivation as well. According to Peffers et al. (2007), it motivates both the researcher and the audience.

Besides, Peffers et al. (2007) indicated that this activity required the knowledge of the state of the problem and the importance of its solution. From my understanding, the first thing is to search for the relevant literature to avoid reinventing the wheel. However, the "relevant" is not limited to one specific discipline due to the need to know if there is any crossover research in other disciplines, *e.g.*, this study involves several interdisciplinary fields of study such as mining software repositories, natural language processing, machine learning, and linguistics.

3.1.2 Objectives of the Solution

After the problem is identified, the next activity is to infer rationally the objectives (quantitative or qualitative) of a solution from the problem definition (Peffers et al., 2007). The rationality of objectives refers to the comparison of existing solutions, *i.e.*, a researcher should not imagine the performance, effect and capability of the solutions without any basis. For this study, the main objective is to improve the performance of the NLoN tool, so I needed to know how the original study evaluated the performance and what the performance had been achieved. Then based on the existing knowledge, I was able to perceive the practical meaning behind the phrase "improve performance". The objective of RQ1 could be qualitative because it was a problem that the current solution did not address, *i.e.*, the classification of mixed statements. In contrast, the objective of RQ2 could be quantitative due to it challenged whether the existing performance could be improved.

3.1.3 Design and Development

This activity included determining the artefact's desired functionality and its architecture and then creating the actual artefact (Peffers et al., 2007). The artefact could be in the form of a construct, a model, a method, or an instantiation (Hevner et al., 2004). In my opinion, the problem is the current place, the objective is the destination, and the implementation (design or development) is the way to connect them. The implementation of this study will be described in detail in Chapter 4.

3.1.4 Demonstration

After the implementation, the use of the artefact should be demonstrated to solve one or more instances of the problem (Peffers et al., 2007). In this paper, the initial idea of demonstration would be experimentation that used data from diverse information sources (*e.g.* mail lists, issue comments, and other online discussions) to train and test the classification tool.

3.1.5 Evaluation

Since objectives were defined in activity 2 and results were generated in activity 4, they could be compared by observing and measuring how well the artefact supports a solution to the problem (Peffers et al., 2007). I think this activity not only allows the author to evaluate the results but also allows audiences to evaluate whether this research is valid or not.

3.1.6 Communication

According to Peffers et al. (2007), communication was to inform researchers and other related audiences about the issue and its significance, the artefact, its usefulness and novelty, the rigour of its design, and its effectiveness. The original research was published as well as making the code open on GitHub and the tool was released as an R package. This research also planned to publish the paper and open the source code. As to whether it will still be developed in the form of R language or Python language, it will be discussed in the following chapters.

3.2 The adoption of DSRM in this study

Although Peffers et al. (2007) mentioned that the DSRM process was structured in nominally sequential order, *i.e.*, researchers do not have to follow this order exactly. This study is a design and development-centred approach that followed the basic process order except for starting with activity three due to the selected entry point (see Figure 12). The entry point of this research is the enhancement of design and development on an existing artefact (NLoN tool). The problem of separating natural language was identified as two research questions, *i.e.*,

- RQ1: How to design and implement a multi-class classification approach based on previous work?
- RQ2: Whether other algorithms can get better performance than Glmnet without breaking the original principle of lightweight design?

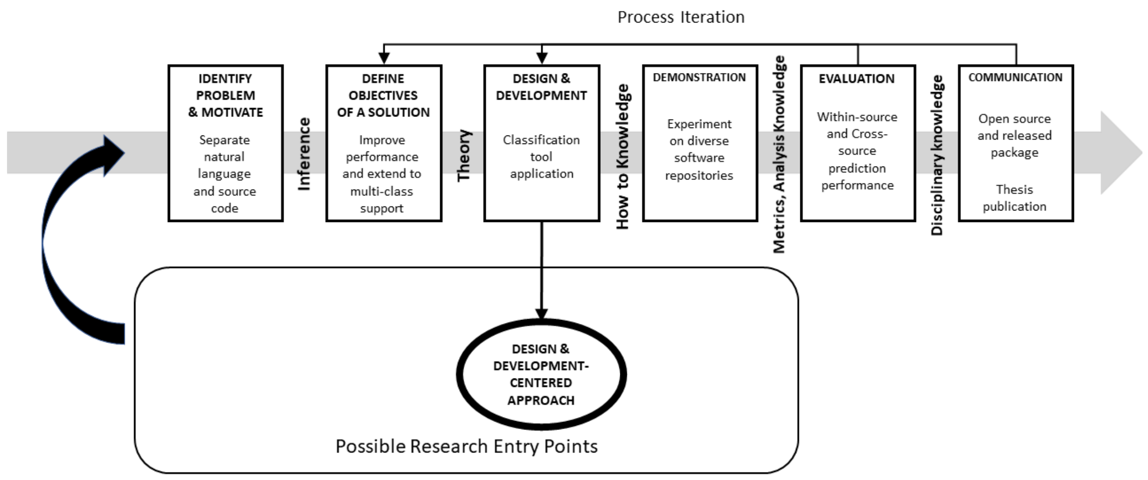

Process Iteration
IDENTIFY PROBLEM & MOTIVATE
Separate natural language and source code
Inference
DEFINE OBJECTIVES OF A SOLUTION
Improve performance and extend to multi-class support
Theory
DESIGN & DEVELOPMENT
Classification tool application
How to Knowledge
DEMONSTRATION
Experiment on diverse software repositories
Metrics, Analysis Knowledge
EVALUATION
Within-source and Cross-source prediction performance
Disciplinary knowledge
COMMUNICATION
Open source and released package
Thesis publication
DESIGN & DEVELOPMENT-CENTERED APPROACH
Possible Research Entry Points

Figure 12. The adoption of the DSRM Process Model.

Regarding the preprocessing of data, the label of the corpus is manually labelled by the author alone. Although the original study has achieved success on the objective of two-class classification, a multi-class classification approach was expected to extend the "Not" class, *i.e.*, the "Not" class could be refining to code, stack trace, error/log messages, and the mixed class which consist of natural language and some "Not" class (Figure 13).

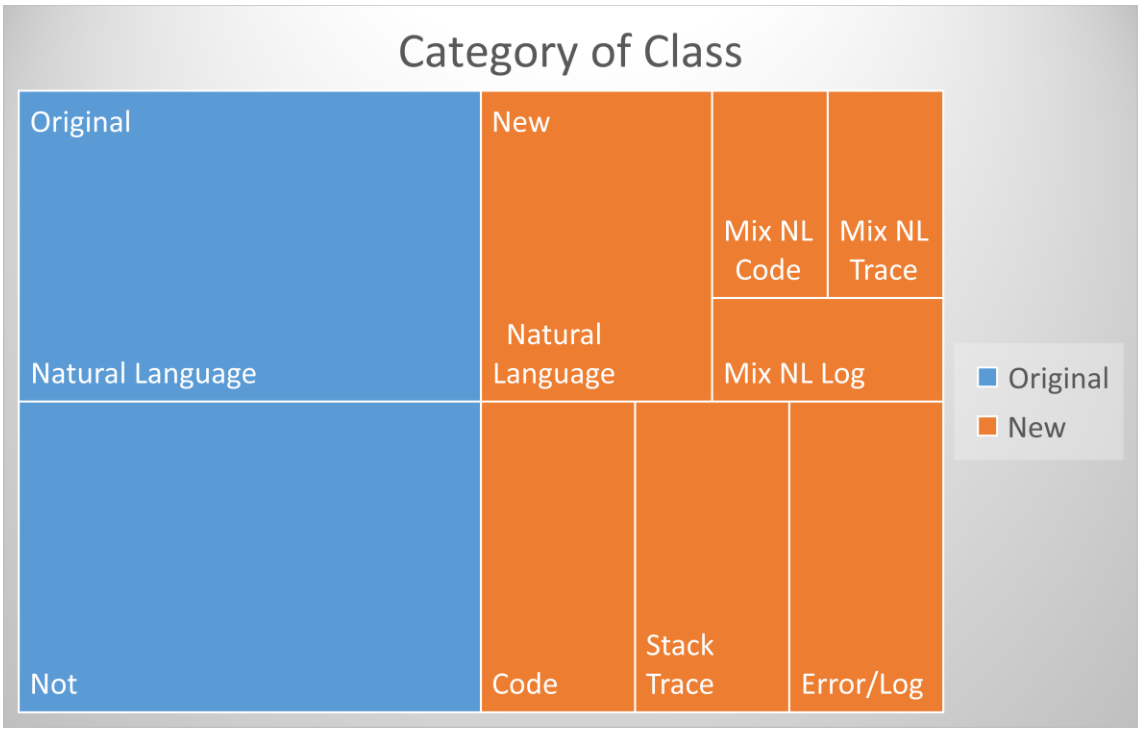


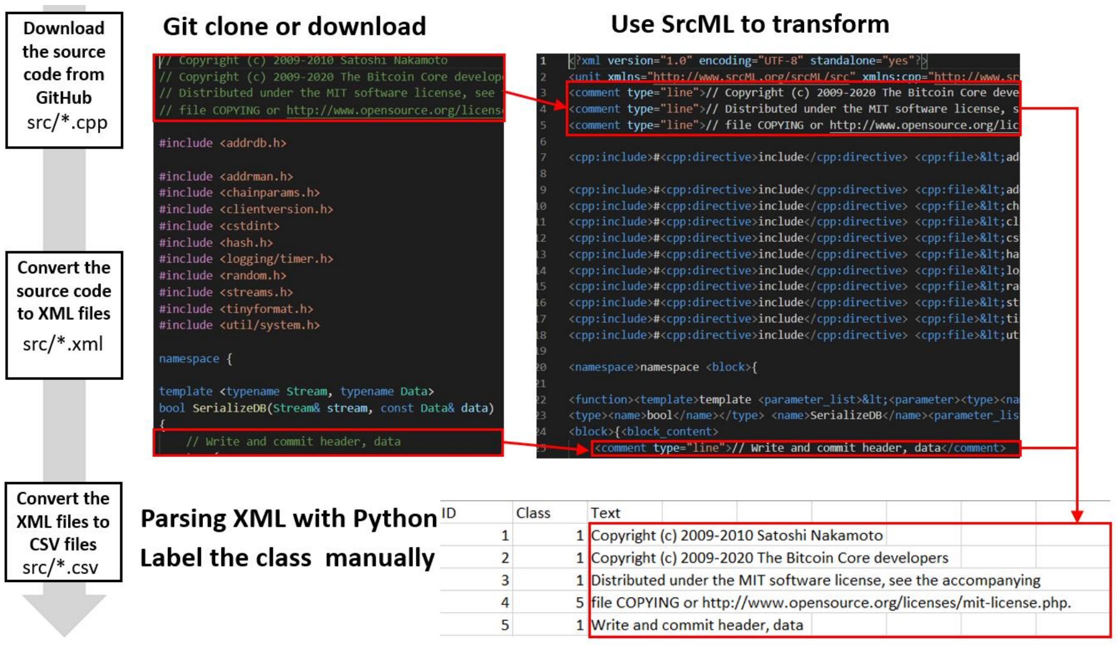
Figure 13. The category of class compared with the original study.

4. Implementation

Although the original implementation is based on the R language, combining the popular machine learning technology and my development ability, I finally chose Python as the development language for this project. Therefore, this project will be an open-source project NLoN-PY[23] hosted on GitHub, and the final artefact will be also released to the PyPI[24] (Python Package Index) repository in the form of a Python package.

4.1 Data Collection and Labelling

Since the original paper provided three different sources of information, *i.e.*, issues tracker comments (Firefox), chat messages (Kubernetes), and mailing list (Lucene), this study will add source code comments as an extra information source. The Github repository of Bitcoin[25] was selected because its code is open-source and consist of enough lines of code comments. As shown in Figure 14, the first step of data collection is to fetch source code from the GitHub repository. The text of the source code file contains both code block and comment block, as introduced in the previous chapter, srcML is used to transform the source code files into XML format. Thereby, all the comment blocks are tagged with <comment> tag in the XML files, hence using the built-in function of Python could parse the comment elements to text corpus.


Download the source code from GitHub src/*.cpp
Convert the source code to XML files src/*.xml
Convert the XML files to CSV files src/*.csv
Git clone or download
Use SrcML to transform
Parsing XML with Python
Label the class manually

```
// Copyright (c) 2009-2010 Satoshi Nakamoto
// Copyright (c) 2009-2020 The Bitcoin Core develop
// Distributed under the MIT software license, see
// file COPYING or http://www.opensource.org/licens

#include <addrdb.h>

#include <addrman.h>
#include <chainparams.h>
#include <clientversion.h>
#include <cstdint>
#include <hash.h>
#include <logging/timer.h>
#include <random.h>
#include <streams.h>
#include <tinyformat.h>
#include <util/system.h>

namespace {

template <typename Stream, typename Data>
bool SerializeDB(Stream& stream, const Data& data)
{
    // Write and commit header, data
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<unit xmlns="http://www.srcML.org/srcML/src" xmlns:cpp="http://www.sr
<comment type="line">// Copyright (c) 2009-2020 The Bitcoin Core deve
<comment type="line">// Distributed under the MIT software license, s
<comment type="line">// file COPYING or http://www.opensource.org/lic

<cpp:include>#<cpp:directive>include</cpp:directive> <cpp:file>&lt;ad

<cpp:include>#<cpp:directive>include</cpp:directive> <cpp:file>&lt;ad
<cpp:include>#<cpp:directive>include</cpp:directive> <cpp:file>&lt;ch
<cpp:include>#<cpp:directive>include</cpp:directive> <cpp:file>&lt;cl
<cpp:include>#<cpp:directive>include</cpp:directive> <cpp:file>&lt;cs
<cpp:include>#<cpp:directive>include</cpp:directive> <cpp:file>&lt;ha
<cpp:include>#<cpp:directive>include</cpp:directive> <cpp:file>&lt;lo
<cpp:include>#<cpp:directive>include</cpp:directive> <cpp:file>&lt;ra
<cpp:include>#<cpp:directive>include</cpp:directive> <cpp:file>&lt;st
<cpp:include>#<cpp:directive>include</cpp:directive> <cpp:file>&lt;ti
<cpp:include>#<cpp:directive>include</cpp:directive> <cpp:file>&lt;ut

<namespace>namespace <block>{

<function><template>template <parameter_list>&lt;<parameter><type><na
<type><name>bool</name></type> <name>SerializeDB</name><parameter_lis
<block>{<block_content>
    <comment type="line">// Write and commit header, data</comment>
```

ID	Class	Text
1	1	Copyright (c) 2009-2010 Satoshi Nakamoto
2	1	Copyright (c) 2009-2020 The Bitcoin Core developers
3	1	Distributed under the MIT software license, see the accompanying
4	5	file COPYING or http://www.opensource.org/licenses/mit-license.php.
5	1	Write and commit header, data

Figure 14. The process of data collecting and labelling.

[23] https://github.com/Jianwen-Xu/NLoN-PY

[24] https://pypi.org/project/nlon-py/

[25] https://github.com/bitcoin/bitcoin

Regarding the labelling process, Mäntylä et al. (2018) classified the original three data sources with tag [1: 'NL', 2: 'Not'], this study extended the tags to 7 categories:

[1: 'NL', 2: 'CODE', 3:'TRACE', 4: 'LOG', 5: 'NL_CODE', 6: 'NL_TRACE', 7: 'NL_LOG']

The example data and labels are shown in Table 5, the labelled data include three columns, *i.e.*, "Index", "Class", and "Text". In total, 8000 lines of text were labelled.

Table 5. Data source from different software repositories.

	Source	**Type**	**Examples**	**Lines**
1	Mozilla Firefox	Issues comment	1 attribute minimization. XML dictates selected="selected". 4 User Agent: Mozilla/5.0 (Mobile; rv:33.0) Gecko/33.0 Firefox/33.0 2 <option value="manage_addProduct/KnowledgeKit/HOWTO_factory">HOW 4 Error: uncaught exception: [Exception... "'[JavaScript Error: "aTabs[ex - 1] is	2000
2	Kuberne tes	Slack chats	1 An ingress is a resource used by kubernetes to indicate a configuration for inb 2 Max replicas: 15 5 Right, this exist, but when i put traefik-web-ui.kube-system in browser show [3 Client Version: version.Info{Major:1, Minor:8, GitVersion:v1.8.1, GitCommit:f	2000
3	Apache Lucene	Mailing list	2 > private void refineFacets(ResponseBuilder r 1 I was thinking of Tomcat at first since some people will start with 5 > cross-referenced with the java.util.concurrent package. Just amazing. 2 > Key: SOLR-1120	2000
4	Bitcoin	Code comments	1 Distributed under the MIT software license, see the accompanying 5 file COPYING or http://www.opensource.org/licenses/mit-license.php. 1 Write and commit header, data 1 Generate random temporary filename	2000

4.2 Text representation and Feature extraction

Based on the original paper, Character tri-grams and feature engineering were used to represent the text corpus. Moreover, standardization was introduced in the new design of this study to preprocess data for adapting to some learning models (such as the RBF kernel of SVMs or the l1 and l2 regularizers of linear models) which may assume the features centred around zero (Standardization).

Formula (2) presents the process of standardizing:

Value of sample x,

Number of samples N

Mean μ, Standard Deviation σ

$$\mu = \frac{\sum x_i}{N} \qquad \sigma = \sqrt{\frac{\sum (x_i - \mu)^2}{N}} \tag{2}$$

Standardized Value s

$$s_i = \frac{x_i - \mu}{\sigma}$$

This study used the same 11 text features as the original paper in the feature extraction process (see Table 5). The main modification in the implementation is to replace the use of the regular expression with the built-in function of Python. The following code demonstrates the difference between the two implementations:

```
## The R code of NLoN
```

```
CapsRatio <- function(text) Caps(text) / nchar(text)

Caps <- function(text) {
  .CountRegexMatches(text, "[A-Z]")
}

.CountRegexMatches <- function(text, re) {
  sapply(stringr::str_match_all(text, re), length)
}

// The Python code of NLoN-PY

def CapsRatio(self, text):
      return len([c for c in text if c.isupper()]) / self.charCount(text)

self.charCount = lambda x: len(x)
```

4.3 Model selection and Parameter tuning

This study compares the original model (Glmnet) and three popular multi-class learning models, *i.e.*, Naive Bayes (NB), k-NN, and SVM. Area Under the Receiver Operating Characteristic Curve (ROC AUC) and F1 score were used to validate the performance of classification models. AUC is a widely used measure of the performance of supervised classification rules, Hand and Till (2001) extended the scope of application from two-class cases to multi-class cases. Likewise, the F1 score is the harmonic mean of precision and recall, see Formula (3). Precision is the fraction of relevant instances among the retrieved instances, while recall is the fraction of relevant instances that were retrieved. The F1 score used in this study is 'macro', *i.e.*, calculate metrics for each class and find their average value.

$$F1\,score = 2 \cdot \frac{precision \cdot recall}{precision + recall} = \frac{tp}{tp + \frac{1}{2}(fp + fn)} \qquad (3)$$

The three models were trained with the default parameters on the 60% dataset and tested on the left 40% dataset. The training time cost was also recorded as another performance, the results were displayed in Table 6. Both NB and SVM achieved acceptable AUC scores, however, considering the diversity of parameter adjustment and F1 score, SVM was finally selected as the main research model. Nevertheless, the performance of Glmnet could be a baseline for the following training. On the other hand, for the SVM model, the time cost of training should be also considering in the optimize process.

Table 6. Comparison of three models on test size 40%.

	Glmnet	NB	k-NN	SVM
AUC	0.838	0.836	0.797	0.858
F1	0.767	0.656	0.759	0.770
Training Time (s)	18.463	23.396	16.733	602.797

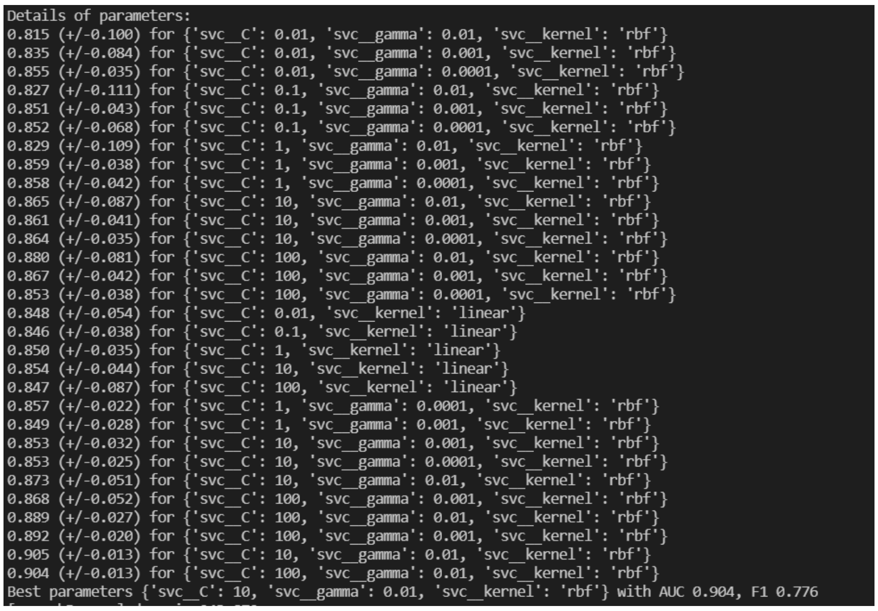
After the model selection, the hyper-parameters (parameters that are not directly learnt by models) can be searched by using cross-validation. Regarding the SVM model, 'C' 'gamma' and 'kernel' are the hyper-parameters that need tuning. Every parameter can have several candidates as presented in the following code:

```
C_range = [0.01, 0.1, 1, 10, 100]
Gamma = [1e-2, 1e-3, 1e-4]
param_grid = [{'kernel': ['rbf'], 'gamma': Gamma, 'C': C_range},
              {'kernel': ['linear'], 'C': C_range}]
```

The popular Python package scikit-learn[26] provided a successive halving (SH) search strategy for search on specified parameters, *i.e.*, evaluating all the candidates with a small number of resources and iteratively selects the best candidates, using more and more resources (HalvingGridSearchCV). Figure 15 shows that the best parameters achieved performance AUC=0.904 and F1=0.776, the winning combination of candidates is:

```
{'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
```

The parameter set would be the initial setting for the learning model, however, this process would repeat several times if the following validation results can not meet the expected requirements.



```
Details of parameters:
0.815 (+/-0.100) for {'svc__C': 0.01, 'svc__gamma': 0.01, 'svc__kernel': 'rbf'}
0.835 (+/-0.084) for {'svc__C': 0.01, 'svc__gamma': 0.001, 'svc__kernel': 'rbf'}
0.855 (+/-0.035) for {'svc__C': 0.01, 'svc__gamma': 0.0001, 'svc__kernel': 'rbf'}
0.827 (+/-0.111) for {'svc__C': 0.1, 'svc__gamma': 0.01, 'svc__kernel': 'rbf'}
0.851 (+/-0.043) for {'svc__C': 0.1, 'svc__gamma': 0.001, 'svc__kernel': 'rbf'}
0.852 (+/-0.068) for {'svc__C': 0.1, 'svc__gamma': 0.0001, 'svc__kernel': 'rbf'}
0.829 (+/-0.109) for {'svc__C': 1, 'svc__gamma': 0.01, 'svc__kernel': 'rbf'}
0.859 (+/-0.038) for {'svc__C': 1, 'svc__gamma': 0.001, 'svc__kernel': 'rbf'}
0.858 (+/-0.042) for {'svc__C': 1, 'svc__gamma': 0.0001, 'svc__kernel': 'rbf'}
0.865 (+/-0.087) for {'svc__C': 10, 'svc__gamma': 0.01, 'svc__kernel': 'rbf'}
0.861 (+/-0.041) for {'svc__C': 10, 'svc__gamma': 0.001, 'svc__kernel': 'rbf'}
0.864 (+/-0.035) for {'svc__C': 10, 'svc__gamma': 0.0001, 'svc__kernel': 'rbf'}
0.880 (+/-0.081) for {'svc__C': 100, 'svc__gamma': 0.01, 'svc__kernel': 'rbf'}
0.867 (+/-0.042) for {'svc__C': 100, 'svc__gamma': 0.001, 'svc__kernel': 'rbf'}
0.853 (+/-0.038) for {'svc__C': 100, 'svc__gamma': 0.0001, 'svc__kernel': 'rbf'}
0.848 (+/-0.054) for {'svc__C': 0.01, 'svc__kernel': 'linear'}
0.846 (+/-0.038) for {'svc__C': 0.1, 'svc__kernel': 'linear'}
0.850 (+/-0.035) for {'svc__C': 1, 'svc__kernel': 'linear'}
0.854 (+/-0.044) for {'svc__C': 10, 'svc__kernel': 'linear'}
0.847 (+/-0.087) for {'svc__C': 100, 'svc__kernel': 'linear'}
0.857 (+/-0.022) for {'svc__C': 1, 'svc__gamma': 0.0001, 'svc__kernel': 'rbf'}
0.849 (+/-0.028) for {'svc__C': 1, 'svc__gamma': 0.001, 'svc__kernel': 'rbf'}
0.853 (+/-0.032) for {'svc__C': 10, 'svc__gamma': 0.001, 'svc__kernel': 'rbf'}
0.853 (+/-0.025) for {'svc__C': 10, 'svc__gamma': 0.0001, 'svc__kernel': 'rbf'}
0.873 (+/-0.051) for {'svc__C': 10, 'svc__gamma': 0.01, 'svc__kernel': 'rbf'}
0.868 (+/-0.052) for {'svc__C': 100, 'svc__gamma': 0.001, 'svc__kernel': 'rbf'}
0.889 (+/-0.027) for {'svc__C': 100, 'svc__gamma': 0.01, 'svc__kernel': 'rbf'}
0.892 (+/-0.020) for {'svc__C': 100, 'svc__gamma': 0.001, 'svc__kernel': 'rbf'}
0.905 (+/-0.013) for {'svc__C': 10, 'svc__gamma': 0.01, 'svc__kernel': 'rbf'}
0.904 (+/-0.013) for {'svc__C': 100, 'svc__gamma': 0.01, 'svc__kernel': 'rbf'}
Best parameters {'svc__C': 10, 'svc__gamma': 0.01, 'svc__kernel': 'rbf'} with AUC 0.904, F1 0.776
```

Figure 15. Search parameters with SH strategy.

4.4 K-fold Cross-Validation

4.4.1 Validate 2-class performance

As for validating the performance of NLoN, the original paper performed 10-fold cross-validation in the six thousand samples of all three data sources and achieved performance 0.982 in the ROC curve (AUC) score. Although this study focused on multi-class classification, the performance on both 2-class and multi-class tasks were validated by 10-fold cross-validation. As shown in Figure 16, the new classifier achieved an AUC

[26] https://scikit-learn.org/stable/

performance of 0.93 for 40% test data and a mean 0.92 ± 0.01 for 10-fold cross-validation on the 2-class classification task.

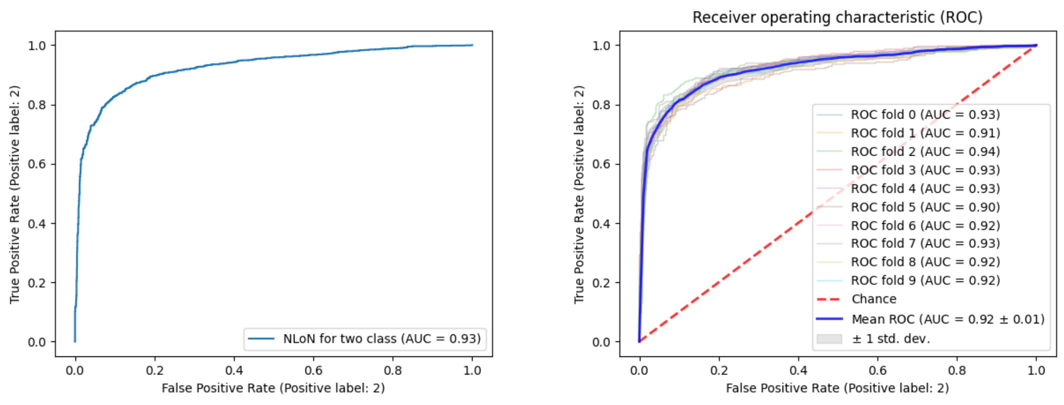


Figure 16. 2-class classifier score for test size 40% (left) and 10-fold Cross-Validation (right).

4.4.2 Validate multi-class performance

The multi-class classification also used the 10-fold cross-validation as the evaluation method, besides the confusion matrix was also used to evaluate the accuracy of the multi-class classification. Figure 17 illustrated the confusion matrix for the 5-class classification task performance, both the NL and Code category achieved good accuracy, though the most misclassification happened between NL and NL_Code (203 false positives). This problem would be discussed in the next chapter.

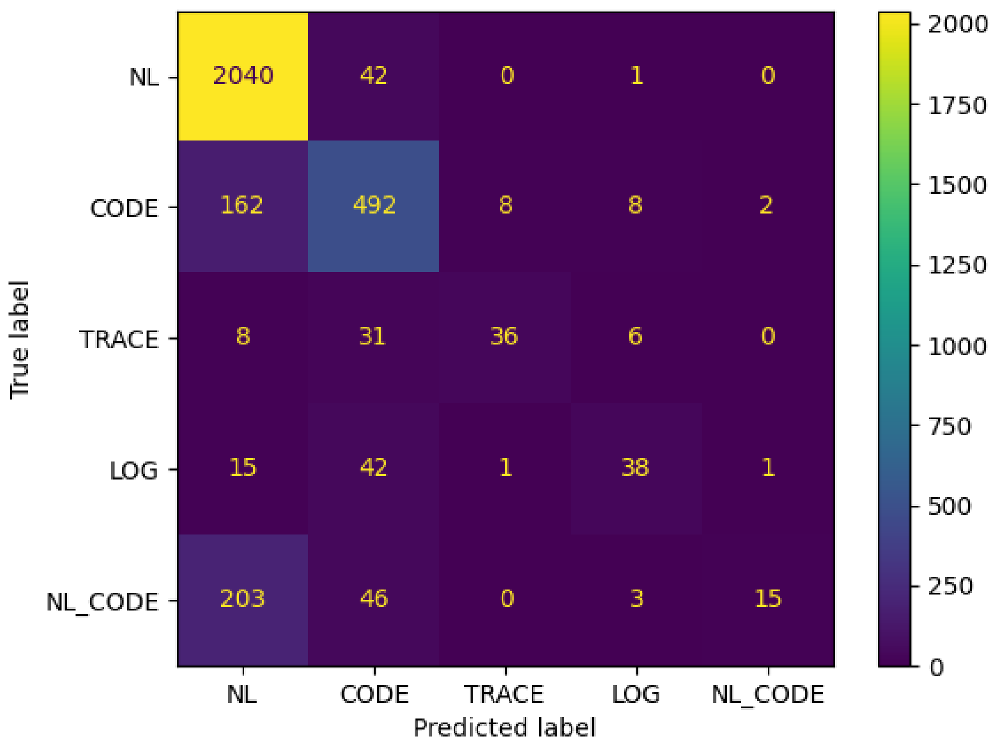


Figure 17. 5-class confusion matrix.

5. Findings and Discussion

5.1 Data Imbalance

The imbalanced distribution of data was found due to the validation score differed significantly in a range from 0.39 to 0.91 on different classes (see Figure 18). The mixed classes (NL_Code 0.38, NL_Trace 0.40, NL_Log 0.39) achieved less than half of the other four classes (NL 0.91, Code 0.90, Trace 0.92, Log 0.88).

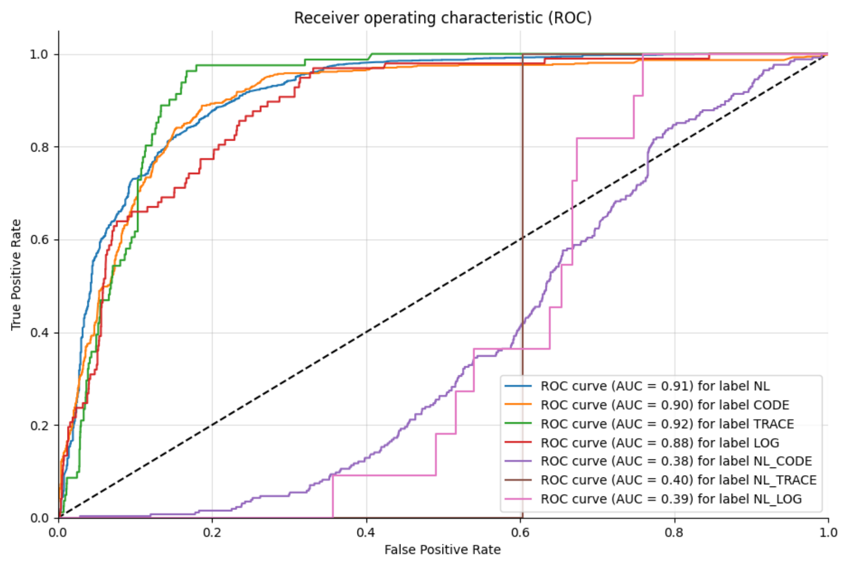


Figure 18. The performance (AUC) on 7-class classification.

The data of four sources were analyzed to detect whether the distribution was balanced, as shown in Table 6, the least two classes only have 2 and 28 samples which are not sufficient for providing good performance in machine learning. However, the data volume and performance are not directly proportional, classes with fewer samples can likewise achieve considerable performance, *e.g.*, Stack Trace (samples=201, AUC=0.92) and Mix NL Code (samples=638, AUC=0.38).

Table 6. Distribution of each data source.

	Natural Language	**Code**	**Stack Trace**	**Error/Log**	**Mix NL and Code**	**Mix NL and Trace**	**Mix NL and Log**
Mozilla Firefox	1187	482	90	162	78	0	1
Kubernetes	1425	337	2	59	151	1	25
Apache Lucene	1039	760	109	22	67	1	2
Bitcoin	1557	101	0	0	342	0	0
Total	5208	1680	201	243	638	2	28
AUC	0.91	0.90	0.92	0.88	0.38	0.40	0.39

The relative trend of the two data indicators is shown in Figure 19, it is apparent that the performance of the mixed categories is generally poorer than that of the unmixed categories.

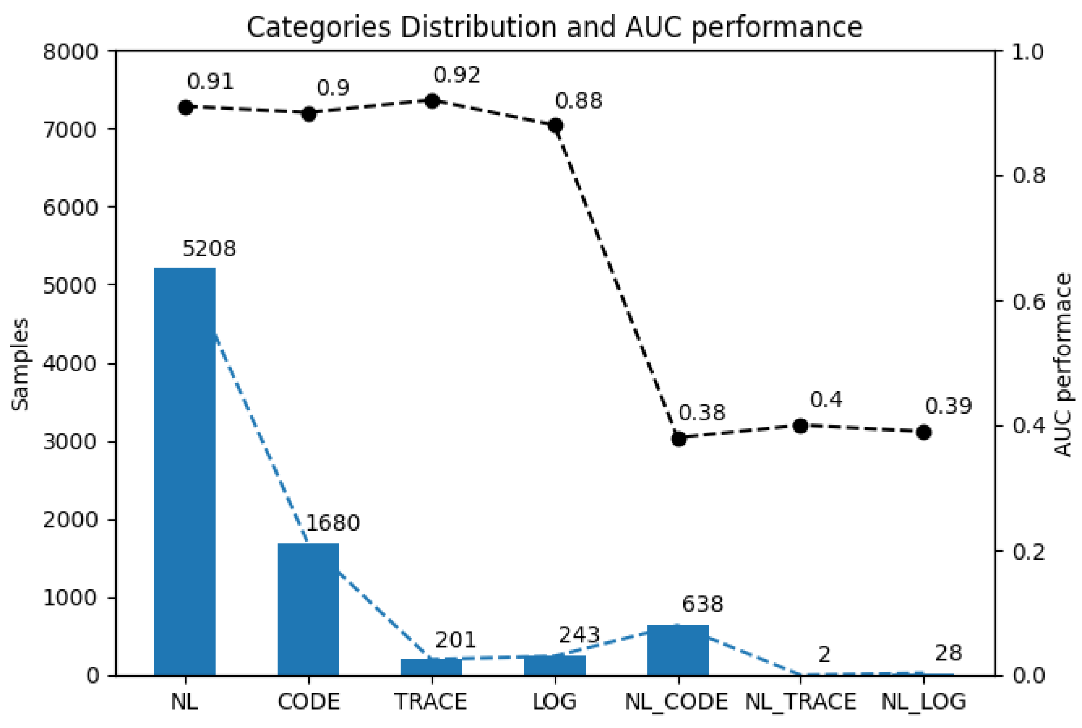


Figure 19. The distribution of categories.

This discovery raises a new question as to whether the originally designed classification (7-class) is reasonable enough (see Figure 13). To locate this problem, the detailed distribution of each category was also analyzed (see Figure 20). According to the diagram, it can be found that natural language and code occupy most of each category, besides, the mixture of natural language and stack trace, log messages is seldom detected.

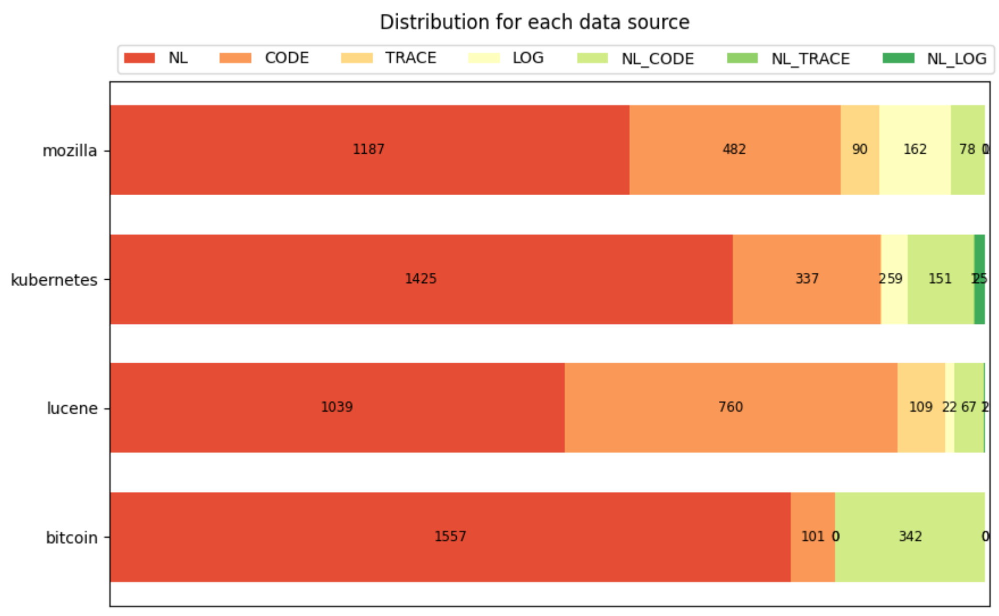


Figure 20. The distribution of categories for each data source.

According to the distribution of the data, the categories of classification can be refined, *i.e.*, eliminate the two mixed classes originally designed (Mix NL Trace and Mix NL Log). The refined category (5-class) is shown in Figure 21.

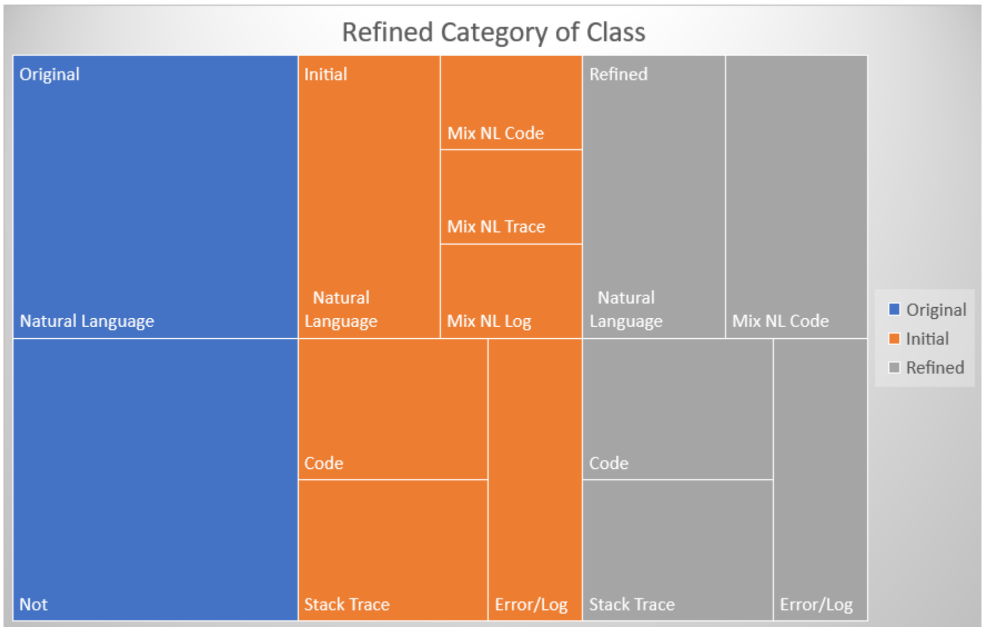


Figure 21. The refined design of categories.

After the design was modified, the model was also learned for the new classification, and the preliminary results showed a significant performance improvement (see Figure 22). The 5-class design achieved better performance than 7-class design (see Figure 18), *i.e.*, NL (samples=5208, AUC=0.91), Code (samples=1680, AUC=0.91), Stack Trace (samples=201, AUC=0.95), Log (samples=243, AUC=0.91), Mixed NL and Code (samples=668, AUC=0.53).

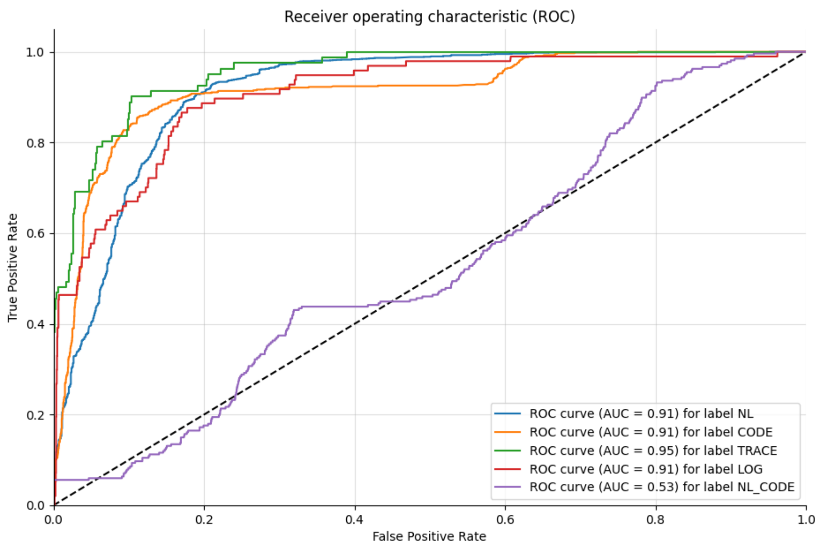


Figure 22. The performance (AUC) on 5-class classification.

5.2 Research Questions

RQ1: How to design and implement a multi-class classification approach based on previous work?

The multi-class classification algorithm implemented in this research is based on two-class classification, *i.e.*, build one two-class classifier for each category to distinguish this category from all other categories. This strategy is also known as one-vs-rest (ovr). In contrast, the one-vs-one (ovo) strategy is to construct one two-class classifier for each pair of categories. Because the number of classifiers to be constructed is different, *i.e.*, if the number of categories is N then ovr needs N classifiers and ovo needs N*(N-1)/2 classifiers, the computational efficiency of ovr $O(n)$ is usually better than ovo $O(n^2)$. Besides, another advantage of ovr is better interpretability, because each class has a unique classifier, and it is easier to calculate the score of each class when verifying performance. In Figure 22, The ROC curve of each category well illustrates the performance of each classifier.

In practice, this study suggested that the 'decision_function_shape' parameter was set to 'ovr' when using the learning models of the scikit-learn library although the default strategy for many multi-class models in the library is ovr.

RQ2: Whether other algorithms can get better performance than Glmnet without breaking the original principle of lightweight design?

This study compared the performance of Glmnet with other algorithms such as Naive Bayes, k-Nearest Neighbours, and Support Vector Machine (see Table 6). According to the experimental results, it can be seen that there is no significant performance improvement. Although the performance of SVM is slightly higher, it can be seen from the extra cost of model training time (from 18s to 602s) that the original design is more lightweight. However, as a user, these performance issues may not be perceived. Due to the use of model persistence technology, after parameter tuning, the model will fit the training data, and the data of this model will be persisted in joblib format. A final classifier is a tool that can be used directly, and the user does not need to retrain the model to use it.

5.3 Limitations

Human bias may be the main limitation of this research. Since the labelled data are all done manually by the researcher, the original paper was independently labelled by two researchers, but this research was labelled by the author alone. Especially for the labelling of mixed categories, the mixing degree of natural language and code may not be completely consistent. In many cases, a sentence may only contain one or two function names, even if it is labelled as a natural language which could be also reasonable. Among the 8000 corpus labels, the consistency of manual labelling cannot be guaranteed in this study. Another limitation is that the current design does not compare neural networks or unsupervised learning algorithms. The model learning process of the two will be quite different from the design of this research. This research is still biased towards relatively traditional supervised machine learning algorithms.

6. Conclusions

In this paper, I designed and implemented NLoN-PY, a tool used to classify natural languages and programming languages and related derivative texts. Three multi-category classification algorithms (Naive Bayes, k-NN, and SVM) have been compared and verified. At the same time, the main purpose is achieved, which is to expand the scope of application of the original papers while keeping the weight as light as possible. The 10-fold cross-validation achieved AUC performance 0.901 for the 5-class classification task and the AUC performance of 0.92 for the 2-class task.

However, this study has limitations on the multi-class definition task. The imbalance of data may prove that the practical communication in software repositories does not use mixed language extensively to describe problems or make comments. The definition of categories need larger data to support the rationality, and it was initially obtained through the combination of natural language and not classes. Another limitation is that human bias may be introduced in the data labelling process, there are ambiguous choices in expanding the labels to multi-category marking.

Regarding future research, I suggest labelling more data and try to ensure that the data labelling does not introduce human bias. In addition, regarding machine learning algorithms, the direction of deep learning may be considered. When I was doing the literature review, I could already feel the popularity of this trend, whether it is neural networks or unsupervised learning are directions that can be tried. The interdisciplinary research of natural language processing and mining software repositories is also the future direction.

References

Alon, U., Zilberstein, M., Levy, O., & Yahav, E. (2019). code2vec: learning distributed representations of code. *Proceedings of the ACM on Programming Languages*, *3*(POPL), 1–29. doi: 10.1145/3290353

Bacchelli, A., Dal Sasso, T., D'Ambros, M., & Lanza, M. (2012). Content classification of development emails. *Proceedings - International Conference on Software Engineering*, 375–385. doi: 10.1109/ICSE.2012.6227177

Bajracharya, S., & Lopes, C. (2009). Mining search topics from a code search engine usage log. *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories, MSR 2009*, 111–120. doi: 10.1109/MSR.2009.5069489

Bangash, A. A., Sahar, H., Chowdhury, S., Wong, A. W., Hindle, A., & Ali, K. (2019). What do developers know about machine learning: A study of ML discussions on stackoverflow. *IEEE International Working Conference on Mining Software Repositories*, *2019-May*, 260–264. doi: 10.1109/MSR.2019.00052

Baysal, O., Holmes, R., & Godfrey, M. W. (2012). Mining usage data and development artifacts. *IEEE International Working Conference on Mining Software Repositories*, 98–107. doi: 10.1109/MSR.2012.6224305

Begel, A., Phang, K. Y., & Zimmermann, T. (2010). Codebook: Discovering and exploiting relationships in software repositories. *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE '10,* 125-134. doi:10.1145/1806799.1806821

Belinkov, Y., & Glass, J. (2019). Analysis methods in neural language processing: A survey. *Transactions of the Association for Computational Linguistics, 7*, 49-72. doi:10.1162/tacl_a_00254

Bengio, Y., Ducharme, R., Vincent, P., & Janvin, C. (2003). A neural probabilistic language model. *The journal of machine learning research*, *3*, 1137-1155.

Bettenburg, N., Premraj, R., Zimmermann, T., & Kim, S. (2008). Extracting structural information from bug reports. *Proceedings of the 2008 International Workshop on Mining Software Repositories - MSR '08,* 27-30. doi:10.1145/1370750.1370757

Bonaccorso, G. (2017). *Machine learning algorithms*. Packt Publishing Ltd.

Brown, P. F., DeSouza, P. V, Mercer, R. L., Della Pietra, V. J., & Lai, J. C. (1992). Class-based n-gram models of natural language. *Computational linguistics, 18(4)*, 467-480.

Cambria, E., & White, B. (2014). Jumping NLP Curves: A Review of Natural Language Processing Research [Review Article]. *IEEE Computational Intelligence Magazine*, *9*(2), 48-57. doi: 10.1109/MCI.2014.2307227

Cavnar, W. B., & Trenkle, J. M. (1994). N-Gram-Based Text Categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, 161-175.

Chang, C. C., & Lin, C. J. (2011). LIBSVM: A Library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, *2*(3), 1–27. doi: 10.1145/1961189.1961199

Chatterjee, P., Damevski, K., Pollock, L., Augustine, V., & Kraft, N. A. (2019). Exploratory study of slack Q&A chats as a mining source for software engineering tools. *IEEE International Working Conference on Mining Software Repositories*, *2019-May*, 490–501. doi: 10.1109/MSR.2019.00075

Chatterjee, P., Damevski, K., Kraft, N. A., & Pollock, L. (2020). Software-related Slack Chats with Disentangled Conversations. *Proceedings - 2020 IEEE/ACM 17th International Conference on Mining Software Repositories, MSR 2020*, 588–592. doi: 10.1145/3379597.3387493

Chen, A., Chou, E., Wong, J., Yao, A., Zhang, Q., Zhang, S., & Michail, A. (2001). CVSSearch: Searching through source code using CVS COMMENTS. *Proceedings IEEE International Conference on Software Maintenance. ICSM 2001,* 364-373. doi:10.1109/icsm.2001.972749

Chowdhury, S. A., & Hindle, A. (2015). Mining StackOverflow to filter out off-topic IRC discussion. *IEEE International Working Conference on Mining Software Repositories*, *2015-Augus*, 422–425. doi: 10.1109/MSR.2015.54

Collard, M. L., Decker, M. J., & Maletic, J. I. (2013). SrcML: An infrastructure for the exploration, analysis, and manipulation of source code: A tool demonstration. *IEEE International Conference on Software Maintenance, ICSM*, 516–519. doi: 10.1109/ICSM.2013.85

Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, *13*(1), 21–27. doi: 10.1109/TIT.1967.1053964

De Farias, M. A. F., Colaço, M., Mendonça, M., Novais, R., Da Silva Carvalho, L. P., & Spínola, R. O. (2016). A systematic mapping study on mining software repositories. *Proceedings of the ACM Symposium on Applied Computing*, *04-08-April-2016*, 1472–1479. doi: 10.1145/2851613.2851786

Demeyer, S., Murgia, A., Wyckmans, K., & Lamkanfi, A. (2013). Happy birthday! a trend analysis on past msr papers. *2013 10th Working Conference on Mining Software Repositories (MSR),* 353-362. doi:10.1109/msr.2013.6624049

Devanbu, P. (1992). GENOA: A customizable language- and front-end independent code analyzer. *International Conference on Software Engineering,* 307-317. doi:10.1109/icse.1992.753508

Dumais, S., Platt, J., Heckerman, D., & Sahami, M. (1998). Inductive learning algorithms and representations for text categorization. *Proceedings of the Seventh International Conference on Information and Knowledge Management - CIKM '98*, 148–155. New York, New York, USA: Association for Computing Machinery (ACM). doi: 10.1145/288627.288651

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 1189-1232.

Friedman, J., Hastie, T., & Tibshirani, R. (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. *Journal of Statistical Software*, *33*(1), 1–22. doi: 10.18637/jss.v033.i01

Fu, Q., Lou, J. G., Lin, Q., Ding, R., Zhang, D., & Xie, T. (2013). Contextual analysis of program logs for understanding system behaviors. *IEEE International Working Conference on Mining Software Repositories*, 397–400. doi: 10.1109/MSR.2013.6624054

Fucci, D., Mollaalizadehbahnemiri, A., & Maalej, W. (2019). On using machine learning to identify knowledge in API reference documentation. *ESEC/FSE 2019 - Proceedings of the 2019 27th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, *19*, 109–119. New York, NY, USA: Association for Computing Machinery, Inc. doi: 10.1145/3338906.3338943

German, D. M., Cubranić, D., & Storey, M. D. (2005). A framework for describing and understanding mining tools in software development. *Proceedings of the 2005 International Workshop on Mining Software Repositories - MSR '05,* 95-99. doi:10.1145/1083142.1083160

Good, I. J. (1950). *Probability and the Weighing of Evidence* (No. 519.2/G646). London: C. Griffin.

Guzzi, A., Bacchelli, A., Lanza, M., Pinzger, M., & Van Deursen, A. (2013). Communication in open source software development mailing lists. *2013 10th Working Conference on Mining Software Repositories (MSR)*, 277–286. doi: 10.1109/MSR.2013.6624039

Han, E., Karypis, G., & Kumar, V. (1999). *Text Categorization Using Weight Adjusted k-Nearest Neighbor Classification*. Retrieved from the University of Minnesota Digital Conservancy, https://hdl.handle.net/11299/215376

Hand, D. J., & Till, R. J. (2001). A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems. *Machine Learning*, *45*(2), 171–186. doi: 10.1023/A:1010920819831

Harris, Z. S. (1954). Distributional Structure. *WORD*, *10*(2–3), 146–162. doi: 10.1080/00437956.1954.11659520

Hassan, A. E. (2008). The road ahead for mining software repositories. *2008 Frontiers of Software Maintenance*. doi:10.1109/fosm.2008.4659248

HalvingGridSearchCV. (n.d.). Retrieved July 21, 2021, from https://scikit-learn.org/stable/modules/grid_search.html#successive-halving-user-guide

Hemmati, H., Nadi, S., Baysal, O., Kononenko, O., Wang, W., Holmes, R., & Godfrey, M. W. (2013). The msr cookbook: Mining a decade of research. *2013 10th Working Conference on Mining Software Repositories (MSR)*. doi:10.1109/msr.2013.6624048

Hevner, A., & Chatterjee, S. (2010). *Design Research in Information Systems*. Boston, MA: Springer US. doi: 10.1007/978-1-4419-5653-8

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly: Management Information Systems*, *28*(1), 75–105. doi: 10.2307/25148625

Ho, T. K. (1995). Random decision forests. *Proceedings of 3rd International Conference on Document Analysis and Recognition*, *1*, 278–282. IEEE Comput. Soc. Press. doi: 10.1109/ICDAR.1995.598994

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. doi: 10.1162/neco.1997.9.8.1735

Howard, M. J., Gupta, S., Pollock, L., & Vijay-Shanker, K. (2013). Automatically mining software-based, semantically-similar words from comment-code mappings. *2013 10th Working Conference on Mining Software Repositories (MSR),* 377-386. doi:10.1109/msr.2013.6624052

Ibrahim, W. M., Bettenburg, N., Shihab, E., Adams, B., & Hassan, A. E. (2010). Should I contribute to this discussion? *Proceedings - International Conference on Software Engineering*, 181–190. doi: 10.1109/MSR.2010.5463345

Indurkhya, N., & Damerau, F. J. (2010). *Handbook of natural language processing* (2nd ed.). Boca Raton, Florida: Taylor & Francis. doi:10.1201/9781420085938

Jauhiainen, T. , Linden, K., & Jauhiainen, H. (2017). Evaluation of language identification methods using 285 languages. In *21st Nordic Conference of Computational Linguistics Proceedings of the Conference,* 183-191. Linköping University Electronic Press.

Joachims, T. (1997). A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, 143-151.

Joachims, T. (1998). Text categorization with Support Vector Machines: Learning with many relevant features. *Machine Learning: ECML-98,* 137-142. doi:10.1007/BFb0026683

Jones, K. S. (1994). Natural Language Processing: A Historical Review. In *Current Issues in Computational Linguistics: In Honour of Don Walker* (Vol. 28, pp. 3–16). Dordrecht: Springer Netherlands. doi: 10.1007/978-0-585-35958-8_1

Jordan, M. I., & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science, 349*(6245), 255-260. doi:10.1126/science.aaa8415

Júnior, M. C., Mendonca, M., Farias, M., & Henrique, P. (2010). OSS developers context-specific Preferred Representational systems: A initial Neurolinguistic text analysis of the Apache mailing list. *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, 126–129. doi:10.1109/msr.2010.5463339

Jurafsky, D., & Martin, J. H. (2009). *Speech and Language Processing (2Nd Edition).* Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Kagdi, H., Collard, M. L., & Maletic, J. I. (2005). Towards a taxonomy of approaches for mining of source code repositories. *Proceedings of the 2005 International Workshop on Mining Software Repositories - MSR '05,* 90-94. doi:10.1145/1083142.1083159

Kagdi, H., Collard, M. L., & Maletic, J. I. (2007). A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of Software Maintenance and Evolution: Research and Practice, 19*(2), 77-131. doi:10.1002/smr.344

Karatzoglou, A., Meyer, D., & Hornik, K. (2006). Support vector machines in R. *Journal of statistical software*, *15*(9), 1-28.

Khan, A., Baharudin, B., Lee, L. H., & Khan, K. (2010). A review of machine learning algorithms for text-documents classification. *Journal of advances in information technology, 1(1)*, 4-20. doi: 10.4304/jait.1.1.4-20

Kim, M., & Notkin, D. (2006). Program element matching for multi-version program analyses. *Proceedings of the 2006 International Workshop on Mining Software Repositories - MSR '06,* 58-64. doi:10.1145/1137983.1137999

Ko, A., Aung, H., & Myers, B. (2005). Eliciting design requirements for maintenance-oriented ides: A detailed study of corrective and perfective maintenance tasks. *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.,* 126-135. doi:10.1109/icse.2005.1553555

Kotsiantis, S. B. (2007). Supervised machine learning: A review of classification techniques. *Informatica (Ljubljana)*, Vol. 31, pp. 249–268. doi: 10.31449/inf.v31i3.148

Lang, K. (1995). NewsWeeder: Learning to Filter Netnews. In *Machine Learning Proceedings 1995* (pp. 331–339). Elsevier. doi: 10.1016/b978-1-55860-377-6.50048-7

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., & Jackel, L. (1990). Handwritten digit recognition with a back-propagation network. *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, *2*, 396–404.

M3SOulu. (2018). M3SOulu/NLoN. Retrieved April 04, 2021, from https://github.com/M3SOulu/NLoN

Mäntylä, M. V., Calefato, F., & Claes, M. (2018). Natural language or not (NLON): A package for software engineering text analysis pipeline. *Proceedings of the 15th International Conference on Mining Software Repositories*. doi:10.1145/3196398.3196444

McCallum, A., & Nigam, K. (1998). A Comparison of Event Models for Naive Bayes Text Classification. In *Learning for Text categorization: Papers from the 1998 Workshop*, 41-48. Menlo Park, CA: AAAI Press.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings*. Retrieved from http://ronan.collobert.com/senna/

Moonen, L. (2001). Generating robust parsers using island grammars. *Proceedings Eighth Working Conference on Reverse Engineering,* 13-22. doi:10.1109/wcre.2001.957806

Murthy, S. K. (1998). Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, *2*(4), 345–389. doi: 10.1023/A:1009744630224

Nagappan, M., & Vouk, M. A. (2010). Abstracting log lines to log event types for mining software system logs. *Proceedings - International Conference on Software Engineering*, 114–117. doi: 10.1109/MSR.2010.5463281

Nazar, N., Hu, Y., & Jiang, H. (2016). Summarizing software artifacts: A literature review. *Journal of Computer Science and Technology, 31*(5), 883-909. doi:10.1007/s11390-016-1671-1

Parnin, C., & Görg, C. (2008). Improving change descriptions with change contexts. *Proceedings of the 2008 International Workshop on Mining Software Repositories - MSR '08,* 51-60. doi:10.1145/1370750.1370765

Pascarella, L. (2018). Classifying code comments in Java mobile applications. *Proceedings - International Conference on Software Engineering*, 39–40. New York, NY, USA: IEEE Computer Society. doi: 10.1145/3197231.3198444

Paul, S., & Prakash, A. (1994). A framework for source code search using program patterns. *IEEE Transactions on Software Engineering, 20*(6), 463-475. doi:10.1109/32.295894

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, *24*(3), 45–77. doi: 10.2753/MIS0742-1222240302

Rahman, M., Rigby, P., Palani, D., & Nguyen, T. (2019). Cleaning StackOverflow for machine translation. *IEEE International Working Conference on Mining Software Repositories*, *2019-May*, 79–83. doi: 10.1109/MSR.2019.00021

Rigby, P. C., & Hassan, A. E. (2007). What can OSS mailing lists tell us? A preliminary psychometric text analysis of the Apache developer mailing list. *Proceedings - ICSE 2007 Workshops: Fourth International Workshop on Mining Software Repositories, MSR 2007*. doi: 10.1109/MSR.2007.35

Robles, G. (2010). Replicating MSR: A study of the potential replicability of papers published in the Mining Software Repositories proceedings. *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010),* 171-180. doi:10.1109/msr.2010.5463348

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536. doi: 10.1038/323533a0

Ruiz, M. E., & Srinivasan, P. (1997). Automatic Text Categorization Using Neural Networks. *Advances in Classification Research Online*, *8*(1), 58–68. doi: 10.7152/acro.v8i1.12728

Ruparelia, N. B. (2010). The history of version control. *ACM SIGSOFT Software Engineering Notes, 35*(1), 5-9. doi:10.1145/1668862.1668876

Schipper, D., Aniche, M., & Van Deursen, A. (2019). Tracing back log data to its log statement: From research to practice. *IEEE International Working Conference on Mining Software Repositories*, *2019-May*, 545–549. doi: 10.1109/MSR.2019.00081

Scott, S., & Matwin, S. (1998). Text classification using WordNet hypernyms. In *Usage of WordNet in Natural Language Processing Systems*, 45–51.

Scott, S., & Matwin, S. (1999). Feature Engineering for Text Classification. In *Proceedings of the Sixteenth International Conference on Machine Learning*, 379-388.

Simon, H. A. (1996). *The Sciences of the Artificial (3rd ed.)*. Cambridge (MA): The MIT Press.

Śliwerski, J., Zimmermann, T., & Zeller, A. (2005). When do changes induce fixes? *Proceedings of the 2005 International Workshop on Mining Software Repositories - MSR '05,* 24-28. doi:10.1145/1083142.1083147

Standardization. (n.d.). Retrieved from https://scikit-learn.org/stable/modules/preprocessing.html

Tian, Y., Achananuparp, P., Lubis, I. N., Lo, D., & Lim, E. P. (2012). What does software engineering community microblog about? *IEEE International Working Conference on Mining Software Repositories*, 247–250. doi: 10.1109/MSR.2012.6224287

Tibshirani, R. (1996). Regression Shrinkage and Selection Via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, *58*(1), 267–288. doi: 10.1111/j.2517-6161.1996.tb02080.x

Vaarandi, R. (2003). A data clustering algorithm for mining patterns from event logs. *Proceedings of the 3rd IEEE Workshop on IP Operations and Management, IPOM 2003*, 119–126. doi: 10.1109/IPOM.2003.1251233

Vapnik, V. N. (1995). The Nature of Statistical Learning Theory. In *The Nature of Statistical Learning Theory*. Springer New York. doi: 10.1007/978-1-4757-2440-0

Wang, W., Malik, H., & Godfrey, M. W. (2015). Recommending posts concerning API issues in developer Q&A sites. *IEEE International Working Conference on Mining Software Repositories*, *2015-Augus*, 224–234. doi: 10.1109/MSR.2015.28

Weiss, C., Premraj, R., Zimmermann, T., & Zeller, A. (2007). How long will it take to fix this bug? *Fourth International Workshop on Mining Software Repositories (MSR'07:ICSE Workshops 2007)*. doi:10.1109/msr.2007.13

Yang, J., & Tan, L. (2012). Inferring semantically related words from software context. *2012 9th IEEE Working Conference on Mining Software Repositories (MSR),* 161-170. doi:10.1109/msr.2012.6224276

Yang, Y., & Liu, X. (1999). A re-examination of text categorization methods. *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR '99*, 42–49. New York, New York, USA: ACM Press. doi: 10.1145/312624.312647

Young, T., Hazarika, D., Poria, S., & Cambria, E. (2018). Recent trends in deep learning based natural language processing [review article]. *IEEE Computational Intelligence Magazine, 13*(3), 55-75. doi:10.1109/mci.2018.2840738

Zeller, A. (2009). *Why programs fail: A guide to systematic debugging* (2nd ed.). Elsevier/Morgan Kaufmann.

Zhai, J., Xu, X., Shi, Y., Tao, G., Pan, M., Ma, S., … Zhang, X. (2020). CPC: Automatically classifying and propagating natural language comments via program analysis. *Proceedings of the ACM/IEEE 42nd International Conference on*

Software Engineering, 1359–1371. New York, NY, USA: ACM. doi: 10.1145/3377811.3380427

Zhang, Y., Jin, R., & Zhou, Z. H. (2010). Understanding bag-of-words model: A statistical framework. *International Journal of Machine Learning and Cybernetics*, *1*(1–4), 43–52. doi: 10.1007/s13042-010-0001-0

Zhu, L., Spachos, P., Pensini, E., & Plataniotis, K. N. (2021). Deep learning and machine vision for food processing: A survey. *Current Research in Food Science*, *4*, 233–249. doi: 10.1016/j.crfs.2021.03.009