Conclusion Stability for Natural Language Based Mining of Design Discussions

by

Alvi Mahadi
B.Sc., Institute of Information Technology, Jahangirnagar University

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

in the Department of Computer Science

Conclusion Stability for Natural Language Based Mining of Design Discussions

by

Alvi Mahadi
B.Sc., Institute of Information Technology, Jahangirnagar University

Supervisory Committee

Dr. Neil A. Ernst, Supervisor
(Department of Computer Science)

Dr. Daniel German, Departmental Member
(Department of Computer Science)

ABSTRACT

Developer discussions range from in-person hallway chats to comment chains on bug reports. Being able to identify discussions that touch on software design would be helpful in documentation and refactoring software. Design mining is the application of machine learning techniques to correctly label a given discussion artifact, such as a pull request, as pertaining (or not) to design. In this work we demonstrate a simple example of how design mining works. We first replicate an existing state-of-the-art design mining study to show how conclusion stability is poor on different artifact types and different projects. Then we introduce two techniques—augmentation and context specificity—that greatly improve the conclusion stability and cross-project relevance of design mining. Our new approach achieves AUC-ROC of 0.88 on within dataset classification and 0.84 on the cross-dataset classification task.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

Dr. Neil A. Ernst, my supervisor, for his support, motivation, encouragement, patience, and mentoring throughout my Master's program. I personally thank him for giving me the opportunity to work with him. I am grateful for his continuous support and feedback that has helped me grow as a person and kept me going in those rough times.

My family, friends and all OCTERA, RIGI and PITA group members for supporting my research with ideas, suggestions and creating great moments throughout my degree.

João Brunet, Giovanni Viviani, and Robert Green for sharing their code, dataset and replication packages. I would also like to thank all the authors of previous work on design mining.

University of Victoria, for funding me with Research Assistantship (RA).

Chapter 1

Introduction

Design discussions are an important part of software development. Software design is a highly interactive process and many decisions involve considerable back and forth discussion. These decisions greatly impact software architecture [38, 87]. However, software design is a notoriously subjective concept. For the related term 'software architecture', for example, the Software Engineering Institute maintains a list of over 50 different definitions [73]). This subjectivity makes analyzing design decisions difficult [67]. Researchers have looked for ways in which design discussions could be automatically extracted from different types of software artifacts [12, 68, 1, 53, 89, 84, 83]. This automatic extraction, which we call *design mining*, is a subset of research based on mining software repositories. The potential practical relevance of research on design mining includes supporting design activities, improving traceability to requirements, enabling refactoring, and automating documentation. However, Design discussion recovery is different from design recovery itself i.e, we do not recover the fact that design is an instance of MVC, but we do recover discussions about how to implement MVC. This thesis only explains a better way of detecting or classifying discussions containing design points.

A design mining study uses a corpus consisting of *discussions*, in the form of software artifacts like pull requests, and manually labels those discussions targeting design topics, according to a coding guide. Machine learning classifiers such as support vector machines learn the feature space based on vectorization of the discussion and are evaluated using a predefined gold-set with metrics like the area under the ROC curve.

1.1 Motivation

The design of software controls various aspects of the system, such as performance, security, maintainability, etc. Yet, it becomes very difficult for the developers to maintain a consistent design by taking proper decisions.

Design erosion— is one of the most common problems in software engineering [81]. Most simpler design decisions can have severe consequences for the codebase and it often gets very difficult to understand the system as well as take new decisions without the knowledge of design changes in-between versions.

Sustainability— of many open source projects depend on the outside contributors who are often newcomers and lack of design documentation is one of the major barriers they face in the early stages [76, 75]. As a result, the contributors either leave the project or the maintainers have to spend more time in mentoring to keep them from leaving [71]. Also, delayed and failed contributions can slow down the growth and affect the quality of the project [14].

API learnability— also depends on the understanding of the high-level design [63]. It was also studied that linear representation of all the design decisions can improve the understandability of APIs for certain developers [65].

Most of these problems occur mainly because of the lack of information about the software's current design. For example, an untraceable design decision that can affect API usability from the documentation point-of-entry is one of the barriers to API learnability [63]. Although there are a number of formats to capture design decisions, it is still very difficult to record all the insights such as evolution stories and document rationale explicitly [64]. Also, the design documentation is often not kept up to date even if the decisions are recorded [54].

Such discussions are also one of the potential artifacts for newcomers to understand the architecture and design of the system [15]. However, these discussions about design are often scattered across different places such as commit messages, pull requests, and issue tracker comments. It is impractical for anyone to go through all the thousands of threads of discussions and find out the discussion about a particular design topic. Solving this problem is the challenge of what we call *design mining*, which is a branch of research based on mining software repositories. Being able to mine design discussions would lead to a novel approach of improved documentation, enabling improved traceability of requirements, refactoring and bug fixing support, and maintainability.

There are several approaches to extract design information from a software system. Project artifacts are very good sources to recover design information [9]. Several static analysis tools[1][2] are available to analyze the design flaws of software by comparing the design decisions with some predefined static rules. These tools are useful to express the difference between design in practice and ideal scenarios. This understanding is particularly important to know how the system currently works and help software developers to make any change in the system. But it does not provide the developer with the rationale of the design such as the intention behind a particular design decision.

Recent studies from Brunet et al. [12] and Shakiba [68] et al. have shown that developers are talking about design in various communication channels such as pull requests, issues, etc. and these discussions can be a great artifact to get ideas about the design changes and decisions taken [12], [68]. These discussions can also be analyzed to understand how a system evolves and can contain more information than the mechanism of the system [80]. Recent works also suggest that developer discussions can also contain reasons behind a particular design choice [85].

Suppose it is possible to extract design information from those developer discussions automatically and make a summary out of it. Because developer discussion is a continuous process, we could record the summary of those design decisions per release to maintain up-to-date design documentation completely without the need for human intervention. It is also possible to build an automatic tagging system that can tag the appropriate developer to review certain pull request based on previous reviews. This can also be taken to a step further to build a recommendation system that can suggest design choices based on recent decisions.

Automatic detection of design discussions can significantly reduce development time for both contributing developers as well as reviewers. It can also be used to enrich design information with ease which is a struggle for newcomers to an open-source project [76]. A recommender agent built on the detected design points can assist the core developers or maintainers to answer the question and queries from the newcomers. Because software design can be very subjective, findings from studies like this can potentially reveal several aspects of how and why design decisions divert from ideal design patterns. Moreover, these different opinions can also be analyzed to further modernize some of the trivial design ideas. Lastly, mining and summarizing

[1]https://www.sonarqube.org/
[2]https://embold.io/

the design discussion gives us a great opportunity to keep an up to date documentation with little to no manual effort and time.

1.2 Objectives

Producing a good-performing, validated classifier that can distinguish design discussions from non-design ones has been the main objective to date for design mining research. Apart from a validation study in Viviani et al. [83], however, the practical relevance (cf. [18]) of design mining has not been studied in detail. Practical relevance in the context of design mining means a classifier with broad applicability to different design discussions, across software projects and artifact types. This study's goal is a practically relevant classifier, which we could run on a randomly selected Github project and identify with high accuracy that project's design discussions.

Practical relevance can be seen as a form of external validity or generalizability, and the underlying concept is *conclusion stability*, after Menzies and Sheppherd [47]. Design mining research has to date performed poorly when applied to new datasets (which we elaborate on in a discussion of related work, following). This is problematic because positive, significant results in a single study are only of value if they lend confidence to scientific conclusions.

An important aspect of our view of a useful design mining classifier is its ability to work well across different domains. We define the source of the data as a domain. For example, if we source our data from Github, the domain of the data is Github. Project is a subset of a domain that is the name of the project to which the discussion belongs (ex. node.js, Rails, etc.). Cross-project/cross-domain transfer learning means training on one domain/project and transferring the knowledge to another domain/project, i.e., it shows good conclusion stability.

1.3 Problem Statement, Research Questions, and Approach

The simplest formulation of design mining is defined as classifying a developer discussion that can be extracted from various software artifacts (i.e., pull request or issue tracker) and labeling them as design topics. This classification process can be conducted both manually as well as automatically. Manual classification refers to

labeling the data using human effort by following a coding guide, while automatic classification leverages the potential of machine learning models and the advancement of natural language processing to classify the discussions according to some specific features. While automatic classification shines in the amount of time and effort needed, manual classification is found to be better for validation. Verification of the correctness of manual classification is achieved by meeting and agreement among the participants. On the other hand, validation of automatic classification is measured by evaluating the classifiers with manually labeled small sets of data, which is often referred to as the gold set. Almost all the studies in this field attempted to produce a best-performing and validated classifier [12, 68, 85]. Achieving practical relevance [18] still remains a challenge. Our aim is to achieve practical relevance by developing a model with wide applicability to different design discussions that could be used with high accuracy across different projects, platforms, and artifact types. By the end of this thesis, we explore the following research questions with approaches:

RQ-1 Is it possible to replicate a previous study and improve that study?

Approach— To tackle RQ-1, we conduct an operational replication of the pioneering design mining work of Brunet et al. [12]. We first replicate, as closely as possible, the study Brunet et al. conducted. We then try to improve on their results in design mining with new analysis approaches. We demonstrate improved ways to detect design discussions using word embedding and document vectors. Our approach results in an improved accuracy 95.12% compared to 87.60% from [12] with stratification of the data. There was no analysis done on Area Under the Curve (AUC) in the replicated paper, hence no AUC score was reported. We also show our results and discussion in AUC which is considered to be better validation criteria for imbalanced data and achieve 84% of AUC which is a significant improvement over our implementation of AUC on the replicated study at around 75%. All these things are discussed in detail in chapter 3.

RQ-2 To what extent can we transfer classifiers trained on one data set to other data sets?

Approach— For the RQ-2, we report on cross-domain transfer of a classifier. We begin with the approaches described in RQ-1, and apply them to new, out of sample datasets. We characterize the different datasets in this study to understand what commonalities and differences they exhibit. We describe the dataset in section 3.3. We discuss more on this with results in chapter 3.

RQ-3 How can we get more labeled data to train, validate, and test models of design mining?

Approach— We take a different approach from the previous studies to address RQ-3. While previous studies manually labeled the dataset for training and testing purposes, the small amount of data is always considered to be a limitation of those studies. We hypothesize that conversational data that are similar to developer discussions might work as the training data we need. Hence, we sourced our data from Stack Overflow conversations in the form of questions, answers and comments which are already tagged by several developers and moderators [7]. Although we understand that the Stack Overflow conversations are not directly comparable with developer discussions, the words of those posts often contain architecture-relevant knowledge [74]. Since this study is about classifying a discussion as design or non-design, conversational texts from Stack Overflow can provide those words that could be used to distinguish between design and non-design classes. We use this dataset only to train and validate our model but the actual testing of the model is conducted with the developer discussions dataset which we obtain from the study at [12]. This allowed us to obtain a dataset of 260,000 examples which is the largest dataset in design mining so far (the latest study from Viviani et al. [83] introduced a dataset of 2500 examples). This data can be used to training, validation, and testing purpose. Validation and testing might seem to be similar, however validation set is different from test set. Validation set actually can be regarded as a part of training set, because it is used to build the model, neural networks or others. It is usually used for parameter selection and to avoild overfitting. We explain the validity of this data elaborately in chapter 4 and illustrate some of the improvements we notice in the results section in chapter 5.

RQ-4 How useful are software-specific word vectorizers?

Approach— Converting words of a text conversation to vectors as feature space representation is a common practice in Natural Language Processing. Previous studies have introduced various vectorization techniques. In response of RQ-1, we demonstrate how word embedding as a vectorization choice can improve the performance of the classifier. However, word embedding needs a reference model. Initially, we use a general-purpose reference model that is trained on texts from Wikipedia. However, some of the software engineering

context can get lost if we use general purpose reference model [19]. For this reason, we decide to build our own reference model that is trained on software engineering related literature. We scrape the plain text from 300 books, conference and journal paper and develop a software-specific corpus to be used to train our software-specific word vectorizer. We train our software-specific word embedding reference model based on the corpus and test it's performance and validity with respect to the general-purpose reference model to address RQ-4. We explain the data collection method for this model elaborately in chapter 4 and the improvements in classification in chapter 5.

RQ-5 How to provide domain context to a small sample of data?

Approach— To answer RQ-5, we take every word from our training and testing data and inject similar words using techniques from [19] to augment the data we use for training and testing. Similar word models are unsupervised models trained on a corpus of text. They can output similar words of a word depending on the position and usage of that particular word with respect to the neighboring words. We show an example of total-domain and cross-domain augmentation using similar word injection model. We use two word injection models: one from the train domain and the other from the test domain. We use augmentation for both the domain in order to transfer some of the context from each domain to another in the form of similar words. Finally, we demonstrate a new state of the art (SOTA) results in cross-domain design mining. We explain the design of our study for augmentation in chapter 4 and discuss the state-of-the-art results in chapter 5.

1.4 Contributions and Thesis Outline

In this work, we contribute the following:

- We provide improved classification results from the study of Brunet et al. [12] using word embedding and stratification, with improved accuracy of 0.94 from the original 0.876. We also introduce a better validation method namely Area Under the Curve (AUC) or balanced accuracy for imbalanced classification study like design mining with AUC of 0.84. Data and replication package: 10.5281/zenodo.3590126 and 10.5281/zenodo.3590123 respectively.

- We also provide a meta-analysis using the vote-counting of previous studies. This will enable future studies to quickly grasp all the information needed from previous studies in design mining. We show a compressed version of the analysis in table 2.1.

- We discuss the characterization of the conclusion stability of NLP models for design mining in §3.3.

- We provide a labeled data set of two hundred and sixty thousand discussions in the form of train, test, and validation data from Stack Overflow. This data set is fully processed with state-of-the-art and modern NLP standards and conventions. We make this available at: 10.5281/zenodo.4010209 .

- We create our new software-specific word vectorizer trained on hundreds of processed and spell corrected literature on software engineering. We make this available as a part of our replication package at 10.5281/zenodo.4010218 .

- We present and discuss the integration of two similar word injector models and show how to achieve total and cross-domain context with them.

- We report on the performance of several machine learning models based on our approach. We discuss the performance improvements of the classifiers and also present the new state-of-the-art results in design mining. We make our code, models, and results available for replication at: 10.5281/zenodo.4010218 .

This thesis begins with a strict replication of the 2014 work of Brunet et al. [12] as a way of explaining the design mining research problem (chapter 3) along with explaining some of the related works in the field(chapter 2). We then extend the replication by examining improved techniques for dealing with the problem, including accounting for class imbalance, in chapter 3, and our attempt to do cross-domain transfer learning. We then start chapter 4 by going into details about the challenges we faced with this transfer learning and then explain how we dealt with the issue of insufficient labeled data, and the need for software-specific context. Our study design is also explained in chapter 4 and the final results in chapter 5. We finish the thesis by characterizing some limitations and study design issues in chapter 6.

Chapter 2

Background and Related Work

This thesis brings together two streams of previous research. First, we highlight works on cross-project prediction and learning in software engineering. Secondly, we discuss previous works in mining design discussions and summarize existing results as an informal meta-analysis. We conclude by looking at the challenges of degrees of freedom in this type of research.

2.1 Cross-Domain Classifiers in Software Engineering

A practically relevant classifier is one that can ingest a text snippet—design discussion—from a previously unseen software design artifact, and label it Design/ Not-Design with high accuracy. Since the classifier is almost certainly trained on a different set of data, the ability to make cross-dataset classifications is vital. Cross-dataset classification [90] is the ability to train a model on one dataset and have it correctly classify other, different datasets. This is most important when we expect to see different data if the model is put into production. It might be less important in a corporate environment where the company has a large set of existing data (source code, for example) that can be used for training.

The challenge is that the underlying feature space and distribution of the new datasets differ from that of the original dataset, and therefore the classifier often performs poorly. For software data, the differences might be in the type of software being built, the size of the project, or how developers report bugs. Herbold [27] conducted a mapping study of cross-project defect prediction which identified such efforts as

strict (no use of other project data in training) or mixed, where it is permissible to mix different project data. We will examine both approaches in this thesis, but in the domain of design mining, not defect prediction. Recent work by Bangash et al. [6] has reported on the importance of time-travel in defect prediction. Time-travel refers to the bias induced in training when using data from the future to predict the past.

To enable cross-domain learning without re-training the underlying models, the field of transfer learning applies machine learning techniques to improve the transfer between feature spaces [58]. Typically this means learning the two feature spaces and creating mapping functions to identify commonalities. There have been several lines of research into transfer learning in software engineering. We summarize a few here. Zimmermann et al. [90] conducted an extensive study of conclusion stability in defect predictors. Their study sought to understand how well a predictor trained with (for example) defect data from one brand of web browser might work on a distinct dataset from a competing web browser. Only 3.4% of cross-project predictions achieved over 75% accuracy, suggesting transfer of defect predictors was difficult.

Following this work, a number of other papers have looked at conclusion stability and transfer learning within the fields of effort estimation and defect prediction. Herbold gives a good summary [27]. Sharma et al [69] have applied transfer learning to the problem of code smell detection. They used deep learning models and showed some success in transferring the classifier between C# and Java. However, they focus on source code mining, and not natural language discussions. Code smells, defect prediction, or effort estimation are quite distinct from our work in design discussion, however, since they tend to deal with numeric data, as opposed to natural language.

Other approaches include the use of bellwethers [44], exemplar datasets that can be used as simple baseline dataset for generating quick predictions. The concept of bellwether for design is intriguing, since elements of software design, such as patterns and tactics, are generalizable to many different contexts.

Transfer learning in natural language processing tasks for software engineering is in its infancy. There is a lot of work in language models for software engineering tasks, but typically focused only on source code. Source code is highly regular and thus one would expect transferability to be less of a problem [29]. Early results from Robbes and Janes [62] reported on using ULMFiT [32] for sentiment analysis with some success. Robbes and Janes emphasized the importance of pre-training the learner on (potentially small) task-specific datasets. We extensively investigate the usefulness

of this approach with respect to design mining. Novielli et al. [56] characterize the ability of sentiment analysis tools to work without access to extensive sets of labeled data across projects, much as we do for design mining.

2.2 Mining Design Discussions

While repository mining of software artifacts has existed for two decades or more, mining repositories for *design-related* information is relatively recent. In 2011 Hindle et al. proposed labeling non-functional requirements in order to track a project's relative focus on particular design-related software qualities, such as maintainability [31]. Hindle later extended that work [30] by seeking to cross-reference commits with design documents at Microsoft. Brunet et al. [12] conducted an empirical study of design discussions, and is the target of our strict replication effort. They pioneered the classification approach to design mining: supervised learning by labeling a corpus of design discussions, then training a machine learning algorithm validated using cross-validation.

Table 2.1: Comparison of recent approaches to design discussion detection. Effectiveness captures the metric the paper reports for classifier effectiveness (accuracy, precision, recall, F1). NB: Naive Bayes; LR: Logistic Regression; DT: Decision Tree; RF: Random Forest; SVM: Support Vector Machine

Study	**Projects Studied**	**Data Size**	**ML Algorithm**	**Effectiveness**	**Prevalence**	**Defn. of Design**	**Defn. of Discussion**
Brunet [12]	77 high importance Github projects	102,122 comments	NB DT	**Acc: 0.86/0.94**	25% of discussions	Design is the process of discussing the structure of the code to organize abstractions and their relationships.	A set of comments on pull requests, commits, or issues
Alkadhi17 [1]	3 teams of undergrads	8,702 chat messages of three development teams	NB SVM + undersampling	**Prec: 0.85**	9% of messages	Rationale captures the reasons behind decisions.	Messages in Atlassian HipChat
Alkadhi18 [2]	3 Github IRC logs	7500 labeled IRC messages	NB SVM	**Prec. 0.79**	25% of subset labeled	Rationale captures the reasons behind decisions.	IRC logs from Mozilla
Zanaty [89]	OpenStack Nova and Neutron	2817 comments from 220 discussions	NB SVM KNN DT	**Prec: 0.66 Recall: 0.78**	9-14	Brunet's [12]	Comments on code review discussions
Shakiba [68]	5 random Github/SF	2000 commits	DT RF NV KNN	**Acc: 0.85**	14% of commits	None.	Commit comments
Motta [53]	KDELibs	42117 commits, 232 arch	Wordbag matching	N/A	0.6% of commits	Arch keywords from survey of experts	Commit comments
Maldonado [16]	10 OSS Projects	62,566 comments	Max Entropy	**F1: 0.403**	4% design debt	Design Debt: comments indicate that there is a problem with the design of the code	Code

Table 2.1 continued from previous page

Study	Projects Studied	Data Size	ML Algorithm	Effectiveness	Prevalence	Defn. of Design	Defn. of Discussion
Viviani18 [85]	Node, Rust, Rails	2378 design-related paragraphs	N/A (qualitative)	N/A	22% of paragraphs	A piece of a discussion relating to a decision about a software system's design that a software development team needs to make	Paragraph, inside a comment in a PR
Viviani19 [83]	Node, Rust, Rails	10,790 paragraphs from 34 pull requests	RF	**AUC 0.87**	10.5% of paragraphs	Same as Viviani18	Same as Viviani18
Arya19 [4]	3 ML libraries	4656 closed issue sentences	RF	F1: 0.69	30% of sentences	"*Solution Discussion* ... in which participants discuss design ideas and implementation details, as well as suggestions, constraints, challenges, and useful references ".	A closed issue thread
Chapter 3	Stack Overflow discussions	51,990 questions and answers	LR/ SVM	**AUC: 0.84**	N/A	A question or answer with the tag "design"	Stack Overflow question/answer
Chapter 4	Stack Overflow discussions and literature	260,000 questions, answers and comment	multiple, including SVM	**AUC: 0.80 with cross data (new state-of-the-art)**	N/A	Stack Overflow questions with tags related to "design"	Stack Overflow question/ answer/ comments
Meta	Open-source	46,470	N/A	**Acc: 0.85-0.94**	15.25%	N/A	N/A

Table 2.1 reviews the different approaches to the problem, and characterize them along the dimensions of how the study defined "design", how prevalent design discussions were, what projects were studied, and overall accuracy for the chosen approaches. We found 12 primary studies that look at design mining, based on a non-systematic literature search. We then conducted a rudimentary vote-counting meta-review [61] to derive some overall estimates for the feasibility of this approach (final row in the table).

Defining Design Discussions—The typical unit of analysis in these design mining studies is the "discussion", i.e., the interactive back-and-forth between project developers, stakeholders, and users. As table 2.1 shows, this varies based on the dataset being studied. A discussion can be code comments, commit comments, IRC or messaging application chats, Github pull request comments, and so on. The challenge is that the nature of the conversation changes based on the medium used; one might reasonably expect different discussions to be conducted over IRC vs a pull request.

Frequency of Design Discussions—Aranda and Venolia [3] pointed out in 2009 that many software artifacts do not contain the entirety of important information for a given research question (in their case, bug reports). Design is, if anything, even less likely to appear in artifacts such as issue trackers, since it operates at a higher level of abstraction. Therefore we report on the average prevalence of design-related information in the studies we consider. On average 15% of a corpus is design-related, but this is highly dependent on the artifact source.

Validation Approaches for Supervised Learning—In table 2.1 column Effectiveness reports on how each study evaluated the performance of the machine learning choices made. These were mostly the typical machine learning measures: accuracy (number of true positives + true negatives divided by the total size of the labeled data), precision and recall (true positives found in all results, proportion of results that were true positives), and F1 measure (harmonic mean of precision and recall). Few studies use more robust analyses such as AUC (area under ROC curve, also known as balanced accuracy, defined as the rate of change). Since we are more interested in design discussions, which are the minority class of the dataset, AUC or balanced accuracy gives a better understanding of the result, because of the unbalanced nature of the dataset.

Verification of the correctness of manual classification is achieved by meeting and agreement among the participants. Validation of automatic classification is measured by evaluating the classifiers with manually labelled small set of data, which is often

referred to as the gold set. Almost all the studies in this field have attempted to produce a best-performing and validated automated classifier [12, 68, 85]. For example, a state of the art result from Viviani et al. [83] talks about a well validated model with Area Under ROC Curve (AUC) of 0.87. However, achieving conclusion stability [6] remains a challenge. Most studies focus on evaluating a classifier on data from a single dataset and discussion artifact. In this thesis we focus on conclusion stability by developing a model with wide applicability to different design discussion which could be used with high accuracy across different projects, and artifact types.

Qualitative Analysis— The qualitative approach to design mining is to conduct what amount to be targeted, qualitative assessments of projects. The datasets are notably smaller, in order to scale to the number of analysts, but the potential information is richer, since a trained eye is cast over the terms. The distinction with supervised labeling is that these studies are often opportunistic, as the analyst follows potentially interesting tangents (e.g., via issue hyperlinks). Ernst and Murphy [20] used this case study approach to analyze how requirements and design were discussed in open-source projects. One follow-up to this work is that of Viviani, [85, 83], papers which focus on rubrics for identifying design discussions. The advantage to the qualitative approach is that it can use more nuance in labeling design discussions at more specific level; the tradeoff of course is such labeling is labour-intensive.

2.3 The Role of Researcher Degrees of Freedom

As Simmons et al. write [72], "In the course of collecting and analyzing data, researchers have many decisions to make ... [yet] it is rare, and sometimes impractical, for researchers to make all these decisions beforehand. Rather, it is common (and accepted practice) for researchers to explore various analytic alternatives, to search for a combination that yields 'statistical significance', and to then report only what 'worked."' They go on to explain this is not malicious, but rather a need to find a 'significant' result, coupled with confusion about how to make these decisions. They introduce the example of dealing with outliers as one such decision. It is unlikely that before the study is started a consistent policy on outlier removal would exist. And yet deciding that something is an outlier is clearly an important data cleaning decision, which possibly impacts the final analysis.

Gelman and Loken [22] elaborate on this concept, which they alternately term

the 'garden of forking paths'[1]. They analyze several studies claiming significant results, and show that with different, equally likely study design choices, different and insignificant results should be expected. In other words, the original study has low conclusion stability. For example, the choice as to what timeframe defines when a woman is at peak fertility is not clear: it could be 6-14 days post-ovulation, or 7-14. Each is based on equally plausible sources. However, under one assumption the work of Beall and Tracy [8] finds a significant result, and under the other assumption, it does not. Which is correct?

Key to note here is that this is not deliberate fishing for significance, or multiple comparisons problems. Instead, the researcher makes a reasonable set of choices, conditional on the data. Gelman and Loken [22] further note that this issue is exacerbated since most often these studies are underpowered, i.e., have noisy measurements and low sample sizes. This is true of the studies we discuss below, as well.

Figure 3.2 outlines just a few of the choices we face in conducting a design mining study. Having formulated our research question, we must then define constructs that (we claim) will help to answer the question. These include

- choice of dataset
- definition of 'design'
- definition and use of stopwords
- whether to oversample
- how to vectorize the discussions
- which machine learning algorithms to use

There has been a recent emphasis on reliability and reproducibility in the software research literature. We discuss a few recent methodological papers that highlight some of the concerns we discuss later in this thesis.

Kitchenham et al.'s seminal guide [39] mentions the tension between exploration and confirmation in empirical research, and the need to beware of multiple comparisons and fishing for results. However, this deliberate fishing is not the issue with which RDOF is concerned. Rather, it is conditioning analysis choices on data: "they [the researchers] start with a somewhat-formed idea in their mind of what comparison

[1] after Borges, https://en.wikipedia.org/wiki/The_Garden_of_Forking_Paths

to perform, and they *refine that idea in light of the data* [22] (emphasis ours)." To be fair, it is only relatively recently that a focus on software specific statistical maturity has begun, with a focus on robustness and reliable results (the preprint from Neto et al. [17], particularly Fig. 7, makes this clear). The most relevant discussions come from reducing the problem of multiple comparisons, and improving the generalizability and replication of software research results (e.g., by ensuring representative samples [55]). In replications, particularly *literal* or *operational* replications as discussed in [24], the focus is traditionally on availability of the artifacts, but a clear understanding of the protocols used, analysis paths not taken, construct definitions, is just as important. In Tantithamthavorn's PhD thesis, the issues around reliability are nicely described in his framework assessing conclusion stability [78]. In particular, he looked at the difference in results based solely on choice of model validation technique. We expand on his work by making all our choices clear, in order to explain where these differences lie.

However, many of these new papers do not discuss analysis and RDOF, e.g. [41], which is a great reference for improving statistical analysis, but does not cover research design reporting. The commendable focus on replication and reproduction of results (e.g. the RoSE festival series, [11], [25]), focus on 'correct' description of the original experiment, which, while critical, is orthogonal to our RDOF concern: that even a correct and fulsomely described protocol may nonetheless have different analysis choices that are equally valid. Shepperd [70] discusses the problems with an overly tight focus on replication at the expense of conclusion validity. Closest to the issue of RDOF is the discussion of *researcher bias* in Jørgensen et al. [34], who report on results showing a full third of their respondents (software researchers) have derived post-hoc hypotheses, i.e., conditioned on the observed data. They define researcher bias as "flexible analyses that lead initially statistically non-significant results to become significant" [34]. This notion of flexibility is key to our use of the term researcher degrees of freedom (RDOF).

Somewhat related is the work on hyperparameter optimization in machine learning tools (e.g., Xia et al. [88]). However, hyperparameter optimization (i.e., automatically searching for optimal analysis approaches) exacerbates the problem, as it removes any decision about the choice from the researcher. Broader questions, such as "why is this the optimal setting" and "will this parameter hold up in other studies" are ignored.

Menzies and Shepperd [47] cast the problem as fundamentally one of *conclusion instability*. Fig. 1 in their paper [47] highlights some of the degrees of freedom that

lead to conclusion instability, including sources of variance (preprocessing choices) and bias (e.g., experimenter). That led to work on the problem with bias and variance in sampling for software engineering studies [42], where Menzies and first author Kocaguneli concluded that perhaps this tradeoff is not as important in software studies[2]. More recently, a preprint by Menzies, Shepperd, and their colleagues [43] highlighted 'bad smells' in analytics papers. We discuss the relationship of bad analytics smells to our work later, when we look at ways forward (Chapter 6).

A related issue to conclusion stability is the concept of researcher degrees of freedom (RDOF). RDOF [22, 21] refers to the multiple, equally probable analysis paths present in any research study, any of which might lead to a *significant* result. Failure to account for researcher degrees of freedom directly impacts conclusion stability and overall practical relevance of the work, as shown in papers such as Di Nucci et al. [57] and Hill et al. [28]. For example, for many decisions in mining studies like this one, there are competing views on when and how they should be used, multiple possible pre-processing choices, and several ways to interpret results. Indeed, the approach we outlined here in figure 3.2 is over-simplified, given the actual number of choices we encountered. Furthermore, the existence of some choices may not be apparent to someone not deeply skilled in these types of studies.

A related concept is the notion of *conclusion stability* from Menzies and Sheppherd [47]. Conclusion stability is the notion that an effect X that is detected in one situation (or dataset) will also appear in other situations. Conclusion stability suggests that the theory that predicts an effect X holds (transfers) to other datasets. In design mining, then, conclusion stability is closely tied to the ability to transfer models to different datasets.

One possible approach is to use toolkits with intelligently tuned parameter choices. Hyper-parameter tuning is one such example of applying machine learning to the problem of machine learning, and research is promising [88]. Clearly one particular analysis path will not apply broadly to *all* software projects. What we should aim for, however, is to outline the edges of where and more importantly, why these differences exist.

[2]from [47], where S is a study accuracy statistic, and $\hat{S}$ is the population (true) statistic: "bias is $S - E(\hat{S})$ where E is the expected value and the variance is $E((S - \hat{S})^2)$"

2.4 Summary

A true meta-analysis [61, 40] of the related work is not feasible in the area of design mining. Conventional meta-analysis is applied on primary studies that conduct experiments in order to support inference to a population, which is not the study logic of the studies considered here. For example, there are no sampling frames or effect size calculations. One approach to assessing whether design mining studies have shown the ability to detect design is with vote-counting ([61]), i.e., count the studies with positive and negative effects past some threshold.

As a form of vote-counting, the last row of Table 2.1 averages the study results to derive estimates. On average, each study targets 29,298 discussions for training, focus mostly on open-source projects, and find design discussions in 14% of the discussions studied. As for effectiveness of the machine learning approaches, here we need to define what an 'effective' ML approach is. For our purposes, we can objectively define this as "outperforms a baseline ZeroR learner". The ZeroR learner labels a discussion with the majority class, which is typically "non-design". In a balanced, two label dataset, the ZeroR learner would therefore achieve accuracy of 50%. In an unbalanced dataset, which is the case for nearly all design mining studies, ZeroR is far more 'effective'. Using our overall average of 14% prevalence, a ZeroR learner would achieve accuracy of $(1 - 0.14) = 0.86$. This is the baseline for accuracy effectiveness. For precision and recall, ZeroR would achieve 0.86 and 1, for an F1 score of 0.93. Comparing this baseline to the studies above, we find that only Brunet and our approach below surpass this baseline. In other words, few studies are able to supersede random, majority-class labeling.

Chapter 3

Design Mining Replication and Extension

In this chapter, we start by explaining our approach to answer **RQ-1**(Is it possible to replicate a previous study and improve that study?) which is to explore if it is possible to replicate an existing design mining study. The first section (section 3.1) describes our approach for replication followed by the results to confirm the successful replication. The replication revealed several shortcomings and thus in section 3.2, we extend the replication to resolve the shortcomings we explored during our study. We also talk about different study designs we faced with including several choices of data processing and sampling techniques, and various vectorization, and classification algorithms during our extension, and also talk about the rationale behind some of the decisions we took along the way. We also discuss our best performing protocol which we call "NewBest" and how we evaluate and compare our classifier to the previous study that we extend with some explanation of the results. After obtaining the "NewBest" classifier, we move on to answering **RQ-2**(To what extent can we transfer classifiers trained on one data set to other data sets?) by testing the model with other datasets in section 3.3. To test the model with other datasets, we introduce a new Stack Overflow dataset and obtain 3 other datasets from the study of [68], [85], and [16] and along with an explanation in section 3.3.

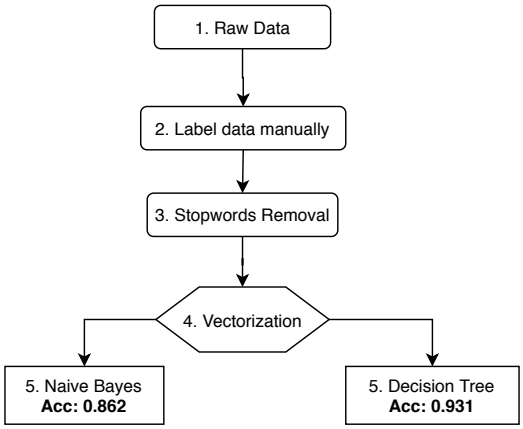


Figure 3.1: Protocol map of the study of Brunet et al. [12] . This shows the pipeline of actions that were taken in that study. This illustrates that the initial raw data were manually labeled before passing through stopwords removal action. After the removal of the stopwords, the data then goes through vectorization and eventually fed into two different classifiers which produced the accuracy values after validating the test data.

3.1 Strict Replication

We now turn to RQ-1(Is it possible to replicate a previous study and improve that study?), replicating the existing design mining studies and exploring the best combination of features for state of the art results. To begin, we conduct a strict replication (after Gómez et al. [24]), a replication with little to no variance from the original study, apart from a change in the experimenters. However, given this is a computational data study, researcher bias is less of a concern than lab or field studies (cf. [77]). The purpose of these strict replications is to explain the current approaches and examine if recent improvements in NLP might improve the state of the art.

To explain the differences in studies, we use protocol maps, a graphical framework for explaining an analysis protocol. This graphical representation is intended to provide a visual device for comprehending the scope of analysis choices in a given study. Figure 3.1 shows a protocol map for strict replication. The enumerated list that follows matches the numbers in the protocol diagram.

1. Brunet's study [12] selected data from 77 Github projects using their discussions found in pull requests and issues.

2. Brunet and his colleagues labeled 1000 of those discussions using a coding guide.

3. Stopwords were removed. They used NLTK stopwords dictionary and self defined stopsets.

4. The data were vectorized, using a combined bigram word feature and using the NLTK BigramCollectionFinder to take top 200 ngrams.

5. Finally, Brunet applied two machine learning approaches, Naive Bayes and Decision Trees. 10-fold cross validation produced the results shown in figure 3.1: mean accuracy of **0.862** for NaiveBayes, and **0.931** for Decision Trees, which also several orders of magnitude slower. However, due to the high accuracy, they took the classification protocol with decision tree to be their preferred classifier.

We followed this protocol strictly. We downloaded the data that Brunet has made available; applied his list of stop words; and then used Decision Trees and NaiveBayes to obtain the same accuracy scores as his paper. The only difference is the use of scikit-learn for the classifiers, instead of NLTK. Doing this allowed us to match the results that the original paper [12] obtained.

After the replication study, we observe several potential omissions. The shortcomings are as follows:

- The replicated study uses 10-fold cross validation to verify the performance of the model. In 10-fold cross validation, the dataset is divided into 10 parts, each part containing an equal amount of data. Then, one part is kept away as test data, and the other 9 parts are used as train data. After the training, that test data is used to test the model. This process is repeated 10 times and each time, a new part is kept as test data. Then an average of the performance is reported as the overall performance of the model. In the replicated study, a dataset of 1000 rows is used where only 24% of the data is reported as design which makes the dataset very imbalanced. According to the rule of 10-fold cross validation, every fold should contain 100 rows of data. We explored the amount of design class and general class in each fold and realize that the ratio of the two classes is not consistent in those folds. For this reason, the actual scenario can not be discovered from using 10-fold cross validation on this kind of imbalanced data without stratification [46]. Stratification distributes the classes among the folds in a way that the ratio of the classes on each fold remains consistent. Hence, we implement stratification on the dataset. After stratification, we run

the experiment as before. We have discovered a significant accuracy drop from 0.931 to 0.876 after using the stratified data.

- 1000 sentences are manually classified in Brunet's dataset [12]. However, only 224 of them are design, which indicates a serious imbalance in the data. Since we are interested in extracting design discussions from overall discussions, the minority class detection is significant to us. The Accuracy rate is used as the evaluation of the classifier in the replicated study. However, accuracy is not a good validation metric in this case because it does not differentiate between the number of the correctly classified example of different classes, i.e., a classifier with an accuracy of 90% with an imbalance ratio(ratio of two class) of 9, is not accurate if it classifies all the test data as negatives.

This thesis attempts to resolve these shortcomings along with improvements in the performance in the following section.

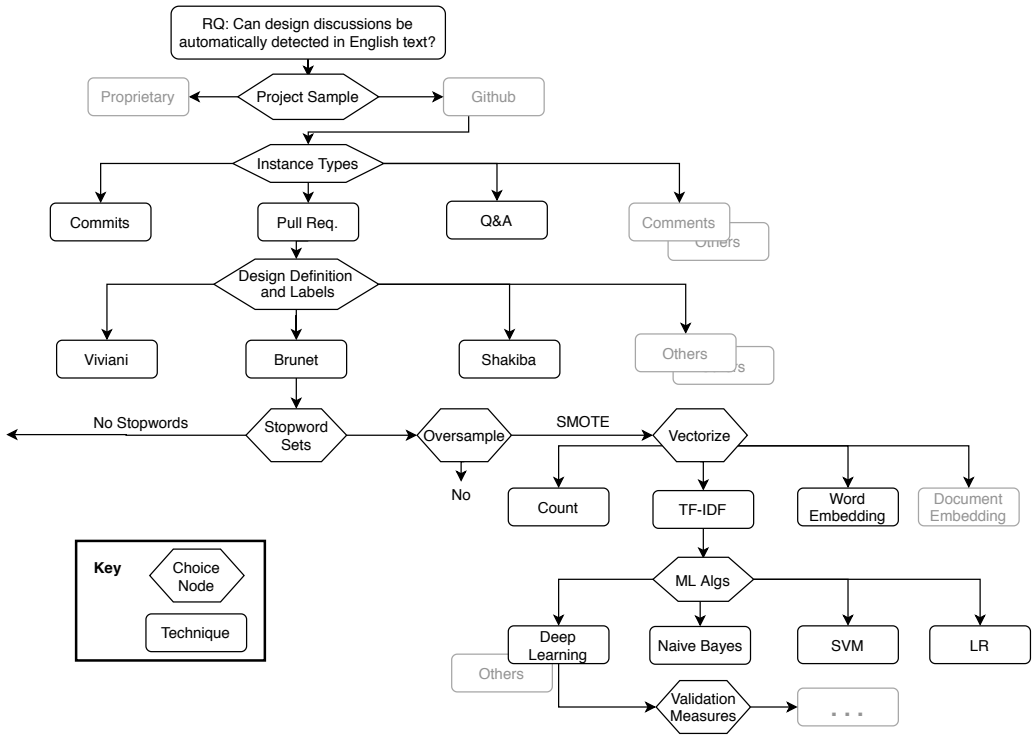


Figure 3.2: Protocol map of possible research paths for design mining studies.

3.2 Extending the Replication

A strict replication is useful to confirm results, which we did, but does not offer much in the way of new insights into the underlying research questions. In this case, we want to understand how to best extract these design discussions from *other* corpora. This should help understand what features are important for our goal of improving conclusion stability.

Shepperd [70] shows that focusing (only) on replication ignores the real goal, namely, to increase confidence in the result. Shepperd's paper focused on the case of null-hypothesis testing, e.g., comparison of means. In the design mining problem, our confidence is based on the validation measures, and we say (as do Brunet and the papers we discussed in §2.2) that we have more confidence in the result of a classifier study if the accuracy (or similar measures of classifier performance) is higher.

However, this is a narrow definition of confidence; ultimately we have a more stable set of conclusions (i.e. that design discussions can be extracted with supervised learning) if we can repeat this study with entirely different datasets. We first discuss how to improve the protocol for replication, and then, in Section 3.3, discuss how this protocol might be applied to other, different datasets.

3.2.1 Approach of the Extension

We extend the previous replication in several directions. Figure 3.2 shows the summary of the extensions, with many branches of the tree omitted for space reasons. One immediate observation is that it is unsurprising conclusion stability is challenging to achieve, given the vast number of analysis choices a researcher could pursue. We found several steps where Brunet's original approach could be improved. These improvements also largely apply to other studies shown in table 2.1. Our approach to the extension is as follows:

- **Imbalance data correction:** We used imbalance correction in order to account for the fact design discussion make up only 14% (average) labels. We took two approaches. One, we stratified folds to keep the ratio of positive and negative data equal. We use SMOTE [13] to correct for imbalanced classes in train data. Recall from table 2.1 that design discussion prevalence is at best 14%. This means that training examples are heavily weighted to non-design instances. As in [1], we correct for this by increasing the ratio of training in-

stances to balance the design and non-design instances. We have oversampled the minority class (i.e., 'design'). Furthermore, we wanted to make our classes randomly distributed to ensure representative folds. To achieve that, we have used 42 as the seed that is used by the random number generator and shuffled the data randomly. We used 5 nearest neighbors to construct the synthetic sample and 10 nearest neighbors to determine if a minority sample is in danger of exceeding the borderline. After the oversampling, the size of our data increase from 1000 to 1508, where the extra 508 data points are generated randomly from the minority classes.

- **More stopwords removal:** We also hypothesized that the software-specific nature of design discussions might mean using non-software training data would not yield good results. Specifically, when it comes to stopword removal, we used our own domain-specific stopword set along with the predefined English stopwords (of scikit-learn). We also searched for other words that may not mean anything significant, such as 'lgtm' ('looks good to me') or 'pinging', which is a way to tag someone to a discussion. These stopwords may vary depending on the project culture and interaction style, so we removed them.

- **Vectorization choices:** Vectorization refers to the way in which the natural language words in a design discussion are represented as numerical vectors, which is necessary for classification algorithms. We present three choices: one, a simple count; two, term-frequency/inverse document frequency (TF-IDF), and three, word embeddings. The first two are relatively common where the last one is gaining popularity among the NLP community.

  Count Vectors—returns the number of occurrences of a specific word in a sentence or a text document. We did not provide an a-priori dictionary and did not use any analyzer since we used domain-specific set of stopwords to filter out insignificant words. The resultant vector is a two-dimensional list containing the sentence and word numbers as the indexes.

  TF-IDF Vectors—is a simple yet incredibly powerful way to judge the category of a sentence by the words it contains. As much as we want to remove the stopwords by developing our domain-specific set of stopwords, it is not possible to predict what stopwords will appear in the future or in the test data. This is where TF-IDF is very handy. It first calculates the number of times a word

Validation criteria	**Equation**	**Insight**
Accuracy	$\frac{tp+tn}{tp+tn+fp+fn}$	Accuracy is the proportion of correct prediction. Suppose, we have an imbalanced data with 95% negative and 5% positive class. If the classifier predicts 100% of the data as negative, the accuracy would still be 95% since it got the 95% of the true negatives right. But, if we are interested in the 5% minority class, we will not get accurate reading using accuracy as the validation criteria.
Balanced accuracy or AUC	$\frac{TPR+TNR}{2}$	Balanced accuracy normalizes true positive and true negative predictions by the number of positive and negative samples. Thus, if we consider the previous example, classifying all as negative gives 50% balanced accuracy score which is equal to the expected value of random guess in balanced dataset.

Table 3.1: Comparison of accuracy and balanced accuracy with proper formulation and insight. Here, p = total number of positive classes, n = total number of negative classes, tp = true positive, tn = true negative, fp = false positive, fn = false negative, TPR = True Positive Rate = $\frac{tp}{p}$, TNR = True Negative Rate = $\frac{tn}{n}$

.

appears in a given sentence which is actually the term frequency. But because some words appear frequently in all the sentences, they become less valuable as a signal to categorize any sentence. So those words are systematically discarded, which is called inverse document frequency. This leaves us with only the frequent and distinctive words as markers. This returns a vector similar to the count vector but becomes very efficient for the vectorization of the test data.

Word embeddings— are vector space representations of word similarity. Our intuition is this model should capture design discussions better than other vectorization approaches. A word embedding is first trained on a corpus. In this study, we consider two vectorization approaches, and one similarity embed-

ding. "Wiki" is a Fasttext embedding produced from training on the Wikipedia database plus news articles [50], and GloVe, trained on web crawling [60]. The embedding is then used to train a classifier like Logistic Regression by passing new discussions to the embedding, and receiving a vector of its spatial representation in return.

As figure 3.2 shows, there are several ways in which vectorization applies. Count and TF-IDF vector performs well to classify sentences based on the content of the sentence (i.e. the frequency of a specific word that occurred). However, word embedding shines in context-based classification since it creates a relation vector of each word based on the position and neighboring words.

- **Performance evaluation:** We switched to use balanced accuracy, or area under the receiver operating characteristic curve (AUC-ROC or AUC), since it is a better predictor of performance in imbalanced datasets[1] [46]. Our choice of balanced accuracy or AUC can also be justified by the comparison between the two validation criteria shown in table 3.1.

3.2.2 Results and Comparisons

Initially, the study we replicate [12] reported their best performing protocol to be Stopwords removal + Bigram features + Decision tree with an accuracy of 0.931 without stratification of the data. We run our extended model(Stopwords removal + Oversample + TF-IDF vectorization + Logistic regression) on the same data and obtain 0.95 in terms of accuracy. However, after stratifying, we have again run the experiment described in [12] and examined that the accuracy dropped significantly from reported 0.93 to around 0.876 where our experiment achieved an accuracy of around 0.94. In both cases, our extended classifier improves the performance in terms of accuracy.

3.2.3 Best Performing Protocol

After applying for these extensions, figure 3.3 shows the final approach. Ultimately, for our best set of choices we were able to obtain an AUC measure of 0.84, comparable to the unbalanced accuracy Brunet reported of 0.931.

[1] defined in the two-label case as the True Positive Rate + the False Positive Rate, divided by two

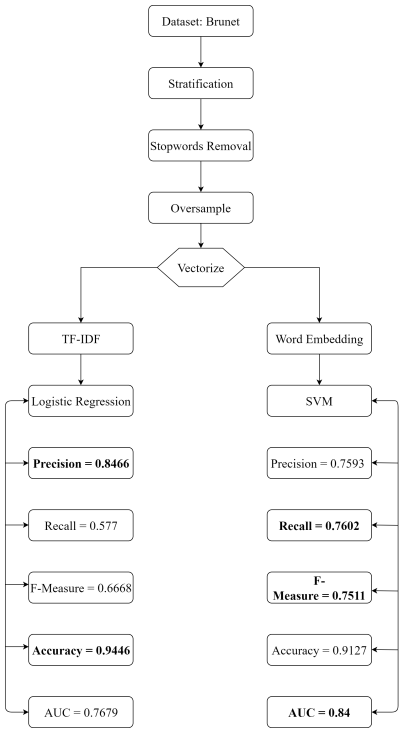


Figure 3.3: Preferred Design Mining Method NewBest. Numbers are the mean of 10-fold cross validation. This figure also represents the pipeline of actions that the dataset goes through. Here we take the dataset from Brunet 2014 and pass it through stratification to ensure an even ratio of the classes in every fold. Then it goes through stopwords removal and after that passes through oversampling to increase the minority class by generating synthetic data. This protocol shows two different vectorizations and classification combination that produces different validation results. The best validation results are made bold for better view.

Logistic Regression with TF-IDF vectorization gives the best results in terms of Precision and Accuracy. On the other hand, Word Embedding with Support Vector Machine provides the best results in terms of Recall, F-Measure, and Balanced Accuracy or AUC. Since we are interested in the 'design' class which is the minority class of the dataset, the highest Recall value should be more acceptable than Precision. As a result, we created a NewBest classifier based on the combination of 'Word Embedding' and 'Support Vector Machine' (right hand of figure 3.3).

Table 3.2: Datasets used for within and cross-dataset classification. All datasets are English-language.

Citation	Dataset	Type	Total instances	Design instances	Projects	Mean Discussion Length (words)	Vocabulary Size (words)
[12]	Brunet 2014	Pull requests	1,000	224	BitCoin, Akka, OpenFramework, Mono, Finagle	16.97	3,215
[68]	Shakiba 2016	Commit messages	2,000	279	Random Github and SourceForge	7.43	4,695
[85]	Viviani 2018	Pull requests	5,062	2,372	Node, Rust, Rails	36.13	24,141
[16]	SATD	Code comments	62,276	2,703	10 Java incl Ant, jEdit, ArgoUML	59.13	49,945
This chapter	Stack Overflow	Stack Overflow questions	51,989	26,989	n/a	114.79	252,565

3.3 Conclusion Stability

In this section, we build on the replication results and enhancements of our **RQ-1**(Is it possible to replicate a previous study and improve that study?). We have a highly accurate classifier, NewBest, that does well *within-dataset* as illustrated from figure 3.5 that shows a self-arrow with 84% of AUC. We now explore its validity when applied to other datasets to understand whether it has conclusion stability to answer **RQ-2**(To what extent can we transfer classifiers trained on one data set to other data sets?).

In [47], Menzies and Shepperd discuss how to ensure conclusion stability. They point out that predictor performance can change dramatically depending on the dataset (as it did in Zimmermann et al. [90]). Menzies and Shepperd specifically analyze prediction studies, but we believe this can be generalized to classification as well. Their recommendations are to a) gather more datasets and b) use train/test sampling (that is, test the tool on different data entirely).

Table 3.3: Sample (raw) design discussions, pre data cleaning.

Dataset	Sample Snippet
StackOverflow	*What software do you use when designing classes and their relationship, or just pen and paper?*
Brunet 2014	*Looks great Patrik Since this is general purpose does it belong in util Or does that introduce an unwanted dependency on dispatch*
SATD	*// TODO: allow user to request the system or no parent*
Viviani 2018	*Switching the default will make all of those tutorials and chunks of code fail with routing errors, and "the RFC says X" doesn't seem like anywhere near a good enough reason to do that.*
Shakiba 2016	*Move saveCallback and loadCallback to RequestProcessor class*

3.3.1 Research Method

In this section, we evaluate our NewBest protocol trained on one dataset to a different dataset, but consisting of the same types of discussions. Before beginning to apply learners to different datasets, it makes sense to ask if this transfer is reasonable. For example, in Zimmermann et al. [90], the specific characteristics of each project were presented in order to explain the intuition behind the transfer. E.g., should a discussion of design in StackOverflow be transferable, that is, considered largely similar to, one from Github pull requests?

In table 3.2 we illustrate each of the datasets considered in this thesis. The 'Dataset' column represents the name of the dataset we allocate to be used further in this thesis. In table 3.3 we show some sample design discussions from each dataset. There are some basic differences between those datasets. For example, Brunet 2014 and Shakiba 2016 consider each sentence of the discussion as a separate data instance while Viviani et al. [85] considered the whole discussion as one data instance. SATD (Self Admitted Technical Debt) is basically code comments while Stack Overflow data is mainly questions. Since the performance of transfer learning is largely based on the similarity between domain (i.e., feature spaces), we would expect to see better AUC results for cross-domain prediction if data sources are the same (e.g. pull requests), projects are the same, and/or the platforms are the same (e.g. Github), demonstrated in figure 3.5. We test the ability to transfer classifiers to new types of discussions

and datasets. We applied the best protocol result from above. That is, the NewBest classifier, using Stopwords removal+Oversampling+Word Embedding+Support Vector Machine. We train this classifier on the Brunet [12] data, and the other 4 datasets described in table 3.2.

3.3.2 Results

Now we talk about the results obtained during our experiments of training our NewBest protocol with one dataset and testing the trained model with the other dataset.

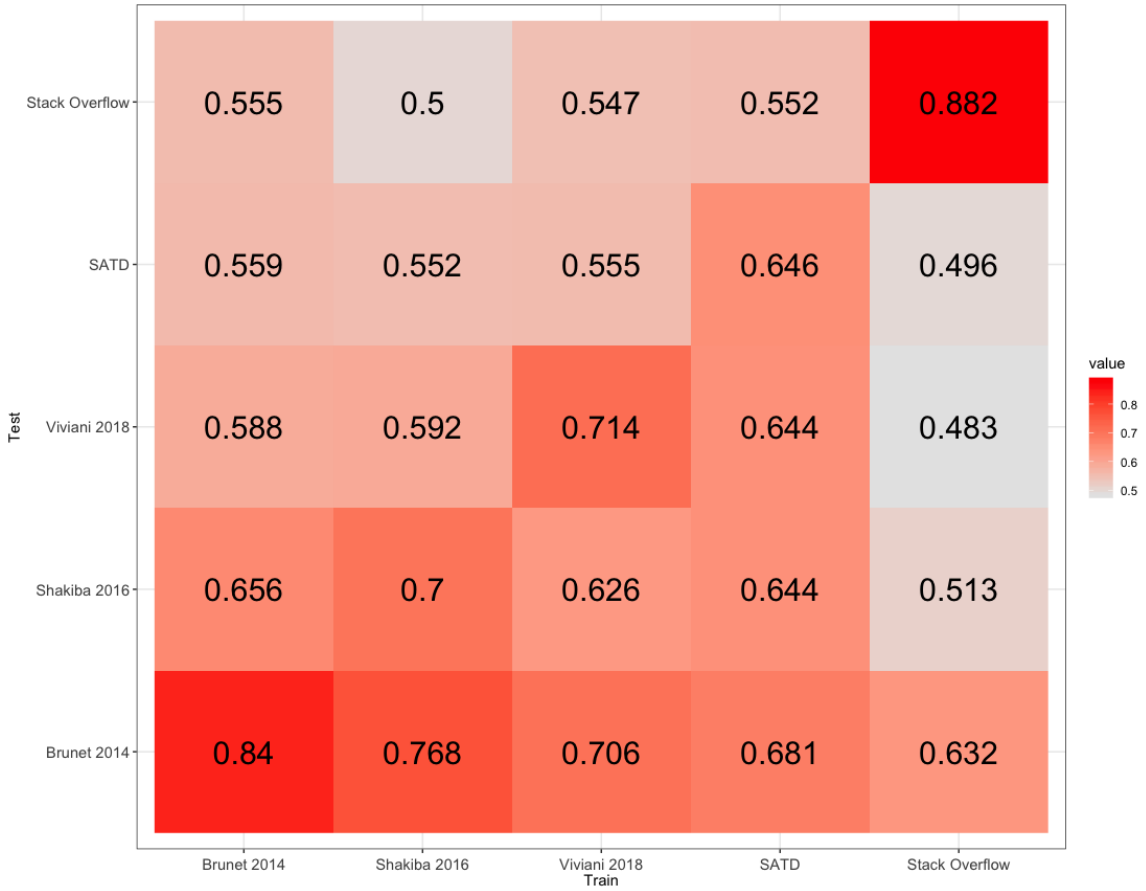


Figure 3.4: Cross-dataset design mining. Numbers: AUC. Read these plots as "the model trained on the Dataset on the X axis has AUC *value* Tested On Dataset on the Y Axis". Higher intensity = better score.

The main challenge for conclusion stability with design mining datasets is that it is hard to normalize natural language text. This means while two datasets might

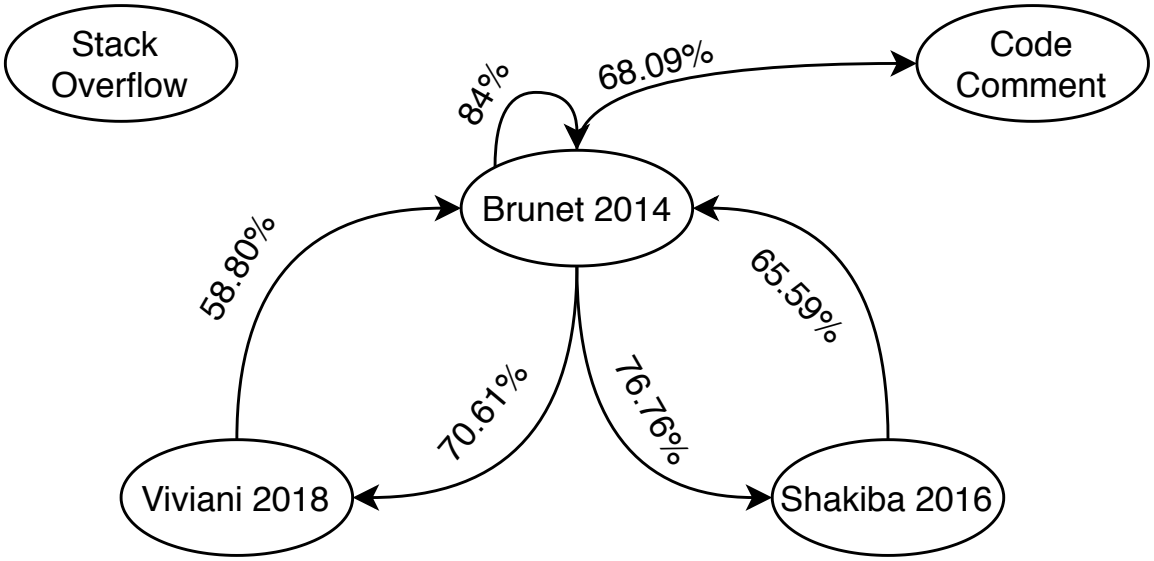


Figure 3.5: Illustration of performance of cross project classification in terms of similarity. Arrow-from means test data and arrow-to represents the train data. An arrow from Brunet 2014 to Shakiba 2016 represents the model tested on Brunet 2014, trained on Shakiba 2016 with AUC value of 76.76%.

reasonably be said to deal with design, one might have chat-like colloquial sentences, while the other has terse, template-driven comments. We illustrate this difference with the example discussions shown in table 3.3. In comparing to other datasets the comparison should still be over reasonably similar 'apples'. As table 3.2 shows, there is some variance in all five datasets, with the type of discussion artifact, source projects, and linguistic characteristics differing. However, despite these differences table 3.3 suggests there should be broad similarities: e.g., concepts such as `Class` or `User`, or ideas like moving functionality to different locations. Intuitively, we suggest the notion of transfer ought to work to some extent on these datasets: they are not completely different.

For the NewBest approach, the diagonal starting bottom-left captures the *within-dataset* performance, which as expected, is better than the cross-dataset AUC scores. Secondly, all models that are tested on Brunet 2014 dataset performed best (bottom row). This is because in building the NewBest classifier, we evaluated our protocol choices against the Brunet dataset. This shows how tightly coupled protocol choices and conclusion stability are.

The effect of dataset type on the performance can also be illustrated from figure 3.5. In this figure, arrow-from means test data and arrow-to represents the train data.

An arrow from Brunet 2014 to Viviani 2018 means model tested on Brunet 2014 data, trained on Viviani 2018 data. As illustrated in the figure, Stack Overflow and Code Comment has the most distant relationship with the other datasets (understandably so because the other three datasets are all from github pull request or issue tracker). It also seems to be the case that results are poorer for datasets that are more removed from each other: using pull requests (Viviani and Brunet) does little better than random for StackOverflow and code comments (SATD).

3.4 Summary

In summary, NewBest or whatever a particular researcher calls to be the best model for a specific task is very subjective. It depends on the task that the user needs to be done. For classifying design discussions, we think that the recall and AUC value is the better choice. We also realized that traditional classifiers work really well within the same dataset. One thing we realized that knowledge transfer in our case is very difficult. Classifiers do not really work well in case of transferring the knowledge. We also learned that similar(not biased) knowledge can be useful for improving contextual information. We also hypothesize that the mixture of extra design context helped a lot to improve the performance. Finally, conclusion stability will be improved by more labeled data. From the results in this chapter, we hypothesize that design mining will be improved if we can provide the context (e.g. can be in the form of words) of one domain to another. These are the two challenges we are going to explore in the next chapter (chapter 4).

Chapter 4

Improving Cross Domain Design Mining with Context Transfer

4.1 Introduction

In this chapter, we propose our idea and implementation to answer the rest of the research questions (RQ-3, 4, and 5). We start this chapter with the challenges we have faced while conducting our design mining study in chapter 3 in §4.2. Then we introduce our solutions to those problems. First, we briefly describe our data collection technique in §4.3.1 to answer RQ-3(How can we get more labeled data to train, validate, and test models of design mining?) of getting more data for training, testing, and validation purpose. We introduce two different types of data for our two types of models. The first dataset we collect is a corpus of texts from software engineering literature. We scrape text data from 300 software engineering related books, conference, and journal papers to build this corpus. We use this data to train our word vectorizer model (We talk more about this later this section). The second dataset has 260,000 texts that are labeled as either 'design' or 'general'. We collect this dataset by scraping Stack Overflow questions, answers, and comments, and label them according to the tag of that particular post. We also explain the combination of steps for our data processor to process the data. We also discuss the validation of the datasets and the suitable usage of the words (which features to choose) for our particular needs. §4.3.2 explains our proposal to resolve the transferability issue of a model trained on one dataset to another. We start this by describing the steps to build software specific word vectorizer which tackles the RQ-4(How useful are

software-specific word vectorizers?). We use the corpus scraped from 300 software engineering related books, conference, and journal papers to train the unsupervised model that can be used as a reference model while converting the text to vector. After that, we discuss augmenting our data by injecting similar words with the help of the similar word injector model which answers the RQ-5(How to provide domain context to a small sample of data?) of providing domain context. We realize that a subset (of data) of a domain is not enough to represent the whole domain (i.e. Stack Overflow). We name this as the lack of total-domain context. We also understand that dataset from one domain (eg. Stack Overflow) can be different (in terms of sentence structure and words) from other domain (eg. Github). We define this issue to be the problem with providing cross-domain context. Hence, in §4.3.2.3, we demonstrate how similar word injectors can be used to provide total-domain context as well as cross-domain context. Finally, we finish this chapter by illustrating the high-level design of our proposed architechture (figure 4.6) with a brief explanation in §4.4. We only explain our methods in this chapter and then, in chapter 5, we discuss on the results we obtained from our experiments.

4.2 Challenges with Cross-Dataset Classification in Design Mining

In chapter 3 we first replicated an existing design mining study. We then introduced an improved design mining classifier that modified previous efforts by adding some modifications to the algorithm and improving the ways of validating the classifier by implementing appropriate validation criteria to account for the imbalanced nature of the data. Our underlying goals were aimed at improving the conclusion stability of a given model, namely, the model's performance on out of sample data, which might be from different projects (cross-project) or across domain (different domains, such as Github issues, pull requests, source code comments, or Stack Overflow discussions).

For cross-dataset and cross-domain classification, our efforts performed poorly as illustrated in chapter 3 section 3.3. Our classifier trained on one set of data does not perform particularly well in classifying discussions of other domains, such as Stack Overflow questions. Even though the preliminary results of our NewBest classifier was better than the previous study, it did not do well in classifying design discussions from cross-domain dataset. As a result, even though we were considering our problem to be

related to typical NLP (Natural Language Processing), models trained with reference to natural literature, such as Wikipedia, often failed to understand the context of software design. Thus our results are not surprising, since lack of domain specificity is a well-known limitation of such language models [19]. We observed the following challenges:

1. As with all machine learning efforts, there is a linear increment in the learning with the amount of vocabulary. Our experiment suffered from a relative lack of data. Thus, it is necessary to find *scalable ways of obtaining labeled design data*.

2. Machine learning models trained on one domain get biased to that input format. For example, the way in which one writes an issue comment is different than how one writes a code comment (above and beyond the syntactical issues). Thus we also concluded that we need to *provide the classifier with the context of both domains* (issues, code comments, etc.) in order to transfer the learning from one domain to another.

3. NLP models perform well with general discussions, however, they *fail to understand the context of software specific vocabularies*. For example, the word "class" has a completely different context in software engineering compared to general-purpose literature.

We, therefore, focus the remainder of this thesis on dealing with these challenges in the context of design mining.

4.3 Solutions to the Challenges

4.3.1 Getting More Labeled Data

Most design mining studies e.g., [12, 68, 84], build a training dataset using human labelers, where two or more humans first label the data and then discuss among themselves to come to an agreement about the label (i.e., design/not-design). While we think this is a very effective method of building the dataset, the limitation on the amount of data a human can label is a problem that was repeatedly reported in the studies mentioned before. This was also one of the conclusions we made in the previous chapter (chapter 3). In this section, we talk about our proposed

idea about acquiring, processing, and validating data that exist in different developer communication platforms. We are mainly building and leveraging two kinds of dataset in our study: one for our word embedding model and the other for the classifier model.

4.3.1.1 Datasets

Here we talk briefly about the two kinds of datasets we talked about previously. Our aim is to build two different models for two very different purposes. First, we talk about the dataset we created to train our word2vec model that acts as a reference model for our data to be converted to vectors from words. Recall step 3.2.1 of our previous approach in chapter 3 where we talk about word embedding as one of the vectorization choices. Previously, we used two general-purpose reference models trained on Wikipedia and Twitter data. In this section, we build our own data to train our reference model and we name the dataset as 'Dataset for Word Embedding'. Details on why and how we decide to create that dataset is described in item 1. The second dataset we talk about is for training our classifier model. We explain the steps to create that dataset in dataset item 2.

1. **Dataset for Word Embedding**— The lack of semantics can harm the performance of a word embedding model [86]. Word embeddings are distributed word representations based on neural networks. While traditional one-hot algorithm represents a word with a large vector, word embeddings embed every word into a low dimensional continuous space taking the semantic and syntactic information into account [45].

   Hence, our word embedding model will be most benefited by structured sentences with grammatical correctness providing semantic relationship information between key terms [33]. The software engineering literature represents more structured sentences in terms of semantics than comments or discussion thread. In our effort to get structured textual data, we scrape the plain text from 300 books, conferences, and journal papers randomly that are related to software engineering. We remove the individual title, figures, names, tables, and all the meta-data to preserve the copyright policies of the literature. Our specific processing approach is discussed later in §4.3.1.2. After all the processing, our dataset yields 1,575,439 total words with 20,607 unique words. Table 4.1 shows an example snippet of the data for the word embedding model. The dataset does not make much sense by just looking at it. This is because of

ultimately reusable design patterns identify sets common recurring requirements define pattern object interactions meet requirements however patterns directly applicable programming design patterns focus problems faced large programs networks challenges quite different challenges include section robustness once deployed sensor network unattended months years resource usage sensor network nodes colloquially known motes little batteries diverse service implementations applications able choose tween multiple implementations routing hardware evolution mote hardware constant evolution applications system services portable across hardware generations adaptability application requirements applications different requirements terms lifetime communication sensing implementation designed challenges mind language event based execution model components similarities objects state interact interfaces significant differences inheritance dynamic dispatch object components interactions fixed rather runtime promotes reliability efficiency programmers cannot easily apply idioms patterns languages when results rarely effective paper present preliminary eight design patterns show used build components address patterns based

Table 4.1: Snippet of the Dataset for Word Embedding. The relation between the words in terms of the position and neighboring words (i.e. the occurrence of design pattern in multiple places) can be illustrated by the table.

the removal of all the punctuation, stop-words, lemmatization, etc. However, a closer look at the snippet would allow us to see the relation between the words in terms of the position and neighboring words (i.e. the occurrence of design pattern in multiple places). We make this dataset and the following datasets available in our replication package at doi:10.5281/zenodo.4010208 and doi:10.5281/zenodo.4010217.

2. **Dataset for Classifier**— We scrape text from Stack Overflow questions, answers, and comments to use them as our training, testing, and validation data for our classifier. For labeling the data, we use the user-assigned tags of the questions and label these tags as design or general depending on the tags of the question. Bazelli et al. [7] showed that a tag acts as a label that can be used to describe the contents of the questions. Their study found that the tags can be at times "misleading" because they can be assigned both by the author of the questions as well as other users. However, they mostly represent actual information about the content of the question because of the moderation by designated moderators and removal of unused tags after a certain period.

   If we find one of the following: "design-patterns", "software-design", "class-design", "language-design", "design-principles", "system-design", "code-design", "api-design", "dependency-injection", "architecture" tags, we label the entire discussion as "design" while labeling "general" otherwise. Table 4.2 shows some output examples based on the code-book we defined, after text processing described in §4.3.1.2. The data distribution in table 4.3 illustrates that we have achieved a large number of data for our training, validation, and testing phase with an equal amount of instances for each class, preventing the problem of class imbalance for this type of data reported in [68]. This even distribution of the dataset also removes some of the steps we took in chapter 3 in our attempt to remove the unbalanced nature of the data such as stratification [66] and oversampling [13]. We use data from the study of Brunet et al. [12] as our test data sample and refer to this as Brunet2014 data for the rest of this thesis. Brunet's data was initially extracted from Github where our data is from Stack Overflow, which fulfills our criteria of being a cross-domain dataset.

Text	Tags	Label
headless device local network trying headless raspberry connect local network want automated though mobile flutter given mobile network raspberry connected flutter will connected firebase	python, flutter, networking, dart, raspberry-pi	general
difference interface design pattern hard time knowing when something interface design pattern example observer	design-patterns, model-view-controller, interface, observer-pattern	design
soap request explain doing something platform called section called repeaters send soap request specific address will honest idea soap something question receive soap data want receive soap request know save works structure guys give information send soap imagine tried investigate truth know soap works using code seem work hope help thanks	php, xml, web-services, soap	general
behind naming visitor pattern book design pattern says visitor pattern visitor lets define changing classes elements read pattern book failing understand intuition behind naming pattern visitor called visitor	design-patterns, visitor	design

Table 4.2: Example of labeling Stack Overflow discussions based on tags.

Type	Total	# Design data	# General data
Train	2,00,000	1,00,000	1,00,000
Validation	30,000	15,000	15,000
Test	30,000	15,000	15,000

Table 4.3: Data Distribution for the Classifier

4.3.1.2 Data Processing

Hemalatha et al. [26] showed that data processing helps to remove noisy and inconsistent data resulting in improved performance, inspiring us to take a pipeline of different text processing techniques to process the data. Raw text from comments and discussions from the web pages often contains unnecessary tags, punctuation, white spaces, new lines, and numeric elements. At the beginning of the processing, we look for these and strip them from the text. We implement our spell correction algorithm as the next step of the process. We take the Britain English dictionary with the affix as our primary dictionary and add Australian, Canadian, American, and South African English vocabulary into it to make it more versatile. Then we take every misspelled word and compare it with the dictionary to find out the most relevant word. We take up to five relevant words and then analyzing them based on the context, number of similar letters present, and the structure. Then we replace the misspelled word with the correct word or a list of possible correctly spelled words.

Ghag et al. [23] show that stopword removal can significantly improve the performance of the traditional models, however, it does not do much good to the more sophisticated deep learning-based models. Since we are trying to make our dataset general and independent of a particular classifier, we take the stopword removal as our next step of processing the text. This step removes all the insignificant and unwanted words from the data such as “the”, “and” etc. which in turn, can hamper the performance by misleading the classifier. We also make sure to keep our word length between 3 and 25 because any word less than 3 letters or more than 25 letters is not relevant to our study. We only considered English vocabularies that does not exceed 25 characters in length. At the final step of our processing, we implement a normalization approach to normalize the words to their root/base form. For example, ‘running’ has a base word of ‘run’. We choose lemmatization over stemming because Balakrishnan et al., [5] shows some versatile experiments where lemmatization outperforms

stemming in terms of precision. A base vocabulary is used to perform lemmatization which eliminates the inflectional endings and reduces the word to its base form. We use a lexical database called WordNet [52] as our base vocabulary. Combining all the steps in a pipeline allows us to achieve a state-of-the-art text processor for our study.

4.3.1.3 Data Validation

Section 4.3.1.1 item 2 discussed how we acquire the data and label them with the help of user-created tags. However, tags can be misleading too. Often, tags are created by the author, and the concept of design being very subjective, labeling data with the help of tags can create doubt on the validity of the data. In this section, we examine the validity of the data with the help of top words and top phrase analysis.

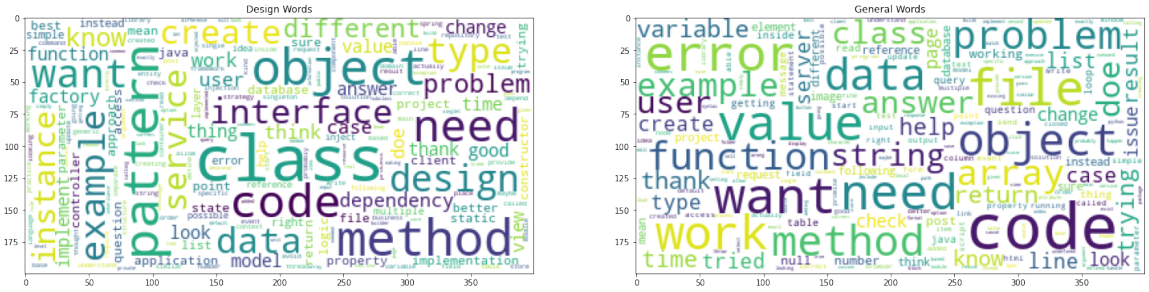
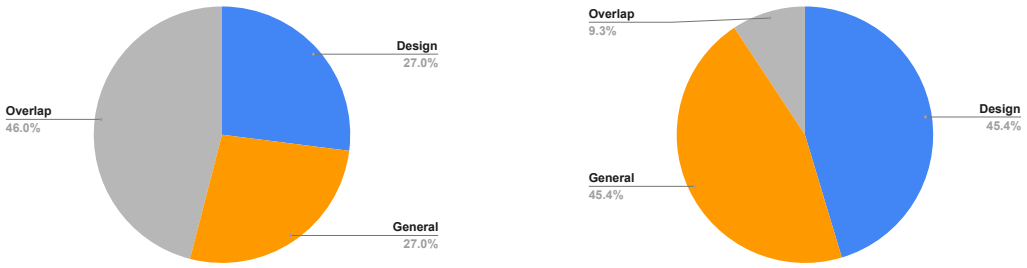


Figure 4.1: Wordcloud of design and general class. This shows that, it is possible that only taking one word to be a member of a class can be deceiving since one word can be representative member of both class, i. e. the words code, method occurs in both of the classes.

The presence of a particular word can contribute a lot in classifying the whole text. For example, if a certain text contains keywords like 'design' or 'pattern' along with some other related words, the text can be easily classified into the design class. However, relying on some specific words and classifying a long text based on some discrete keywords often can be misleading as illustrated in figure 4.1. We first analyze the top 100 words in each class (design/not-design) and analyze the overlap between the words. Overlap means that a particular word is present in texts that relate to both of the classes and it falls into one of their top 100 words. This overlap implies that word would be a poor indicator of class membership.

Figure 4.2a shows a 46% overlap of top words in the design and general class. This figure also suggests that relying on a particular set of discrete keywords can often lead to misclassification since both classes contain similar kinds of keywords.



(a) Overlap in top words (b) Overlap in top phrases

Figure 4.2: The percentage of overlap in the top 100 words and the top 100 tri-gram phrases. This illustrates that taking phrases instead of word can be unique for each class thus shows the reduction of overlapping from (a) to (b).

To deal with this overlap, we instead rely on a tri-gram model. A tri-gram model contains three words each time. This way, the middle word can have the context of both the neighbouring words surrounding it. Figure 4.2b illustrates a significant reduction in the amount of overlap between the top 100 tri-gram phrases. This way, the classes can have features that are very unique to that particular class. Although it is possible to reduce the overlap by increasing the size of the phrases, we opted for using a tri-gram model because of its proven capability of being the best predictor for probabilistic measures [79].

4.3.2 Resolving Potential Transferability Issues

Our final two challenges referred to building software context into the model, both for context based on data domain (issue/comment/pull request) and based on software semantics, rather than general language models, which are typically trained on Wikipedia or other general-purpose sources. To resolve these problems, we introduce software specific word vectors and data augmentation.

4.3.2.1 Software Specific Word Vectorizer

Since we are using an embedding model to turn our text data into a numeric vector, it is important that our word embedding model has the context of software engineering since we are dealing with texts that are mostly about software. In our previous study, we used Glove word embedding [60] which is trained on Wikipedia data. This word embedding works well for general-purpose text classification, however, does not perform well with text from the software domain. For example, the word ‘class’ has

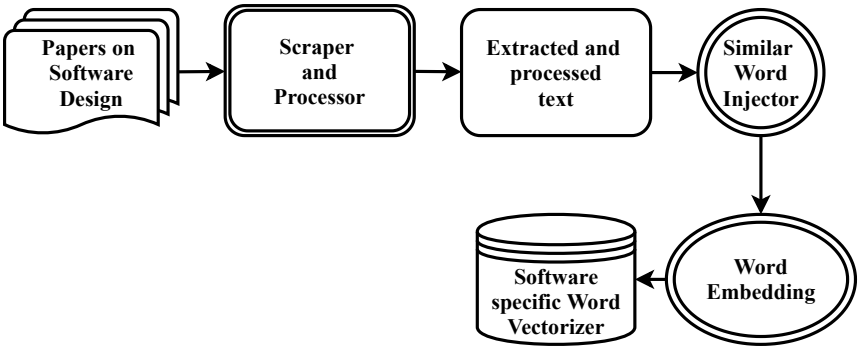


Figure 4.3: Training the Word Vectorizer model. First, we scrape plain text from the software engineering related books, journal, and conference paper. Then, we use similar word injector model to inject similar words into the corpus. Then we use unsupervised word2vec to train the model on the corpus of texts.

a general contextual relation with education while in software engineering, the word 'class' is an integral vocabulary of object-oriented design patterns. To address this issue, we collect our data for training the word embedding model from literature related to software engineering.

Joulin et al. [36] described the use of subword information to enrich a word vector. They also used a similar technique to implement a compression algorithm [35] for classification models. We implement the algorithm by Bojanowski et al., [10] with the help of fasttext[1] with the steps illustrated in figure 4.3. The literature corpus described in §4.3.1.1 item 1 is used to train our new word embedding model.

First, the dataset is passed through the processing steps described in §4.3.1.2. Then, each word of the corpus is evaluated and injected with similar words using the ECS similar word injector. During the training phase, we train the classifier unsupervised since we just want to group the data according to the similarity. We have used skip-gram [49] as one study [51] shows that skip-gram models work better with subword information than cbow [49]. We take words with length from 4-20. Since we are removing every word with less than three characters in our text processing step, it is not important to take the words less than 4 characters. In addition, design words seem to be on the longer side, i. e. 'reproducibility' contains 16 characters. We are considering characters up to 25 characters in length. We take 300 dimensions of each word training by looping for 10 epochs. Both of our decisions of taking 300 dimensions and looping for 10 epochs is taken because the training corpus is relatively

[1]https://fasttext.cc/

small.

4.3.2.2 Data Augmentation Using Similar Word Injection

Machine learning models typically require a large amount of data to train, test, and validate. All the previous studies that we explored in table 2.1 either reported manual labeling or synthetic generation of the data. We used a relatively large amount of data containing 260,000 rows of discussion from Stack Overflow for this study, yet, this is only a very small subset of the total discussions in Stack Overflow.

To augment the data we collected, we use an unsupervised word embedding from the study of Efstathiou, Chatzilenas, and Spinellis [19] (which we call ECS) to inject similar words into the small subset of data we can have. This unsupervised model, as the name suggests, only needs data but does not require any labeling. On the other hand, the similar word injection into the small subset of data enables the data to have all the possible vocabulary from the domain with the surrounding context. We call this approach "providing total domain context". Figure 4.4 shows an example: the word 'design' is augmented with, among others, 'redesign' and 'architecture'.

The ECS word vector was trained on 1.5 Gb of text obtained from Stack Overflow. Our injection algorithm first splits the whole corpus into individual words. Then, each word is passed through the model to obtain a set of similar words for that specific word. The Efstathiou model outputs a similarity index from 0 to 1 of each word in the set of similar words. We take those words which have a similarity index more than or equal to 0.6, after experimenting with different cutoff numbers to find an optimal AUC result. Then, we concatenate the set of similar word with the actual word to obtain the corpus of similar word injected words.

4.3.2.3 Providing and Transferring Context

In the previous chapter (chapter 3) we found that both the lack of domain context in a small subset of the total data and making the train data and test data similar in terms of vocabulary, are a challenge. The out-of-vocabulary problem is one such instance since word vectors are not helpful if they do not contain a particular word (such as a unique source code identifier) that is out of its vocabulary.

Appropriate classification of text requires the training data to have proper understanding of the whole domain, e.g., of projects and artifact types. Ideally, the test and train data also have a similar context. For example, training on sentences from

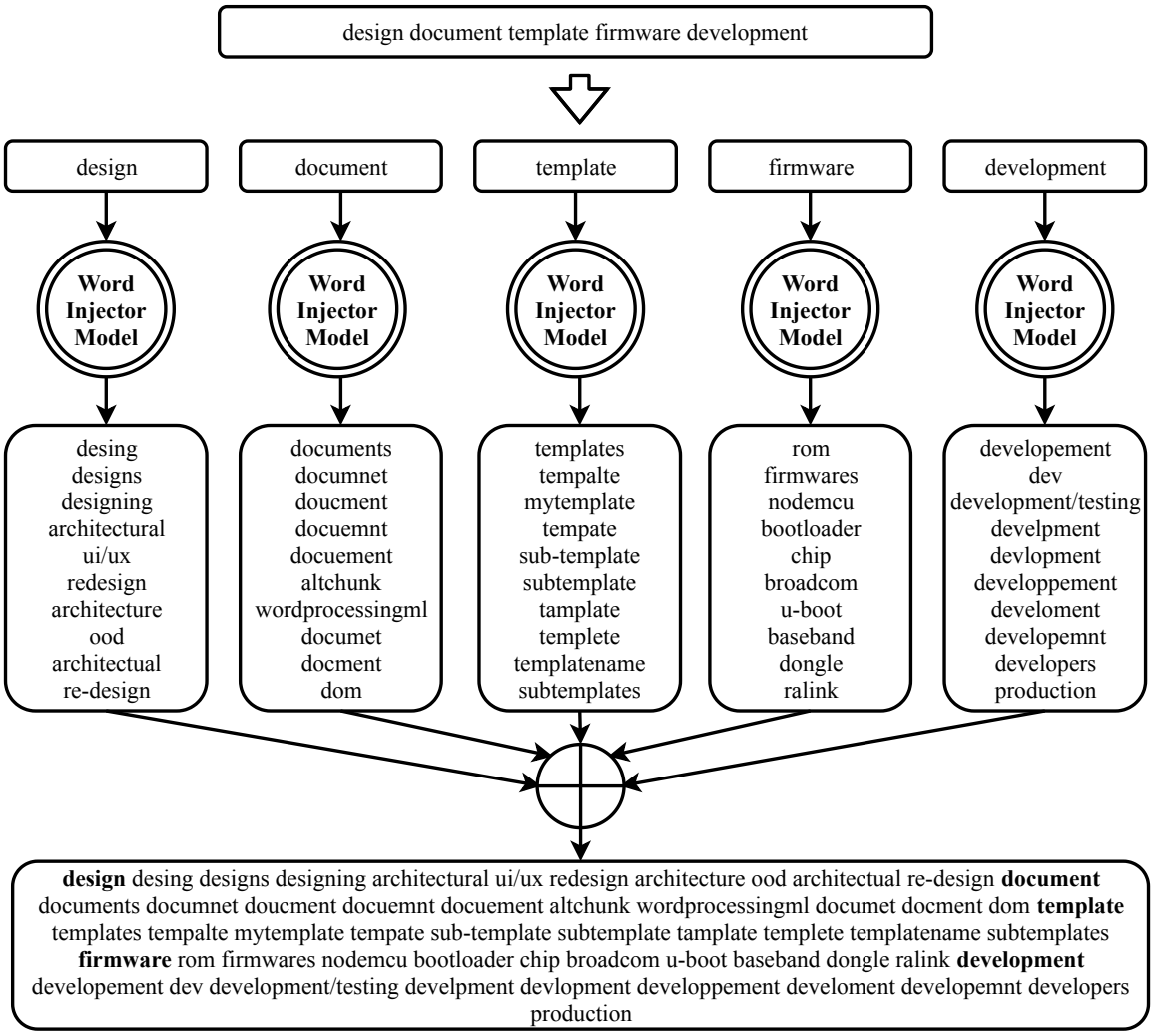


Figure 4.4: Similar word injection workflow. First, every word is split from the input text. Then we iterate through every word to find similar words based on the similarity index. Then we merge all the words along with the similar words to get the augmented text for the input text.

pull requests from Project A will have poor results if applied to code comments from Project B. Nonetheless, we expect a design classifier to be able to understand (as humans would) that both are valid, and frequent, ways in which design discussions happen.

Vocabulary from one domain (ex. Stack Overflow) can differ from another communication medium (e.g., Github, email). For example, most of the discussion happening in Stack Overflow relates to questions and answers, while Github represents mostly statements in the form of issue tracking and pull requests. Hence, the vocabulary and the context also vary from one domain to another.

We create an unsupervised word vector trained on data with a similar context to the test domain (in this case, data from Github) to inject similar words into the training data (Stack Overflow data). We also repeat this step in the opposite direction (reusing the already trained ESC similar word injector model) to inject the training domain's vocabulary context to the test data. We name this step "cross-domain context transfer". While doing this, we must be careful not to bias the training data too much with information from the test data set. The Github word vector is trained on a random selection of pull request text obtained from the Github BigQuery data dump. However, it is possible that some contamination from the Brunet2014 dataset appears in our random selection, and it is possible that some of the 260,000 sentences we use from Stack Overflow also appear in the ECS word vector. However, we judged this risk to be minimal, given the data volumes involved. We outline the context transfer approach in figure 4.5.

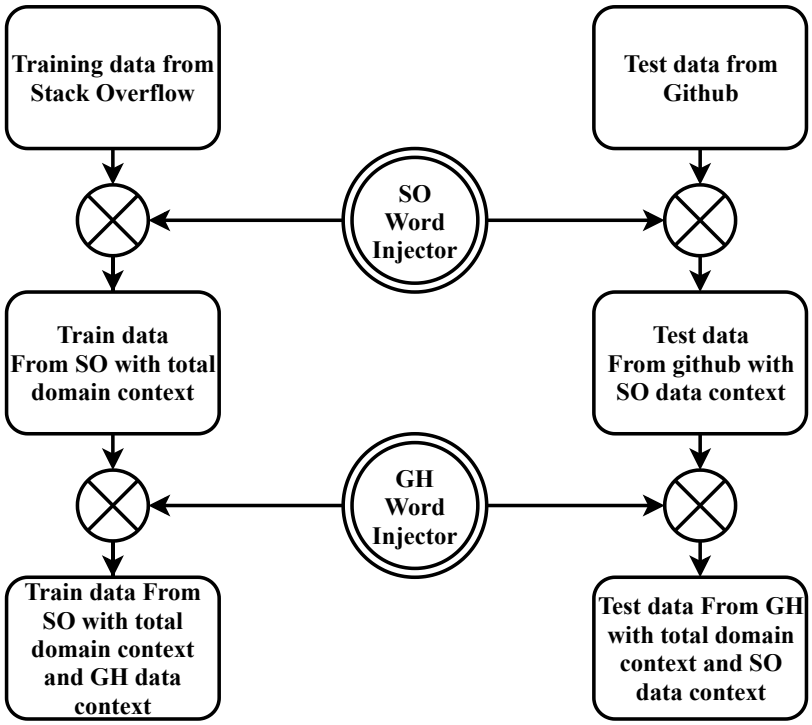


Figure 4.5: The Proposed idea of providing total and cross-domain context. First, the 'SO Word Injector' is used to provide total domain-context to the Stack Overflow data and Stack Overflow domain-context to the Github data. Similarly, the 'GH(Github) Word Injector' is used to provide Github domain-context to the Stack Overflow data and total domain-context to the Github data.

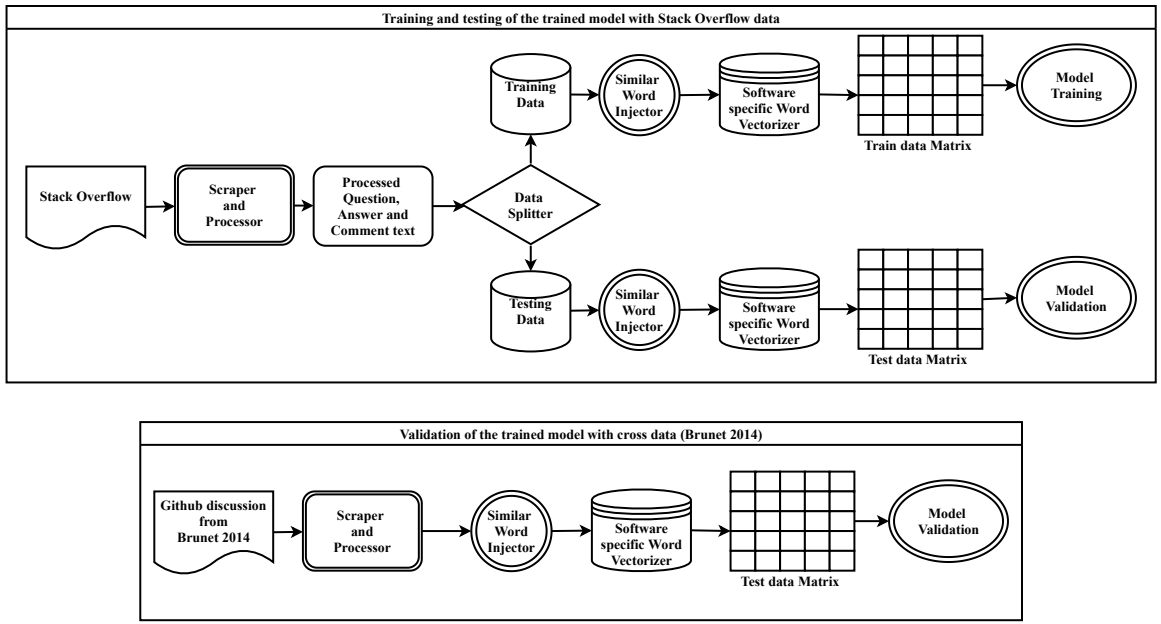


Figure 4.6: High-level design of the study. The first box illustrates our approach to validate our model with the test data (standard validation approach). The second box shows cross-domain validation of our model with Brunet 2014 data.

4.4 Study Design

In this section, we describe the overall design of our study along with the proposed design of each component of our system architecture. Figure 4.6 illustrates an overall and high-level view of our study with all the components. We have talked about the source of the data and how to acquire and process the data in §4.3.1. After processing, the data goes through a data splitter that splits the dataset into train and test data. 200,000 data is kept as train data where 30,000 data is kept for validation and 30,000 for test purposes. Then the data goes through augmentation described in §4.3.1. The augmented data then goes through our 'Software Specific Word Vectorizer' model to be converted into vector from words. Similarly, the classes are also expressed as a form of a vector. We refer the matrix and vector of train data and train class respectively as train vectors. Similarly, we name the combination of the text matrix and its associated vector test vector[2].

We implemented 10 classifiers namely: 'Nearest Neighbors', 'Decision Tree', 'Random Forest', 'Logistic Regression', 'Gaussian Naive Bayes', 'Neural Net', 'AdaBoost', 'QDA', 'Linear SVM', 'RBF SVM', and analyzed the output of the model with every transformation of the data, using Scikit-learn [59]. These 10 classifiers represent a good variety of probability, regression and neural-network based approach. We limit the scope of deep learning model to simple neural network rather than going for a more sophisticate deep learning approach like LSTM.

We have used a Google Cloud Platform[3] instance with a processor of Intel's E2 platform and 16 GB memory to load four models (1-word vectorizer, 2-word injectors, 1-classifier) at once.

We use our validation data in training the neural networks to validate the results after each iteration. Our test data set is kept completely separate from the training phase. Hence, our test data can be considered to be an unknown set of data from the same domain. After the completion of the training, we evaluate our models with the test data.

The bottom box in figure 4.6 represents our cross-domain validation approach. This approach is very similar to the approach of validating with the test data. The only difference is, we use Brunet 2014 data as the test data in this case. We do not use 10-fold cross validation (like chapter 3) since we are testing the whole data at

[2]for access, see doi:10.5281/zenodo.4010208 and doi:10.5281/zenodo.4010217

[3]https://cloud.google.com/

once. Hence, there is no need to use stratification on the data.

4.5 Summary

In summary, we start this chapter with discussions on the challenges of design mining study based on our replication, extension, and conclusion stability study in chapter 3. Then we mainly focus on the problem of lack of data along with imbalanced characteristics while training a classifier and difficulties in transferring the context while classifying cross-domain dataset. Unlike previous approaches in table 2.1, we propose to collect the data from a publicly available discussion forum with state-of-the-art data processing. To provide context, we take two separate approaches. First, we propose to create a software-specific word vectorizer trained on data from the literature in the software engineering domain. This way, we hope to capture software-specific contexts in texts. Secondly, we propose to include a domain-specific similar word injector to augment the data we are using for training and test purpose to the pipeline of the classification. Finally, we apply the data with 10 different classification models to experiment with the performance improvements. In the next chapter (chapter 5), we illustrate our step-by-step results and discuss our understanding and interpretation of the results.

Chapter 5

Results, Analysis, and Comparisons

5.1 Introduction

In this chapter, we describe and analyze the results of every step we have taken for context transfer in chapter 4. First, we talk a little bit about our experiments followed by results from using software specific word vectorizers in §5.3. Then we discuss our result of similar word injection. First, we explain the results we obtain by injecting similar words from the Stack Overflow word injection model. Then we discuss the results obtained from using data augmentation in §5.4. We have also illustrated our within dataset results in the last part of the section which shows our improved performance compared to the most recent study on design discussion mining in [83].

5.2 Experiment

Our main goal of this study is not simply to show a new state of the art result outperforming a previously studied classifier when classifying data from the *same* domain. Rather, we want to illustrate how the data can be generalized and fed to the model so that the model can perform better than the previous studies in terms of detecting design discussions from an unknown domain. We used 10 modern classifiers to demonstrate the performance of each classifier individually. We report on the AUC score of all classifiers in the results and charts below. We conduct our experiment in several stages. We first look at how using software word vectors improves the results.

Next, we consider data augmentation using similar words. Results reflect the AUC score of a classifier trained on the 200,000 rows of Stack Overflow discussions and the Brunet2014 data (Github pull requests) as testing data.

5.3 Software Specific Word Vector

First, we want to make sure that our hypothesis of domain-specific word vectorizer performs better than the conventional word embedding model to answer RQ-4(How useful are software-specific word vectorizers?). For this purpose, we compare the performance of our software literature word vectorizer with a state-of-the-art word vectorizer, Glove [60]. Except for using two different word vectorizers, all the other constraints were kept the same during the two test runs.

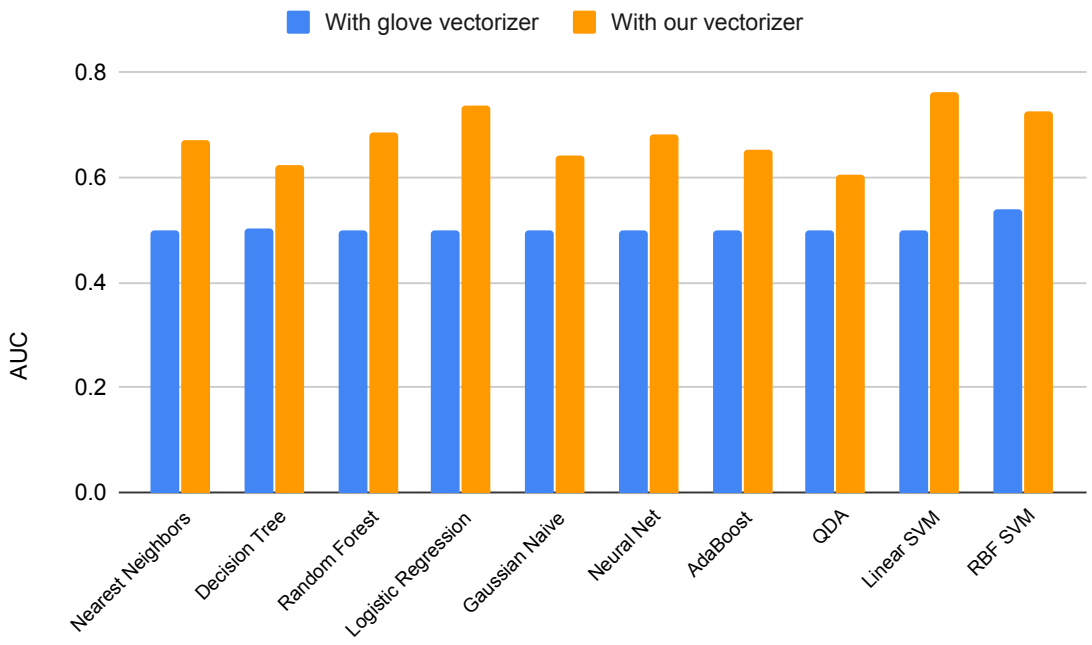


Figure 5.1: Comparison of our word vectorizer model compared with Glove while classifying Brunet2014 data. The left bar is the performance of Glove and right bar represents the performance of our vectorizer in terms of AUC.

Figure 5.1 illustrates the comparison of our vectorizer with Glove vectorizer. The horizontal axis represents the classifiers used and the vertical axis shows the performance of different classifiers in terms of AUC. Using Glove as the vectorization model

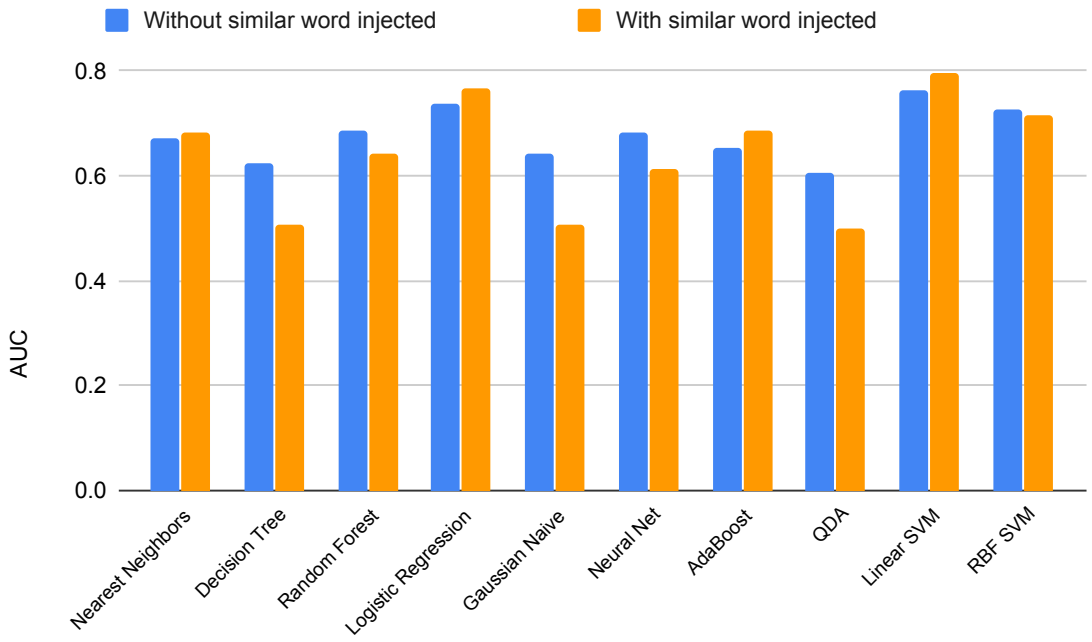


Figure 5.2: Comparison of performance in AUC without and with similar word injection in the train data respectively illustrated by left and right bar of each bar group.

provides AUC scores in the range of 0.5 - 0.55. The performance is significantly improved while vectorized with our software specific word embedding model as shown with the right bar compared to the left bar for every classifier group. Thus we conclude that the software specificity of our new vectorization (trained on 300 literature sources from the SE domain) significantly improves results.

5.4 Data Augmentation Results

After success on the first stage, we augment the dataset by injecting similar words into the data according to figure 4.5. First, we explore how similar word injection performs in providing total domain context. The performance after injecting similar words into the training data is shown in figure 5.2. As seen from the figure, we get a mixed range of performance by this approach. The performance of most of the classifiers decreases following this method. This is because the injection of similar words only to the train data increases the statistical weight and bias towards only one domain (namely Stack Overflow). On the other hand, because the test data

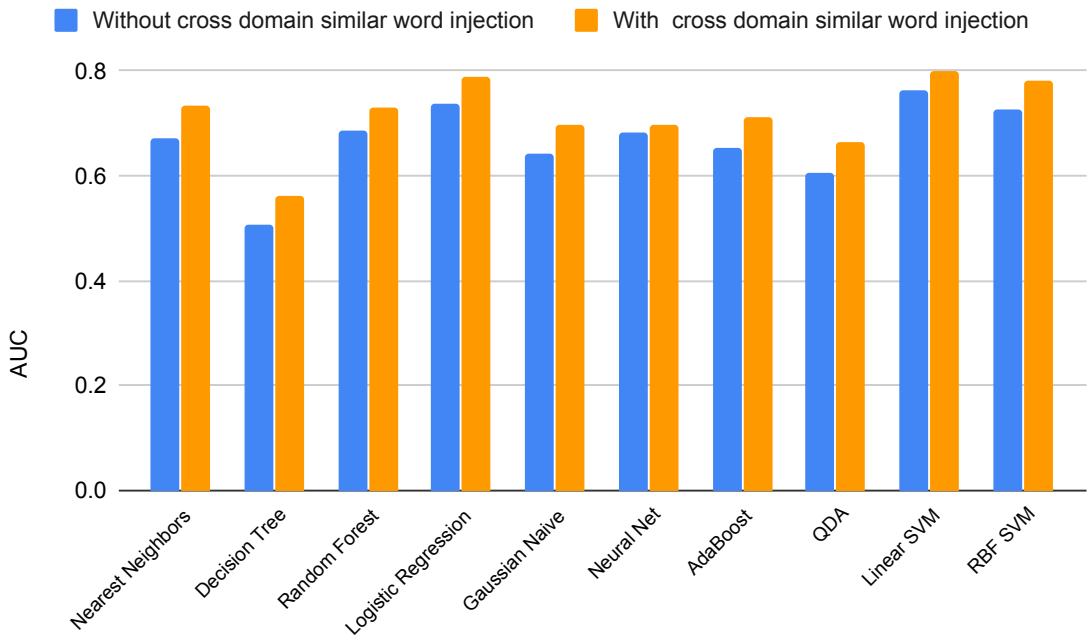


Figure 5.3: Comparison of performance in AUC without and with cross similar data injection in both train and test data. Left bar represent AUC score without using cross similar word injection while right bar of every group shows the AUC after using cross injection.

is from another domain (Github), the high bias in one domain prevails during the probability analysis resulting in a missed classification of the test data. This problem of high bias is removed by using instead a cross-domain similar word injection as result shown in figure 5.3. Results shows the AUC score of all classifiers is higher after augmentation using cross-domain similar word injection. Some of the scores, namely Linear SVM at 0.80, are SOTA in *cross-domain* design discussion classification studies. Most of the related work in classifying design discussion does not report or apply cross-domain classification. Only one study by Viviani et al. [83] attempts to show the performance of their classification while classifying an unknown dataset. A gold standard dataset was manually created by them which was used to justify the performance of their model with unknown data. While that classifier produced better results than ours, the gold standard dataset was not cross-domain data. It was made from Github discussions similar to the train data. We believe that this study is unique in using datasets, vectorizers, and attempts to tackle cross-domain design discussion classification.

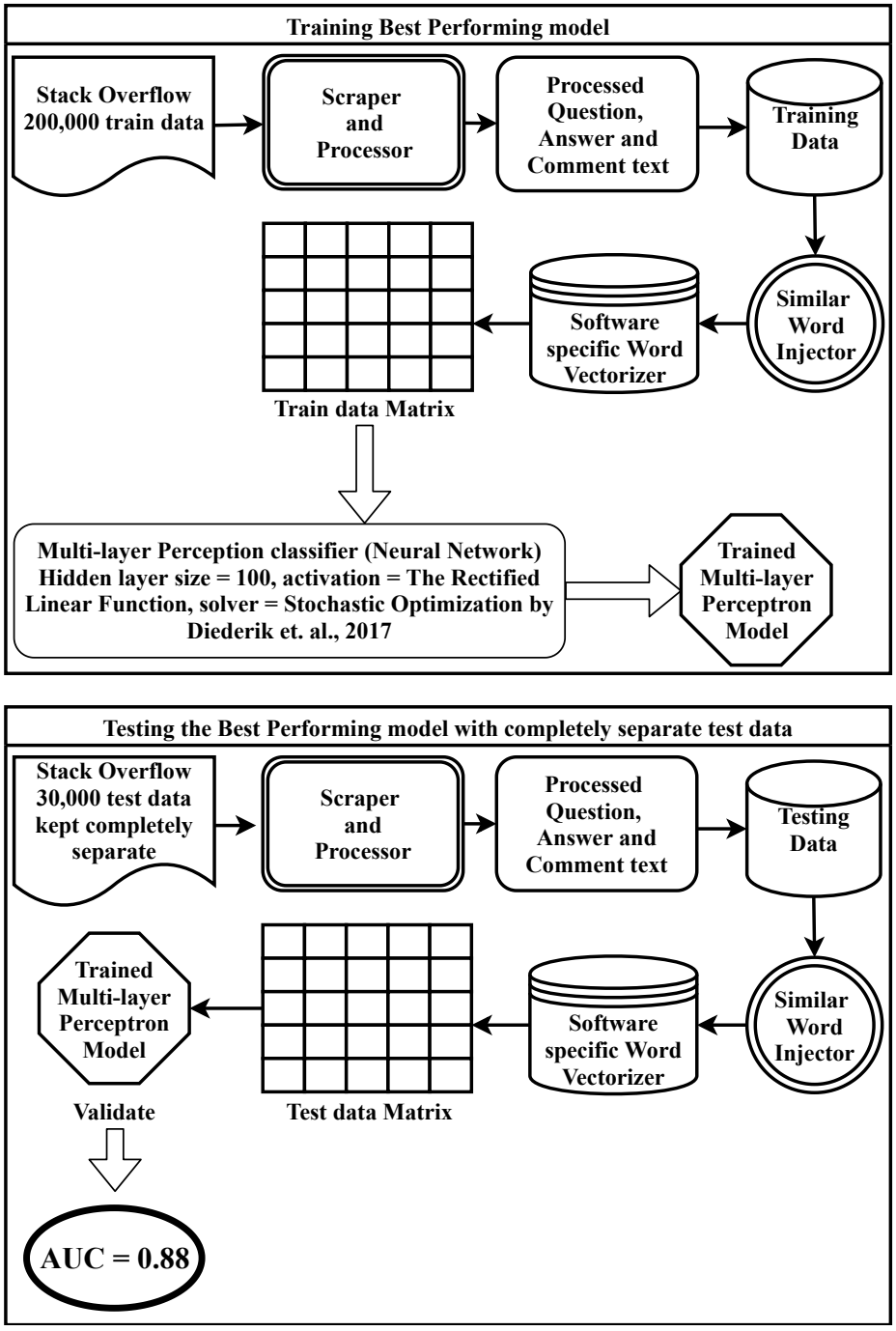


Figure 5.4: Protocol map of the test data validation

Within Dataset Results We use the software specific word vectorizer along with similar word augmentation for train and test data and a simple neural network (Multilayer Perceptron classifier) illustrated in figure 5.4 yielding 88% AUC (MCC: 0.76), which is a state-of-the-art score for classifying unknown data within the *same* domain (in this case, Stack Overflow design discussions). This compares directly with the approach NewBest we described in §3.2 which obtained AUC of 0.84 (MCC 0.63).

5.5 Summary

In summary, we have demonstrated how we might overcome three challenges in design mining studies that we outlined in §4.2. First, we illustrate how using a software-specific word vectorizer can improve the performance significantly with our results. Our results also demonstrate a significant improvement in the AUC after data augmentation. Finally, we show substantial improvements in classification accuracy for across-domains by implementing a collective step of using more labeled data, software specific context, and cross-domain context. We also present a state-of-the-art approach for classifying unknown discussion within the domain.

Chapter 6

Discussion, Future Work and Conclusion

6.1 Introduction

In this chapter, we discuss our finding of the thesis along with some potential future work in this field. We start our discussion with the threats to the validity of this thesis. Then we discuss some of the ways we think design mining can be improved. We discuss the researcher degrees of freedom we experience during our experiments and discuss briefly three steps to resolve the problem. We also explain our choice of the training data size and discuss the rationale behind it. Then, we discuss the effectiveness of software-specific vocabularies and some of the other avenues to explore in this area.

6.2 Discussion

We discuss the implications of our results for future design mining studies, discuss the ways to improve future studies, and account for researcher degrees of freedom. We begin with threats to validity for our work in this thesis.

6.2.1 Threats to Validity: Bad Analytics Smells

We use the concept of bad analytics smells from Menzies and Sheppherd [48]. In that paper, the authors introduce a succinct list of twelve potential study design flaws in

analytics research, and suggest some mitigations. Here, we list the smells this work might emit, and ignore the ones we believe we have dealt with or do not apply.

(a) *Using suspect data*: we rely extensively on the tags that the author and moderators of Stack Overflow created. However, we conduct some statistical tests to validate the data labels. We also re-use previous datasets, but several different ones, and contribute a new dataset to the literature. As we mentioned, it is possible the context transfer approach tainted the test data with training context (and vice versa), since we rely on word vectors from the same domains as a fine-tuning approach. However, the specificity of the data involved and the overall large volumes of data used to train the vectors makes us think this risk is minimal.

(b) *Low power*: ultimately, the design mining data relies on a limited set of labeled data (or makes the possibly invalid assumption that the tagging in Stack Overflow reflects real design). While more data is always better, there is always an optimal number of data we can use. However, we were not able to find how much is enough. We show some preliminary results below at figure 6.1

(c) *No data visualizations*: we present a limited set of visualizations because text data is always difficult to visualize. However, we provided some statistical analysis in the data validation in §4.3.1.3.

(d) *Not tuning*: Because of our focus on making the vectorizer and data generalized for models, we did not emphasize optimization for any specific model. In the case of neural network, we use the most simple one with default optimization. Thus, it is possible our results are an under-estimate of a properly tuned model. We use a cutoff point of 0.6 for word similarity indexes in data augmentation. We experimented with several similarity indexes and 0.6 seem to work the best in terms of AUC and the size of the vocabulary. However, we do not validate this choice during our experiments for this thesis. One other limitation is that we used 300 different literature sources to extract text to train our word vectorizer. We maintained little to no relationship between the sources of the data. We took this approach to remove bias towards a specific kind of data. However, this was a missed opportunity for us to direct our model in a very specific way (e.g., by only mining design-related papers or texts). Furthermore, the disadvantage

of using a limited number of literature sources is reflected in the total number vocabulary in the training corpus of the vectorizer.

(e) *Not justifying the choice of learner*: we report on the standard machine learning approaches including some deeper network models. It is possible but some other learners could improve results.

(f) *Not exploring simplicity*: this smell argues that we should ensure we do not overfit our results with complex models. We explicitly test our models on non-similar (cross-project/domain) datasets.

One other limitation is that expecting design mining classifiers trained on one dataset to transfer to totally different datasets is improbable. That is to say, machine learning is innately tightly coupled and optimized for a particular dataset. However, as table 3.2 shows, these design datasets are not so different. Furthermore, the literature on design patterns, and our intuition from consulting with many different projects, supports the notion that some high-order design structure crosses project boundaries.

6.2.2 Improving Design Mining

Recall that our first research objective, **RQ-1**(Is it possible to replicate a previous study and improve that study?), was whether it was possible to accurately label a given natural language discussion as pertaining to software design. We showed that with judicious use of analysis choices, our classifiers (NewBest) outperformed previous studies, and a naive baseline classifier in this task. However, with regard to **RQ-2**(To what extent can we transfer classifiers trained on one data set to other data sets?), the conclusion stability of our approach was low. Our accuracy suffers once we apply that same classifier on entirely different discussion sets.

The problem lies in the overfitting—biasing—of the classifier to a particular dataset, given our study begins by making analysis choices that improve local (within-dataset) accuracy. This is because researchers are implicitly or explicitly conditioning on the dataset they analyze. The result is poor conclusion stability. Can we do better?

Our results show that biased learners are a problem; applying a classifier trained on one dataset to a different dataset had very poor performance. Design discussions can change venue (issue trackers vs chat vs StackOverflow), are highly dependent on *who* is communicating, and have different conceptualizations of software design. For

example, if we train the document embedding on StackOverflow, and apply it to the Brunet dataset of issue discussions, our accuracy is 0.48, well below the simple ZeroR classifier which would predict the majority class, and achieve around 86%. If we add the Brunet data to the document embedding, the accuracy doubles. However, this requires one to add and retrain the embedding each time, which is time-consuming.

What is the generative model that leads to design discussion features? Viviani et al. [83] have begun promising work in this area by looking at higher-order features such as the location of the discussion or the number of comments on an issue. However, we likely need to look at even more robust features such as social interactions and other contextual cues.

6.2.3 The Role of Researcher Degrees of Freedom

Researcher degrees of freedom (RDOF) [22, 21] refers to the multiple, equally probable analysis paths present in any research study, any of which might lead to a *significant* result. Failure to account for researcher degrees of freedom directly impacts conclusion stability and overall practical relevance of the work, as shown in papers such as Di Nucci et al. [57] and Hill et al. [28]. For example, for many decisions in mining studies like this one, there are competing views on when and how they should be used, multiple possible pre-processing choices, and several ways to interpret results. Indeed, the many combinations possible in figure 3.2 is actually over-simplified, given the actual number of choices we encountered. Furthermore, the existence of some choices may not be apparent to someone not deeply skilled in these types of studies.

One possible approach is to use toolkits with intelligently tuned parameter choices. Hyper-parameter tuning is one such example of applying machine learning to the problem of machine learning, and research is promising [88]. Clearly, one particular analysis path will not apply broadly to *all* software projects. What we should aim for, however, is to outline the edges of where and more importantly, why these differences exist. We think there are three major steps to take to help solve the RDOF question and improve conclusion stability.

1. Use protocol maps or other graphical models to clearly outline the degrees of freedom, and chosen paths. Improve study reporting in general. There are lessons to be learned from scientific workflow software already well-developed in, for example, high-energy physics.

2. For confirmatory studies, pre-registered hypotheses and protocols, like in medicine, make it clear what conclusions are valid, and which might be conditioned on the observed results.

3. Improve understanding of the concept of RDOF. Develop tools that can automatically generate protocol maps based on common data science pathways. For example, we could apply our existing strengths in software slicing to analyze available parameter choices and dependencies in machine learning frameworks.

6.2.4 Choice of Training Size

The box plot in figure 6.1 represents the AUC of the 10 classifiers with different sizes of chunks of the train data. We experiment with four chunks: 50,000, 100,000, 150,000, 200,000. The x-axis represents the chunk size. From the plot, we can see that chunk of 50,000 train data has the highest median (dark black line inside the boxes) with a relatively high range for minimum and maximum. Although the median is high, most of the classifiers perform below the median. One classifier performs extremely high contributing to the high median for this chunk. However, our decision to go with a chunk of 200,000 train data is because of the low range illustrated by the right-most box. Although it produces a lower median than the first chunk, most of the classifier performs above the median and within the 25 to 75 percentile (this is how we know that we have made the data generalized for most of the classifiers). The low median is because of one outlier (bottom of the box) that performs very poorly (decision tree–0.58).

6.2.5 The Effectiveness of Software Specific Vocabularies

Most neural language models are intended for general-purpose language tasks, such as translation and question answering. However, these models do not deal well with domain-specific terminology, because such terminology is a small fraction of the overall training data [19].

Introducing software vocabularies resulted in a big improvement in accuracy of the design mining classifier, suggesting that even fairly simple contextual approaches, such as similar word augmentation, and software specific vectors, can be a big help. The importance of individual project context, as opposed to general software context, is still unclear (and training models are less useful with fewer data to work with).

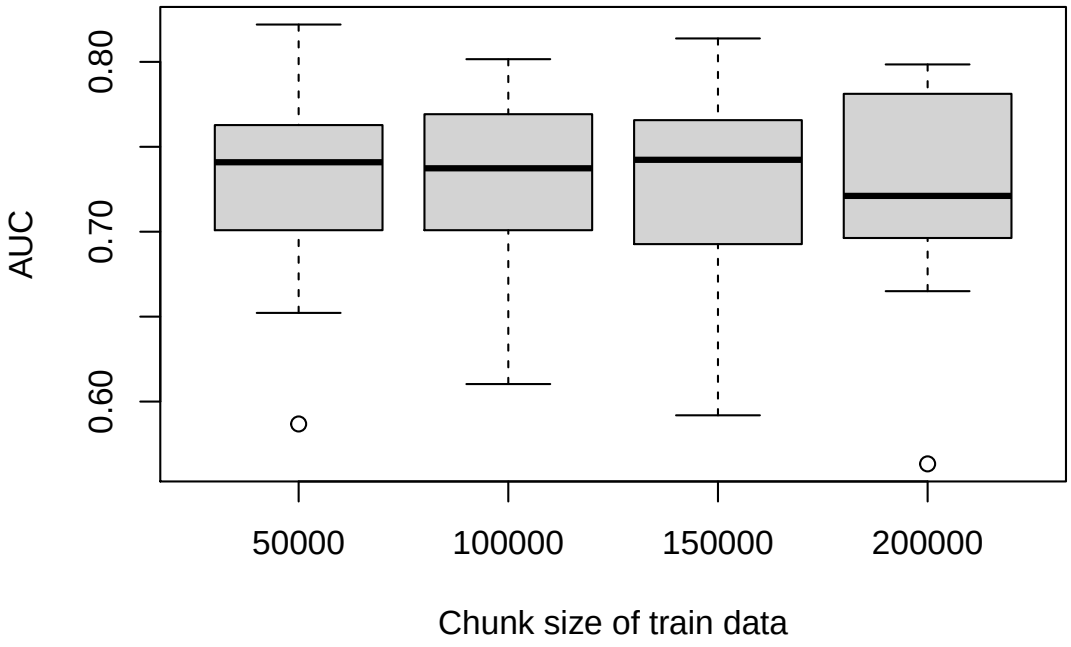


Figure 6.1: Performance of the 10 classifiers after training with different size of train data. The four boxes represent four chunk size of the data we used for training to explore which one works better. We explain our decision to go with the 200,000 chunk size in §6.2.4

.

Novielli et al. suggest that fine-tuning for sentiment analysis, at the project level, can be quite helpful [56]. We intend to investigate how these vector models can help with fine-tuning general-purpose deep learning models like ULMFiT or BERT, as suggested (for source code) by recent work from Karampatsis et al. [37].

6.3 Future Work

Like Shakiba et al. [68] and Viviani [84, 83], we envision a design tagging tool that can be applied broadly to all design discussions. This would be a necessary first step in automatically analyzing design decisions and recommending alternatives or improvements. To get to that point, the community needs to increase the amount of data available for these sorts of mining tasks. Stack Overflow, as we demonstrate, is one potentially rich source for labeled data. More importantly, a better understanding of the nature of design discussions is needed. Expanding on qualitative studies such as Viviani et al. [84] or those surveyed in van Vliet and Tang [82] is the likely way forward, as opposed to blindly mining relatively small samples of software artifacts.

The importance of design mining to practitioners is largely speculative at this point, as researchers try to improve the state of the art (SOTA) to the point it can be useful in practice. More efforts in bringing even preliminary design mining results into prototype tools is important to understand how (and why) practitioners might use design mining.

6.4 Conclusion

Chapter 3 has shown a new state of the art results in classifying software design discussions, with a maximum AUC score of 0.84 (using the combination of support vector machines, rebalancing, and word embedding vectorization). We also showed that the conclusion stability for design mining remains poor.

Chapter 4 represents a continuation of our previous work in chapter 3 by introducing a concept of building software specific word vectorizer to improve word embedding for software engineering related discussions backed by results on applying it on a wide variety of classifiers. We showed a state of the art results on the cross-project classification problem using this approach. This thesis also demonstrates the application of augmentation (similar word injection) to transfer context between cross-domain with experimented results and discussions. We also restrict ourselves from going further

with the classifiers by tuning the hyper-parameters and exploring plausible explanation on the difference in performance for the different classifiers. However, we believe that our idea of software specific word embedding and context transfer, along with an efficient choice of classifier and parameter optimization, can improve the study of design discussion mining even further.

Bibliography

[1] Rana Alkadhi, Teodora Lata, Emitza Guzmany, and Bernd Bruegge. Rationale in development chat messages: An exploratory study. may 2017.

[2] Rana Alkadhi, Manuel Nonnenmacher, Emitza Guzman, and Bernd Bruegge. How do developers discuss rationale? IEEE, mar 2018.

[3] Jorge Aranda and Gina Venolia. The secret life of bugs: Going past the errors and omissions in software repositories. 2009.

[4] Deeksha Arya, Wenting Wang, Jin L.C. Guo, and Jinghui Cheng. Analysis and detection of information types of open source software issue discussions. In *International Conference on Software Engineering (ICSE)*. IEEE, May 2019.

[5] Vimala Balakrishnan and Lloyd-Yemoh Ethel. Stemming and lemmatization: A comparison of retrieval performances. *Lecture Notes on Software Engineering*, 2(3):262–267, 2014.

[6] Abdul Ali Bangash, Hareem Sahar, Abram Hindle, and Karim Ali. On the time-based conclusion stability of cross-project defect prediction models. *Empirical Software Engineering*, 2020.

[7] B. Bazelli, A. Hindle, and E. Stroulia. On the personality traits of stackoverflow users. In *2013 IEEE International Conference on Software Maintenance*, pages 460–463, 2013.

[8] Alec T. Beall and Jessica L. Tracy. Women are more likely to wear red or pink at peak fertility. *Psychological Science*, 24(9):1837–1841, jul 2013.

[9] T. J. Biggerstaff. Design recovery for maintenance and reuse. *Computer*, 22(7):36–49, 1989.

[10] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[11] A. Brooks, M. Roper, M. Wood, J. Daly, and J. Miller. *Replication's Role in Software Engineering*, pages 365–379. Springer London, London, 2008.

[12] Joao Brunet, Gail C. Murphy, Ricardo Terra, Jorge Figueiredo, and Dalton Serey. Do developers discuss design? In *Working Conference on Mining Software Repositories*, Hyderabad, India, September 2014.

[13] Nitesh Chawla, Kevin Bowyer, Lawrence Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

[14] Kevin Crowston, Hala Annabi, and James Howison. Defining open source software project success. *Proceedings of the International Conference on Information Systems*, 06 2003.

[15] D. Cubranic and G. C. Murphy. Hipikat: recommending pertinent software development artifacts. In *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 408–418, 2003.

[16] Everton da Silva Maldonado, Emad Shihab, and Nikolaos Tsantalis. Using natural language processing to automatically detect self-admitted technical debt. 43(11):1044–1062, nov 2017.

[17] Francisco Gomes de Oliveira Neto, Richard Torkar, Robert Feldt, Lucas Gren, Carlo Furia, and Ziwei Huang. The evolution of statistical analysis in empirical software engineering research. preprint arXiv:1706.00933, arXiv, 2017.

[18] Francisco Gomes de Oliveira Neto, Richard Torkar, Robert Feldt, Lucas Gren, and Carlo A. Furia. Evolution of statistical analysis in empirical software engineering research: Current state and steps forward. *Journal of Systems and Software*, 2019.

[19] Vasiliki Efstathiou, Christos Chatzilenas, and Diomidis Spinellis. Word embeddings for the software engineering domain. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18, page 38–41, New York, NY, USA, 2018. Association for Computing Machinery.

[20] Neil Ernst and Gail C Murphy. Case Studies in Just-In-Time Requirements Analysis. In *Empirical Requirements Engineering Workshop at RE*, pages 1–8, Chicago, September 2012.

[21] Andrew Gelman, Jennifer Hill, and Masanao Yajima. Why we (usually) don't have to worry about multiple comparisons. *Journal of Research on Educational Effectiveness*, 5:189–211, 2012.

[22] Andrew Gelman and Eric Loken. The garden of forking paths: Why multiple comparisons can be a problem, even when there is no "fishing expedition" or "p-hacking" and the research hypothesis was posited ahead of time. Technical report, Colombia University, 2013.

[23] K. V. Ghag and K. Shah. Comparative analysis of effect of stopwords removal on sentiment classification. In *2015 International Conference on Computer, Communication and Control (IC4)*, pages 1–6, 2015.

[24] Omar S. Gómez, Natalia Juristo, and Sira Vegas. Understanding replication of experiments in software engineering: A classification. *Information and Software Technology*, 56(8):1033–1048, aug 2014.

[25] Jesús M. González-Barahona and Gregorio Robles. On the reproducibility of empirical software engineering studies based on data retrieved from development repositories. *Empirical Software Engineering*, 17(1-2):75–89, oct 2011.

[26] I Hemalatha, GP Saradhi Varma, and A Govardhan. Preprocessing the informal text for efficient sentiment analysis. *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*, 1(2):58–61, 2012.

[27] Steffen Herbold. A systematic mapping study on cross-project defect prediction, 2017.

[28] Emily Hill, Shivani Rao, and Avinash Kak. On the use of stemming for concern location and bug localization in java. In *International Working Conference on Source Code Analysis and Manipulation*. IEEE, September 2012.

[29] Abram Hindle, Earl T. Barr, Zhendong Su, Mark Gabel, and Premkumar Devanbu. On the naturalness of software. In *Proceedings of the International Conference on Software Engineering*, page 837–847, 2012.

[30] Abram Hindle, Christian Bird, Thomas Zimmermann, and Nachiappan Nagappan. Do topics make sense to managers and developers? 20(2):479–515, 2015.

[31] Abram Hindle, Neil Ernst, Michael W Godfrey, and John Mylopoulos. Automated topic naming to support cross-project analysis of software maintenance activities. In *MSR*, pages 1–10, Honolulu, 2011.

[32] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *Annual Meeting of the Association for Computational Linguistics*, 2018.

[33] Jian Hu, Lujun Fang, Yang Cao, Hua-Jun Zeng, Hua Li, Qiang Yang, and Zheng Chen. Enhancing text clustering by leveraging wikipedia semantics. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, page 179–186, New York, NY, USA, 2008. Association for Computing Machinery.

[34] Magne Jørgensen, Tore Dybå, Knut Liestøl, and Dag I.K. Sjøberg. Incorrect results in software engineering experiments: How to improve research practices. *Journal of Systems and Software*, 116:133–145, Jun 2016.

[35] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.

[36] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[37] Rafael-Michael Karampatsis, Hlib Babii, Romain Robbes, Charles Sutton, and Andrea Janes. Big code != big vocabulary: Open-vocabulary models for source code. In *Proceedings of the International Conference on Software Engineering*, 2020.

[38] Rick Kazman and Humberto Cervantes. *Designing Software Architectures: A Practical Approach*. SEI Series in Software Engineering. Addison-Wesley, 2016.

[39] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K. El Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. 28(8):721–734, 2002.

[40] Barbara Kitchenham, Lech Madeyski, and Pearl Brereton. Meta-analysis for families of experiments in software engineering: a systematic review and reproducibility and validity assessment. *Empirical Software Engineering*, Jul 2019.

[41] Barbara Kitchenham, Lech Madeyski, David Budgen, Jacky Keung, Pearl Brereton, Stuart Charters, Shirley Gibbs, and Amnart Pohthong. Robust statistical methods for empirical software engineering. 22(2):579–630, jun 2016.

[42] Ekrem Kocaguneli and Tim Menzies. Software effort models should be assessed via leave-one-out validation. *Journal of Systems and Software*, 86(7):1879–1890, jul 2013.

[43] Rahul Krishna, Suvodeep Majumder, Tim Menzies, and Martin Shepperd. Bad smells in software analytics papers. Technical report, ArXiv, 2018.

[44] Rahul Krishna, Tim Menzies, and Wei Fu. Too much automation? the bellwether effect and its implications for transfer learning. In *International Conference on Automated Software Engineering*, pages 122–131, 2016.

[45] Yitan Li, Linli Xu, Fei Tian, Liang Jiang, Xiaowei Zhong, and Enhong Chen. Word embedding revisited: A new representation learning and explicit matrix factorization perspective. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[46] Victoria López, Alberto Fernández, and Francisco Herrera. On the importance of the validation technique for classification with imbalanced datasets: Addressing covariate shift when data is skewed. *Information Sciences*, 257:1 – 13, 2014.

[47] Tim Menzies and Martin Shepperd. Special issue on repeatable results in software engineering prediction. *Empirical Software Engineering*, 17(1):1–17, Feb 2012.

[48] Tim Menzies and Martin Shepperd. "Bad smells" in software analytics papers. *Information and Software Technology*, 112:35–47, August 2019.

[49] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[50] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. In *Pro-*

ceedings of the International Conference on Language Resources and Evaluation (LREC 2018), 2018.

[51] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality, 2013.

[52] George A Miller. *WordNet: An electronic lexical database*. MIT press, 1998.

[53] Tiago Oliveira Motta, Rodrigo Rocha Gomes e Souza, and Claudio Sant'Anna. Characterizing architectural information in commit messages. In *Proceedings of the Brazilian Symposium on Software Engineering*. ACM Press, 2018.

[54] G. C. Murphy, D. Notkin, and K. J. Sullivan. Software reflexion models: bridging the gap between design and implementation. *IEEE Transactions on Software Engineering*, 27(4):364–380, April 2001.

[55] Meiyappan Nagappan, Thomas Zimmermann, and Christian Bird. Diversity in software engineering research. 2013.

[56] Nicole Novielli, Fabio Calefato, Davide Dongiovanni, Daniela Girardi, and Filippo Lanubile. Can we use se-specific sentiment analysis tools in a cross-platform setting? In *International Conference on Mining Software Repositories*, 2020.

[57] Dario Di Nucci, Fabio Palomba, Damian A. Tamburri, Alexander Serebrenik, and Andrea De Lucia. Detecting code smells using machine learning techniques: Are we there yet? In *International Conference on Software Analysis, Evolution and Reengineering (SANER)*, March 2018.

[58] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, October 2010.

[59] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[60] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[61] Lesley M. Pickard, Barbara A. Kitchenham, and Peter W. Jones. Combining empirical results in software engineering. *Information and Software Technology*, 40(14):811 – 821, 1998.

[62] Romain Robbes and Andrea Janes. Leveraging small software engineering data sets with pre-trained neural networks. In *International Conference on Software Engineering: New Ideas and Emerging Results*, ICSE-NIER '19, pages 29–32, 2019.

[63] M. P. Robillard. What makes apis hard to learn? answers from developers. *IEEE Software*, 26(6):27–34, Nov 2009.

[64] M. P. Robillard and N. Medvidovic. Disseminating architectural knowledge on open-source projects: A case study of the book "architecture of open-source applications". In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 476–487, May 2016.

[65] Martin P. Robillard and Robert Deline. A field study of api learning obstacles. *Empirical Softw. Engg.*, 16(6):703–732, December 2011.

[66] Konstantinos Sechidis, Grigorios Tsoumakas, and Ioannis Vlahavas. On the stratification of multi-label data. In Dimitrios Gunopulos, Thomas Hofmann, Donato Malerba, and Michalis Vazirgiannis, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 145–158, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[67] Arman Shahbazian, Youn Kyu Lee, Duc Le, Yuriy Brun, and Nenad Medvidovic. Recovering architectural design decisions. IEEE, apr 2018.

[68] Abbas Shakiba, Robert Green, and Robert Dyer. FourD: do developers discuss design? revisited. In *Proceedings of the 2nd International Workshop on Software Analytics - SWAN 2016*. ACM Press, 2016.

[69] Tushar Sharma, Vasiliki Efstathiou, Panos Louridas, and Diomidis Spinellis. On the feasibility of transfer-learning code smells using deep learning. Technical Report 1904.03031v2, arXiv, 2019.

[70] Martin Shepperd. Replication studies considered harmful. 2018.

[71] S. E. Sim and R. C. Holt. The ramp-up problem in software projects: a case study of how software immigrants naturalize. In *Proceedings of the 20th International Conference on Software Engineering*, pages 361–370, April 1998.

[72] Joseph P. Simmons, Leif D. Nelson, and Uri Simonsohn. False-positive psychology: Undisclosed flexibility in data collection and analysis allows presenting anything as significant. *Psychological Science*, 22(11):1359–1366, oct 2011.

[73] Software Engineering Institute. What is your definition of software architecture? Fact sheet, Software Engineering Institute, 2010.

[74] Mohamed Soliman, Matthias Galster, Amr R Salama, and Matthias Riebisch. Architectural knowledge for technology decisions in developer communities: An exploratory study with stackoverflow. In *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 128–133. IEEE, 2016.

[75] Igor Steinmacher, Tayana Conte, Marco Aurélio Gerosa, and David Redmiles. Social barriers faced by newcomers placing their first contribution in open source software projects. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work and Social Computing*, CSCW '15, page 1379–1392, New York, NY, USA, 2015. Association for Computing Machinery.

[76] Igor Steinmacher, Igor Scaliante Wiese, Tayana Conte, Marco Aurélio Gerosa, and David Redmiles. The hard life of open source software project newcomers. In *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE 2014, page 72–78, New York, NY, USA, 2014. Association for Computing Machinery.

[77] Margaret-Anne Storey, C. Williams, Neil A. Ernst, A. Zagalsky, and E. Kalliamvakou. Methodology matters: How we study socio-technical aspects in software engineering. Technical Report arXiv:1905.12841, arXiv, 2019.

[78] Chakkrit Tantithamthavorn. *Towards a Better Understanding of the Impact of Experimental Components on Defect Prediction Models*. PhD thesis, Nara Institute of Science and Technology, 2016.

[79] Antoine Tremblay and Benjamin V Tucker. The effects of n-gram probabilistic measures on the recognition and production of four-word sequences. *The Mental Lexicon*, 6(2):302–324, 2011.

[80] Jason Tsay, Laura Dabbish, and James Herbsleb. Let's talk about it: Evaluating contributions through discussion in github. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, page 144–154, New York, NY, USA, 2014. Association for Computing Machinery.

[81] Jilles van Gurp and Jan Bosch. Design erosion: problems and causes. *Journal of Systems and Software*, 61(2):105 – 119, 2002.

[82] Hans van Vliet and Antony Tang. Decision making in software architecture. *Journal of Systems and Software*, 117:638–644, jul 2016.

[83] G. Viviani, M. Famelis, X. Xia, C. Janik-Jones, and G. C. Murphy. Locating latent design information in developer discussions: A study on pull requests. *IEEE Transactions on Software Engineering*, pages 1–1, 2019.

[84] Giovanni Viviani, Calahan Janik-Jones, Michalis Famelis, and Gail C. Murphy. The structure of software design discussions. ACM Press, 2018.

[85] Giovanni Viviani, Calahan Janik-Jones, Michalis Famelis, Xin Xia, and Gail C. Murphy. What design topics do developers discuss? 2018.

[86] Peng Wang, Bo Xu, Jiaming Xu, Guanhua Tian, Cheng-Lin Liu, and Hongwei Hao. Semantic expansion using word embedding clustering and convolutional neural network for improving short text classification. *Neurocomputing*, 174:806 – 814, 2016.

[87] Eoin Woods. Software architecture in a changing world. *IEEE Software*, 33(6):94–97, Nov 2016.

[88] Tianpei Xia, Rahul Krishna, Jianfeng Chen, George Mathew, Xipeng Shen, and Tim Menzies. Hyperparameter optimization for effort estimation. Technical report, ArXiv, 2018.

[89] Farida El Zanaty, Toshiki Hirao, Shane McIntosh, Akinori Ihara, and Kenichi Matsumoto. An empirical study of design discussions in code review. ACM Press, 2018.

[90] Thomas Zimmermann, Nachiappan Nagappan, Harald Gall, Emanuel Giger, and Brendan Murphy. Cross-project defect prediction: A large scale experiment on data vs. domain vs. process. pages 91–100, 2009.