ABSTRACT

VIVEK. LocalMine - Probabilistic Keyword Model for Software Text Mining. (Under the direction of Dr. Timothy Menzies.)

Traditionally software change messages are classified as "bug-fixing commits" or "not" using a keyword-based model. This classification is very useful in defect prediction, automatic program repair and fault localization. However, these keywords are usually generic like "bug", "fix" or "patch". These special words do not characterize all variants of bug-fixing change messages. In some cases the keywords are very specific to a software project. And, developing a keyword model for another project requires manually discovering keywords specific to that project. Recently, it has been seen that automated machine learning approaches for change message classification using probabilistic models have been successful. However, these studies are still based on generic "special" keywords. To address these problems we propose LocalMine which is based on probabilistic approaches for keyword modeling. LocalMine does not rely on earlier generic keywords for classification, rather it creates features based on the probabilistic model that discovers the keywords for a particular project. Then it uses these features to train a machine learning binary classifier. The results are promising for change message classification. For commit messages dataset, LocalMine achieves 82% recall (median over 3 projects) with a 61% F1 score. This is significantly better than results from state of the art models that use keyword methods (76% recall and 26% F1 score). One interesting feature of LocalMine is that it generalizes to different kinds of natural-language software texts. For example, on tech debt dataset it has 76% recall and 64% F1 score, compared to 72% and 59% for the baseline model, respectively. LocalMine can be very successful in software text mining. These results make us recommend LocalMine for keyword modeling.

LocalMine - Probabilistic Keyword Model for Software Text Mining

by
Vivek

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh, North Carolina

2019

APPROVED BY:

Dr. Min Chi	Dr. Kathryn Stolee

Dr. Timothy Menzies
Chair of Advisory Committee

DEDICATION

To my parents, Rajendra Prasad Singh and Malti Devi.

BIOGRAPHY

The author was born in small town of Danapur in the state of Bihar in India. He obtained his Bachelor of Technology degree in Electrical Engineering (Power) in 2015 from Indian Institute of Technology Delhi in New Delhi, Delhi, India. He worked as software engineer for 2 successful startups in India before starting his Master of Science degree in Computer Science in Fall of 2018 from North Carolina State University, Raleigh, NC, USA. With a massive interest in machine learning and text mining, the author undertook this thesis work under Dr. Tim Menzies following interesting coursework of Foundations of Software Science in Fall 2018. The author also underwent summer internship at LexisNexis, Raleigh, NC, USA working on natural language processing and machine learning application to understand user query intent in search for legal documents and research.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Timothy Menzies and RAISE lab for their help and guidance. In particular, I would like to thank Dr. Menzies for believing in my approach and encouraging me to explore software text mining. For different queries and datasets I often sought help from Rahul Krishna, Amritanshu Agrawal, Fahmid Fahid and Huy (Ken) Tu. I am grateful for their constant help and guidance in various times of need.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER

1

INTRODUCTION

Software projects have become pervasive in industry. With increase in the number of projects, increases the number of software related texts. For instance, most of the software projects uses version control system. Github is a popular version control system. Github reports (octoverse.github.com) 31 million developers, 96 million repositories and 200 million pull requests in 2018. Another source of software texts is Stack Overflow (a question answering platform) originally created for professional and enthusiast programmers. According to insights.stackoverflow.com, there are 100 million users, 18 million questions and 28 million answers. This is humongous volume of texts. To extract knowledge from these texts we need to have automated text mining models. Therefore, it is very important to study software text mining.

To limit the scope for the purpose of this thesis, we focus on a specific type of software text mining. We will study change messages. Change messages are very crucial to version control system. It is, generally, a small size text that captures information on the purpose of commit. In this way not only does version control system maintain the software, it also helps maintain history of a particular piece of code. It can be well established with a particular change, who did the change, for what

purpose, at what time, what files that change corresponds to and why that change. *Why that change* is very important as this natural language message can give a wealth of information on the nature of the change introduced.

Classifying a change message as a bug-fixing commit or not requires understanding of the natural-language texts in the commit message. It is a challenge to understand natural-language software texts because they do not follow the natural convention of natural language, particularly related to grammar. Traditionally change message classification has been done using a generic keyword dictionary [1] [2]. We have also seen that probabilistic methods have shown some success too, particularly, Latent Dirichlet Allocation (LDA) [3] [4] and probabilistic Latent Semantic Analysis[5].

Taking inspiration from success of dictionary models that rely on "special" words, our motivation is to create a keyword model that is automated and can be applied to any version control project. An automated model can be built on top of probabilistic approaches to capture keyword. In this project, we explore automated machine learning methods to predict if a change message corresponds to a bug-fix. Most of the git projects are managed by only a certain group of individuals, hence, they tend to have a characterizing way of expressing commit messages. However, as all of them are software projects, certain software engineering domain specific words are frequently used. These general keywords are more prominent to identify a bug-fixing commit, but not exhaustive. Due to this localized nature of software management, we need to explore a model that learns these words specific to a project. In a nutshell, we will explore an automated generic model that learns "local" keyword features for discovering bug-fixing messages.

Mauckza et al. [6] proposed a variation of keyword based approach to automate classification of change messages. This work has been based on Mockus et al. [1] natural language "modification request" words. Keyword based models have been a benchmark for change message classification. In this project, we aim to question if a generic keyword dictionary is suitable for all projects. Or if there exists some localized natural language reference of equal importance with respect to global software engineering specific keywords. This leads to our first research question

- **RQ1:** Does an automated probabilistic model perform better than generic dictionary based classifiers?

A common trend is observed that most of studies in software engineering text mining rely on these

keywords specified by [1]. In Zhong et al [7] if words "bug" or "fix" appeared in the commit message, it has been classified as bug-fixing commit. Similarly, Martinez et al [8] we see that words that match "bug", "fix" or "patch" are called as bug-fix patterns. We see a similar way of labeling or classification in Kim et al [9] and Osman et al [10] approaches for text classification of bug-fixing patterns. We have seen that probabilistic approaches have been used earlier for change message classification. In particular, we will take inspiration from [11] approach of using Latent Dirichlet Allocation, which is very effectively used to identify keywords for a particular topic. Because of the nature of these topics being a latent feature of the corpus of text, it is difficult to attribute the latent topic to a class. Comprehensibility of the topics is a major concern. We aim to build features for the classifier that has comprehensible features. One interesting approach Ying et al. [4] takes is that of keeping number of topics equal to number of labels. This approach helps in comprehensibility.

- **RQ2:** How many topics should be specified for LDA?

Our motivation is to build a generic model that can do change message classification specific to a project. One interesting feature of building this automated keyword model for change message classification is that it can be generalized to other types of software text mining problems. So, we aim to verify generality of LocalMine on stackoverflow questions and technical debt datasets. These datasets comprise of natural language texts in software engineering domain as described in Chapter 7.

- **RQ3:** How does LocalMine generalize to different natural-language software texts, specifically stackoverflow and technical debt dataset?

Although, we focus mostly on change message classification but our core aim is to build a probabilistic keyword model that can be used for any type of natural-language software text mining. We establish that by answering this research question.

The paper proceeds as follows. In Chapter 2, earlier works in the space of change message classification are detailed. The paper proceeds with explaining the research methodology for LocalMine in Chapter 3. The metrics used for evaluating LocalMine and the baseline models are explained in Chapter 4. Chapter 5 deals with evaluation of the baseline models and corresponding useful data and graphs. Chapter 6 has the evaluation of LocalMine. Chapter 7 verifies the generality of

LocalMine with different natural language texts pertaining to software engineering domain. Chapter 8 delves into answers for research questions framed above. Chapter 9 discusses threats to validity and future work that can be extended from this keyword model. And Chapter 10 concludes on the results and performance of LocalMine

CHAPTER

2

RELATED WORK

In Chapter 1 we gave an introduction to what we are trying to achieve with LocalMine. In this chapter, we discuss earlier works in the same domain.

The research field of change classification has been evolving over many decades. The first classification system of software changes was put forward by Swanson in 1976 [12]. Swanson defined that a change can be classified in one of the following 3 categories:

- Corrective: Changes that are fix for prior failures.
- Adaptive: Changes in environment.
- Perfective: Changes to improve efficiency.

Based on this foundation, corrective software changes have been studied for the purpose of software maintenance [1] and defect prediction[9]. Mockus and Votta [1] established that to understand the motive of introducing a change in the software can be estimated from the associated message. Natural language processing comes into play when understanding these change messages.

Kim et al [9] mentions that a *bug-introducing* change can be identified through the analysis of a bug-fix. These bug fix changes are in turn identified by mining change messages. They propose that change message classification can be done through two ways:

- Keyword search model aimed at searching for well identified keywords such as "Fixed" or "Bug". This is a reasonable assumption and is also seen from Hindle et al. to be successful [3]. This sort of injection of domain knowledge into the classification of change message does prove helpful. Therefore, we use a similar model as our baseline.

- Searching for references for bug reports such as their IDs. It is very difficult to track as Issue tracking system is often used for Feature tracking too. These two data, generally, reside in two different systems i.e. version control and feature tracking. It seems that injecting more domain knowledge may help us identify bug-fixing commits more precisely.

These methods of identifying bug-fixing commits are costly. It is not easy to get version control data and tracking data together, as they are generally two separate systems. Moreover, using issue tracking system IDs does not resonate well with our main idea of simple automation of text mining model. A generic keyword based model does well for identifying the bug-fix commits, but it is not local to the project. Tracking issue IDs is local to a project but is not a scalable. We learn from this limitation and try to come up with an algorithms that is both automate, and can identify both general and local keywords corresponding to a bug-fix commit message.

Initial models were built on keyword basis [1] [2] [13] and Naive Bayes classifiers [14]. In a way, it can be said that these classifiers were either look up or needed help from experts for understanding classification. Antonio et al [14] proposed a method based on decision trees, naive bayes and logistic regression models that does change message classification based on external features, and not just commit messages. However, Support Vector Machine (SVM) remains a popular choice for this classification [11] [9]. Hindle et al. [3] employed the Weka Machine learning version of SVM. Random Forest have also been shown to be used successfully for classification task in software text mining [15]. We find Random Forest to be most robust to any type of data because of its ensemble nature. Hence, we use Random Forest in our experiments.

Mauczka et al. [6] proposed a variation of keyword based approach to automate classification of change messages. This model can be seen as a "Generalized evolving dictionary". The dictionary

evolves as more and more commits messages are studied. However, a stopping criteria needs to established. This is very specific to a project and has good performance, however the model is still rule based and manual intervention is required. Hindle et al. [11] discusses Latent Dirichlet Allocation (LDA), which is a more automated way of discovering keywords. Use of probabilistic model helps discovering more features and properties of natural-language text. In this particular case, it helps discover keywords. A dictionary model is not scalable as it always needs manual intervention, even when the keyword is developed specific to the project as modeled by Mauczka et al. [6]. Hindle et al [11] used LDA based method to analyze this kind of unstructured data i.e. raw texts, and then assign meaning to group of topics discovered. This is an attempt to provide comprehension to latent topics of LDA. LDA based text mining proves to be useful in increasing comprehension of the model as opposed to only TF (term frequency) and TFIDF (term frequency inverse document frequency) models [16]. However, assigning meaning to topics is still a manual process. LocalMine overcome this by taking inspiration from Ying et al [4] choice of keeping number of topics only corresponding to number of classes. In this way, the topics become comprehensible. However, to further utilise the topic for classification we require a classifier. Hence, Ying et al. [4] uses "signifier" documents to ascertain that documents generated from specific seed values. They try to take benefit of property of LDA to have an order bias. However, we do see that taking LDA to be the sole classifier does not yield us results significant enough compared to dictionary models. Therefore, we will see that in LocalMine, we resort to LDA to generate comprehensible features only and rely on Random Forest as a classifier for further classification utilising these features.

Another tool that has been studied and shown to have success is pLSA (probabilistic Latent Sematic Analysis) [5]. As established by Blei et al [17] LDA is an improvement upon pLSA, we use LDA for our study. Studies have shown the success of LDA models for defect prediction [18] [19] [16] [4]. Defect prediction, fault localization and automated program repair are few areas where understanding of bugs and bug-fixes can tremendously help improve efficiency of their applications. Success of these related works and applications of change message classification have motivated us to explore keyword modeling using probabilistic approaches, specifically LDA, for our study. LocalMine is, therefore, based on LDA and Random Forest.

CHAPTER

3

RESEARCH METHODOLOGY

LocalMine is based on raw textual data. To employ methods of LDA and Random Forest, we require to convert text to vectors. LocalMine makes use of a vectorization technique. In the next few sections of this chapter we will see how datasets and methods of this research have been set up. As you can see from Fig 3.1, first LocalMine processes raw textual data. After processing of the texts to convert them to Tf-idf vectors to feed to LDA. We extract top N (=25) words based on words in each topic and get K(=2) probabilities of topics for each commit message. These 2*N + K features are fed to a Random Forest classifier for binary classification.

3.1 Datasets

The data under study here is collected through mining of open source software projects from github.com. Three projects were selected for this study. These are popular github projects in the field of computation. They have a large number of commits, as will be explained later. The commit messages were labelled during a mechanical turks study. The mechanical turks were graduate students from the Computer Science department at North Carolina State University, Raleigh, NC,

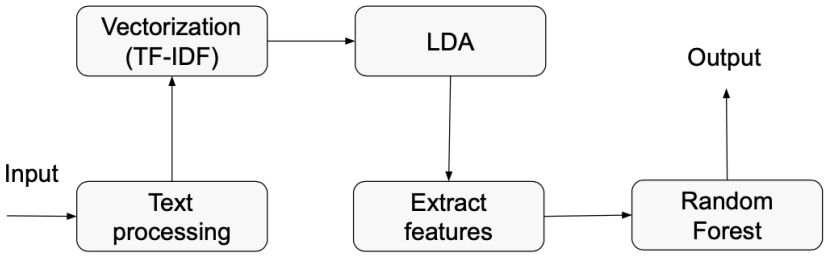

Figure 3.1 LocalMine Core Flow

Table 3.1 Commits dataset

Project	0: Not bug-fix	1: bug-fix	% of 1s	avg # tokens	vocab len
abinit	4024	572	12	5	3629
libmesh	6462	651	9	7	2798
mdanalysis	2802	379	12	7	2889

USA. The details of the project is shown in Table 3.1.

Each of the commit messages has been labelled as 0 or 1. Here, label 1 corresponds to a message being tagged as bug-fixing commit. We see that on average each project has approximately 90% non-buggy commits. This is expected. Otherwise, a software project is poorly managed if there are a lot of bug-fix commits. Also, we can see from Table-3.1, that on an average, we have buggy commits to be approximately 10 percent of the total commits. LocalMine first processes the commit message text by tokenizing and removing unnecessary tokens like stopwords and symbols like '/', '#', '-', etc. The number of tokens in each project is mentioned in 'avg # tokens' column in Table-3.1 and standard deviation of 4 tokens on average across project. This mean length is almost equally divided among both the classes. Maximum length of tokens in processed messages of a project goes high as 132. And minimum length is 1. We discard those message tokens that may not contain any terms after text processing. We also find that unique number of words in each project is as shown in 'vocab len' column of Table-3.1

3.2 Latent Dirichlet Allocation

Topic modeling algorithms are statistical methods that analyze the words of the original texts to discover the themes that run through them[17]. Topic modeling enables us to organize and summarize electronic archives at a scale that would be impossible by human annotation[17]. Specific to text based corpus, we can say that Latent Dirichlet Allocation (LDA) is an unsupervised clustering algorithm aimed at clustering the words in the corpus to specific *topics*. Since, it is probabilistic in nature, it means that certain words can reside in different topics. According to Blei et al. [17] the joint distribution of LDA can be described as

$$P(w,z|\alpha,\beta) = P(w|z,\beta) * P(z|\alpha)$$

Here, z is the topic and w is the word. And, α and β are model parameters. P(z|α) is the probability of topic z appearing in the document d. P(w|z, β) describes the probability of word w appearing in a particular topic z. We will make use of both P(z|α) and P(w|z, β). We use python scikit-learn implementation of LDA. There are certain hyper parameters to that model. *Number of topics* is the first and foremost hyper parameter. We need to decide how many topics prior to the training process. In our work, we classify change messages in two distinct values 0 or 1 indicating non-buggy or buggy commit message. Hence, our natural option for selecting *'k'* i.e. number of topics is 2. This ensures that the topic could correspond to these labels. This decision greatly enhances our ability to interpret these topics. This is great addition to comprehensibility of our model. This decision is inspired by a similar work of Ying et al on change message classification where they choose number of topics corresponding to number of classes [4]. Another hyper parameter is total number of words to consider before sending to LDA for training. This is called *maximum number of features*, i.e. the number of dimensions. This sets the maximum number of words to be fed to LDA. We see that lower number of maximum features is much better [20]. For this experiment we default to the value of 100 for this purpose. Another hyper parameter that we change in our experiment is *number of iterations* for LDA. We set it to 20. Essentially, LDA is a generative probabilistic model. LDA has two matrices. First, describes the probability of selecting a particular word when sampling a topic. Second, describes the probability of selecting a topic when sampling a particular document.

Now to build word versus topic matrix, a sample is drawn from dirichlet distribution for each topic using β as input. For second a composite is defined as a document and a sample is drawn from dirichlet distribution using α as input. Then the actual composites are defined basis these two matrix for each document. LDA does expectation maximization in each iteration. We set *maximum number of features* as 100 for our study. The LDA iteration method is Gibbs sampling. The other hyper parameters namely β, and that we choose are taken from literature study [4] or set to default.

LDA is a common technique in text mining used for dimensionality reduction [21]. TF and TF-IDF are common feature generation methods applied to text mining. The order of number of features is 1000s or more. However, with LDA we get very small number of dimensions. As we keep only 2 topics, we have 2 features corresponding to that, then we choose top words from these topics, 25 for LocalMine. So, this is a reduced dimension from 25+25+2 as opposed to 100s or 1000s in TF-IDF vectorization. The output of LDA is very comprehensible [21] [16]. We agree with their work and validate LDA and comprehensibility it offers in this paper.

3.3 Text Processing

Even before we feed data to LDA, we need to process the natural language expressed in change messages. Some examples of change messages are as follows:

- Quickfix: paral_kgb=1 shoulkd be ignored for DFPT (was OK in previous versions)
- Fix bug that was preventing the kxc to be stored.
- multibinit: add initialization for the fit and add restartxf -3,0.0.
- Fixup receive_packed_range call.

These messages are expressed in a natural language, not necessarily following the English grammar well. Most of the the times it contains only few words. We want to vectorize these words to feed into LocalMine's LDA. First of all we extract important words from this message. Using the python NLTK module, the words are first tokenized using RegExp Tokenizer. Then all the tokens which are either numerical values of symbols or punctuations are removed. To remove redundant tokens, all words that are English stopwords like "is", "are", "has", etc are removed. One caution is

taken that even if there remains any token of length 2 character or less it is removed. In addition, all tokens are stripped of white spaces and symbols like "-" and "_". Finally, the tokens are lemmatized. Lemmatization brings the natural language English word to its root form. For instance, "fixes" is reduced to "fix".

Above processing of words to a list of tokens gives us the corpus that we want the vectorizer to fit on. We use TF-IDF vectorizer. TF-IDF puts higher weight on word which are very frequent in a document and rarer across documents. This is to ensure that the words selected for training LDA are discriminatory of the documents. Maximum number of features is defined here as the length of our feature space for our experiments. We define it as 100.

3.4 Random Forest Classifier

Random forest (RF) is an ensemble approach that is specifically designed for decision tree classifier. Simply defined as collection of decision trees that are used for consensus on the decision. Each decision tree in RF is built on randomly selected subset of features. We control the depth and number of trees in the forest. For our case, we use python scikit-learn implementation of RF and we have set number of estimators to be 100 and max_depth of 100. Since, we have top 25 words and 2 topic probabilities, summing to 52 features, we believe this is a reasonable choice of hyperparameter. Moreover, we find that these parameters are values found by grid search over discrete values. The main benefits of RF are: 1) Feature importance is defined automatically, it removes the bias of pre-defined keywords and onus of curating important keywords for our case; 2) As RF is an ensemble of many different decision tree classifiers, it avoids overfitting.

CHAPTER

4

EVALUATION STRATEGY

In this chapter, we talk about the different metrics that are used to evaluate performance of LocalMine.

4.1 Classification metrics

Since this is a binary classifier, evaluating our results on the basis of confusion matrix seems apt. We will evaluate the performance of the model by precision, recall and F1 score. The definitions of these metrics are below:

- Precision: Defined as number of true labels as a fraction of total truly predicted labels, it says, number of true positives of all the positive predicted labels. Basically it indicates, of all positives that was predicted how many of them were actually positive. For precision score, the higher the better.

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

- Recall: Defined as number of predicted true labels as a fraction of total true labels, it says,

number of true positives of all the positive labels. Basically it means, of all the true labels how many did the model correctly recall. For recall score, the higher the better.

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

- F1-measure: F1 measure is the harmonic mean of precision and recall. For both precision and recall to be high, we need to have high F1 score.

$$\text{F1} = \frac{2 * \text{Precision} * \text{Precision}}{\text{Precision} + \text{Recall}}$$

Since, F1 is harmonic mean of precision and recall, we need not mention precision explicitly as it is implied. So, in some cases we have mentioned recall and F1 score only. We also report F2 score in some cases. F2 score is derived from general form of F1 score. F2 score puts higher weight on recall, i.e. cost of making False Negative is higher.

4.2 Statistical test

Effect size is necessary to be ascertained in any phenomenon. As popularly said, don't sweat small stuff. If the effect size is small, then the success of the experiment holds little value. For testing success of LocalMine, we will use Cohen's delta test. Cohen's d is defined as follows:

$$d = \frac{\bar{x_1} - \bar{x_2}}{s} = \frac{\mu_1 - \mu_2}{s}$$

Here, $\bar{x_1}$ and $\bar{x_1}$ refer to mean of LocalMine and baseline, respectively. And, s is the standard deviation of LocalMine. We will look at the precision, recall and f1 and test with respect to Cohen's delta. We define small effect corresponding to Cohen's convention i.e. 0.2 as small effect size. If any result corresponds to small effect size, it will be mentioned with a (*) star in the tables. Cohen's conventional criteria that defines an effect size i.e. measure of magnitude of phenomenon, as *small)*, *medium* or *big*, is acceptable in most studies. However, these are only rule of thumb and should be used for understanding only in the context of the experiment.

CHAPTER

5

EVALUATION OF BASELINE MODELS

In this chapter, we will discuss about the baseline models. We have chosen two models that are dictionary models and two models that can be automated and are probabilistic models. One interesting thing to note is that we have implemented the model in python by carefully understanding the algorithm and parameters provided in respective papers. Then we evaluate the models on the commit messages dataset to understand their true baseline metrics in regards to our experiment of LocalMine on commit messages dataset.

5.1 Baseline Models

As discussed in earlier chapters we have some keyword dictionary lookup models and some probabilistic models that have shown to perform well for change message classification. We discuss four models proposed in literature. We argue for two of them to be contestants for baseline against which we will measure LocalMine.

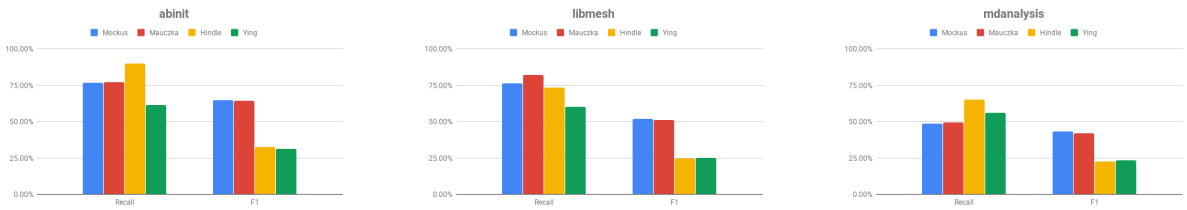

Figure 5.1 Evaluation of the baseline models on different data-sets

Table 5.1 Mockus et al. [1] dictionary

Category	Associated words
Corrective	fix, bug, problem, incorrect, correct, error, fixup, fail
Adaptive	new, change, patch, add, modify, update
Perfective	style, move, removal, cleanup, unneeded, rework

Table 5.2 Mauczka et al. [6] dictionary

Category	Associated words
Corrective	wrong, except, cause, null, warn, correct, valid, fail, bug, dump, opps, error, invalid, failure, incorrect, bugfix, fix, problem, bad, miss
Adaptive	context, introduce, future, appropriate, information, add, compatibility, simple, structure, configuration, available, require, init, function, internal, security, current, config, provide, easier, switch, default, faster, text, release, install, user, feature, old, useful, method, change, trunk, protocol, patch, version, replace, necessary, new, create
Perfective	clean, include, whitespace, dead, consistent, prototype, static, style, remove, definition, header, documentation, variable, inefficient, useless, cleanup, move, unused, declaration
Blacklist	cvs2svn, cvs, svn

5.1.1 Fixed Dictionary

Mockus et al. [1] studied and created classification based on word frequency analysis and keyword clustering. Mockus et al. derives inspiration from Swanson's classifications [12]. They have created a generic keyword based dictionary that corresponds to Swanson's classes [12]. It has been envisioned that there are 3 primary types of maintenance: fault fixes for keywords such as *fix, problem, incorrect, correct*, new code development for keywords *add, new, modify, update* and code improvement for keywords *cleanup, unneeded, remove, rework*. These keywords help in clustering of similar words based on some heuristics, followed by a simple classification algorithm. The keywords used in our implementation of the baseline is shown in Table-5.1. A developer survey has been used

as validation method in the study. Kappa coefficient is 0.5 indicating moderate agreement. A log linear model was employed to further this validation that indicated strong agreement between automatic classification and developer's classification. A claim is made that this method can be used for different projects as only basic information from version control system is used. This is a very simple algorithm and very comprehensible. It can be easily reproduced. So, it is a good candidate as baseline for our project.

5.1.2 Generalized Evolving Dictionary

Mauczka et al. [6] proposed a variation of dictionary based approach to automate classification of change messages. This work has been based on Mockus et al. [1] natural language "modification request" words. Weights for certain keywords were also defined. Since, the weight was subject to researchers tuning of their algorithms, it may have suffered a bias. A first set of 3 keywords is for each category is selected based on prior work [13] [1]. This keyword dictionary is extended by new words basis a frequency analysis algorithm until 80% of the project is classified. This dictionary was localized to a project starting from an initial globally accepted seed. A modified version of the original dictionary is shown in Table-5.2. However, they go on to do validation over cross-projects to establish a general keyword dictionary. Mauczka et al. [6] made an extension of Swanson's original classification by adding a *Blacklist* category. This dictionary can be seen as an extension of exemplary work of Mockus et al. [1]. Moreover, it has been validated across different projects. Hence, this model can also be a viable candidate for baseline.

5.1.3 Latent Dirichlet Allocation with Manual Annotation of Topics

Hindle et al. [11] discusses Latent Dirichlet Allocation (LDA). The proposal says discovering change message classification topics over a window of time. LDA is a mixture model based method used to analyze unstructured data. Hindle et al. [11] used certain parameters to conclude topic meaning. Top 10 words were seen among all 20 topics. Then they were manually assigned a topic name. If some topic did not give proper meaning it was discarded. The main objective is to create a trend in topics for a software project. For this purpose, similarity between two topics needs to be computed. A simple algorithm of 8 matches among top 10 words implied similar topic. There are many "magic" parameters used in this study. Nevertheless, as the generation of topic from commit message is an

automated process. The final classification of a topic is based on human intervention. Hence, it is kind of semi-supervised classification based on LDA with software domain knowledge injection. Such a model is viable candidate for our baseline.

5.1.4 Semi-supervised Latent Dirichlet Allocation

Ying et al. [4] proposes an automated discovery of change message features using LDA. The algorithm includes a semi-supervised LDA method for change message classification. It is called semi-supervised as topic generation and data transformation through LDA is largely depended on *signifier documents*. A set of documents is built as signifier documents corresponding to Mauczka et al. [6] extended dictionary that is validated across project. The exact dictionary used is shown in Table-5.2. In a way, Ying et al. method is initialized using Mockus et al. [1] "keywords" and is based on Swanson's original change message classification [12]. Key difference of this algorithm is two folds:

- Injection of software engineering domain knowledge to generate topics and words distribution.
- Classification algorithm is similarity between topic distribution of signifier documents and target documents.

Validation is done through developer survey and accuracy is reported. Approximately, 85% of the messages were classified with approximately 70% accuracy. The algorithm is claimed to work on cross project analysis of software change message classification because of developer agreement. It's dependence on both software domain knowledge and automated change message classification with features specific to a project makes it a good candidate for baseline.

5.2 Comparing Baseline Models

For the purpose of establishing a baseline, a version of each of the above is implemented by us. We sample the data with 20 times and report the mean results as shown in Figure-5.1 and in Table 5.3.

The model for Mockus et al. [1] is implemented as a look-up dictionary as in Table-5.1. If any keyword is present in the change message tokens, it is classified according to the category. Similar

Table 5.3 Evaluation metrics of baseline models

	Models	Recall	F1
abinit	Mockus et al.	77%	65%
	Mauczka et al.	77%	64%
	Hindle et al.	90%	32%
	Ying et al.	61%	31%
libmesh	Mockus et al.	76%	52%
	Mauczka et al.	82%	51%
	Hindle et al.	74%	25%
	Ying et al.	60%	25%
mdanalysis	Mockus et al.	49%	43%
	Mauczka et al.	49%	42%
	Hindle et al.	65%	23%
	Ying et al.	56%	23%

implementation is for Mauczka et al. [6]. The dictionary look-up is mentioned in Table-5.2. This is close to actual algorithm implemented in the paper and not exactly same, as in the paper there is reliance on manual intervention for establishing classification. Our implementation of the models has no human intervention. However, this is based on the final dictionary proposed for cross project change message analysis by Mauczka et al [6]. The algorithm proposed by Ying et al. [4] is implemented exactly as mentioned. The signifier documents used are as in Table-5.2. Ying et al.[4] model is semi-supervised and is classified basis similarity to topic distributions in signifier documents using cosine similarity. Also, we take inspiration of using LDA as language model as mentioned in Hindle et al. [11]. A similar model is implemented. Instead of manual annotation of a topic classification, we generated only 2 topics and use *Corrective* keywords from Table-5.1 to annotate one of the topics as "corrective" topic. After a topic is annotated as buggy based on analysis of top 10 words with the dictionary, we classify the change message according to the probability of that topic.

We compare all the three software projects under study on the implementation of the baseline models as described above. Although, Mockus et al. [1] and Mauczka et al. [6] are not automated models but are dictionary lookup, they still tend to perform equally good as the other models for the given dataset. One other thing to note is that all the models based on "corrective keywords" is better. That is to say, precision is low for all these models as they have been specifically designed for

bug-fix indicating words. If we say that we require an automated language model rather a dictionary based model, Hindle et al. [11] model is the most acceptable baseline. Otherwise, our baseline is Mauczka et al. [6] keyword based model. So, we will evaluate our model, LocalMine, against both these baselines.

CHAPTER

6

LOCALMINE ALGORITHM AND EVALUATION

In this chapter, we talk about the main innovation of this thesis, the probabilistic keyword model i.e. LocalMine. The algorithm of LocalMine is novel and inspired by the work of Ying et al [4]. LocalMine uses LDA for feature generation, essentially keyword generation. These features are fed into a binary classifier, Random Forest, for final prediction. We explain the algorithm for LocalMine model building and evaluation on commits messages dataset in this chapter.

6.1 Algorithm

The text processed data is split into 80% training and 20% testing. From the training data create a random dataset of 2:1 label counts i.e. one-third of data should be 1s and two-thirds as 0s and drop other data points. Shuffle the training data. Using the TfidfVectorizer of python's scikit-learn, the training set is vectorized and top 100 features are sent to LDA for building topic and word

distributions. LDA is used to select features (words) that matter the most in this classification. We select top 25 words from each LDA topics. We also take the topic probabilities of each commit message.

First, after text processing and train/test split, we go through LDA transformation to get both probabilities of topics and probabilities of feature words. Let's call the two topic probabilities as *Topic_0* and *Topic_1*. Table-6.2 describes the top 10 words found in each topic according to the probabilities of occurrence in the topic. It is quite evident that *Topic_1* corresponds to a topic that describes bug fix words.

After LDA transformation, we go through the following algorithm:

1. Compute the probability of each word in the topic and store in a `word_topic_map`
2. Go through each document.
    - Iterate over all the initial tokens present.
    - If a token matches the featured words, add the probability (from `word_topic_map`) of it being in a topic.
    - If the word is in both topics, use the maximum probability.
    - Along with these 50 features, either 0 or otherwise computed in previous step, append the topic probabilities for the document as last two features i.e. total 52 features.
3. Split train data into 80% and 20% for training and validation.
4. Feed these 52 columns to Random Forest(max_depth=100, n_estimators=100) for training.
5. Test on testing set.
6. Repeat the entire training process 20 times
7. Run Cohen's delta test to assess for small effect size.

6.2 Evaluation Metrics

All the above steps including initial train/test split, training LDA and training RF is done 20 times to remove sampling bias. Finally average evaluation metrics are collected as shown in Table 6.1.

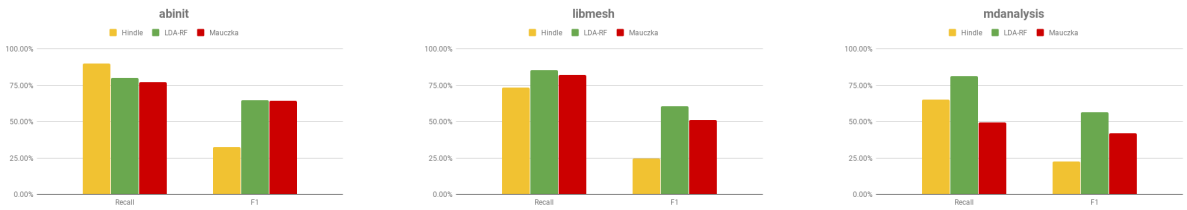

Figure 6.1 Evaluation of LocalMine with baseline

Table 6.1 Evaluation metrics of LocalMine (** indicates Cohen's small effect size)

project	precision	recall	f1	f2
abinit	54%	80%	65%	73%
libmesh	47%**	86%	61%	74%
mdanalysis	43%**	81%	57%	69%

Table 6.2 LocalMine Topics - Top 10 keywords per topic

project	topic 0	topic 1
abinit	merge, branch, develop, abinit, trunk, remote, gitlab, initialize, abirules, routine	fix, error, file, update, add, test, compilation error, minor fix, correction, problem
libmesh	merge, request, pull, libmesh, mesh, update citation, refactor, pull request	fixed, warning, bugfix, patch, bug, change, enable, removed, disable, update citation, refactor
mdanalysis	merge, doc, mdanalysis, update, pull, request, version, change, release, issue	fix, test, issue, develop, added, updated, removed, code, mdanalysis, error

Performance of LocalMine model with corresponding baseline model i.e. Hindle et al and Mauckza et al is show in Fig. 6.1. Table 6.2 shows the top words corresponding to the topics generated by LDA.

Looking at Table 6.2 we can easily point the topic corresponding to bug-fix, which Topic 1 in our case. This comprehensibility is at par with comprehensibility offered by dictionary based models. Moreover, the metrics of LocalMine, in terms of recall and F scores, outperform the baseline models. It is also worth noting that LocalMine used the vocabulary mentioned in 6.3 for generating above topic word distribution. The vocabulary is based on TF-IDF vectorization of the text corpus under study i.e. commit messages. To come up with the final model we also do hyper-parameter optimization over discrete values of various "magic" parameters involved in our study. The parameters mentioned throughout the thesis are the most optimized ones.

Table 6.3 LocalMine Vocabulary

project	vocabulary
abinit	abilint, abinit, abinit develop, abinit org, abirules, add, also, beauty, branch, branch develop, branch trunk, bug, bug fix, buildsys, case, change, compilation, compilation error, correct, correction, create, define, develop, develop gitlab, dfpt, doc, error, file, fix, fix bug, fix compilation, fix compilation error, fix minor, fix problem, fix typo, fixed, gitlab, gitlab abinit, gitlab abinit org, gitlab trunk, gitlab trunk abinit, gmatteo, gnu, hotfix, html, improve, input, input variable, input web, intel, issue, lesson, libxc, link, make, merge, merge branch, merge branch develop, merge remote, merge remote tracking, minor, minor fix, missing, modified, multibinit, new, new file, nonlinear, one, org, output, paw, problem, ref, ref file, reference, release, remote, remote tracking, remote tracking branch, remove, small, suppress, test, tolerance, topic, tracking, tracking branch, tracking branch trunk, trunk, trunk abinit, trunk develop, tutorial, typo, update, update ref, use, variable, version, web
libmesh	add, added, adding, also, amr, assert, boundary, branch, bug, bugfix, build, case, change, changed, citation, class, code, comment, complex, constraint, data, default, dependency, disable, dof, elem, element, enable, enable complex, error, example, file, fix, fix disable, fix enable, fixed, fixing, function, get, header, include, libmesh, make, map, merge, merge branch, merge pull, merge pull request, mesh, method, mode, mpi, need, neighbor, new, node, non, one, option, order, output, parallel, parallelmesh, patch, petsc, point, pull, pull request, read, refactor, remove, removed, request, return, rule, run, set, side, solution, std, still, support, system, test, time, type, unused, unused variable, update, update citation, updated, use, user, variable, variable warning, vector, warning, work, write, wshadow
mdanalysis	add, added, also, analysis, atom, atomgroup, bond, branch, branch develop, bug, case, change, changed, changelog, check, close, code, code google, code google com, code mdanalysis, com, com mdanalysis, coordinate, core, corresponds, corresponds code, corresponds code mdanalysis, data, dcd, default, develop, distance, doc, documentation, error, example, feature, file, fix, fix issue, fixed, fixed issue, format, frame, function, google, google com, google com mdanalysis, html, http, http code, http code google, import, issue, link, list, make, mdanalysis, mdanalysis develop, merge, merge branch, merge branch develop, merge pull, merge pull request, method, minor, module, new, numpy, pdb, pull, pull request, pull request mdanalysis, reader, regenerated, release, removed, request, request mdanalysis, rest, return, selection, setup, sphinx, test, test case, timestep, topology, trajectory, trr, type, unit, universe, update, updated, use, version, warning, writer, xtc

CHAPTER

7

GENERALITY OF LOCALMINE

To establish generality and robustness of our model we evaluate the performance of LocalMine model on two different datasets. These datasets are natural-language software texts. The datasets have been generously provided by Real-world Artifical Intelligence for Software Engineering research lab in Department of Computer Science at NC State. In this chapter, we will experiment LocalMine with different types on natural language texts. The first dataset is Stack Overflow dataset. This dataset primarily has questions. The content is mostly natural language and is specific to a domain. Most of the questions can be categorised by a few or more tags. The second dataset is Technical Debt dataset. Technical debts mostly comprise of self admitted errors that were introduced because of a reason to implement quick or temporary fix. These natural language comments are more technical in nature and has more similarity with change messages.

Table 7.1 StackOverflow.com dataset

project	Counts
academia	8081
cs	9270
diy	15287
expressionengine	9469
judaism	14393
photo	12733
rpg	11208
scifi	19546
ux	13424
webmasters	17911
total	131322

7.1 Datasets

7.1.1 Stack Overflow dataset

Stack overflow dataset is collected from stackoverflow.com. The data contains questions and tags associated with them. We specifically focus on 10 classes as shown in Table 7.1 created by analyzing associated tags. This table shows the amount of data (number of stack overflow questions) we have in each category. While using this data we will mark one class as 1 and rest as 0 and sample desired amount of rows from this modified dataset. An example of such a data is as follows: ***engineering****: "i was just wondering how a diesel engine starts given that there are no spark plugs how does it build up momentum for that first explosion in the cylinder does it have something to do with the starter motor"*.

7.1.2 Technical Debt Dataset

Technical Debt dataset is extracted from comments in the code of 10 open source projects as shown in Table 7.2. Each of the comments have been associated with one of the classes namely: defect, design, documentation, implementation, test or without any classification. For the purposes of testing the generality of our model, we classify each of the technical details as 1 and the rows without classification as 0. An example of such a data is as follows: ***apache-ant-1.7.0 : DEFECT****: // FIXME formatters are not thread-safe*

Table 7.2 Technical Debt Dataset

project	DEFECT	DESIGN	DOCUME-NTATION	IMPLEME-NTATION	TEST	WITHOUT_-CLASSI-FICA-TION
apache-ant-1.7.0	13	95	0	13	10	3967
apache-jmeter-2.10	22	316	3	21	12	7683
argouml	127	801	30	411	44	8039
columba-1.4-src	13	126	16	43	6	6264
emf-2.4.1	8	78	16	2	0	4286
hibernate-distribution-3.3.2.GA	52	355	1	64	0	2496
jEdit-4.2	43	196	0	14	3	10066
jfreechart-1.0.19	9	184	0	15	1	4199
jruby-1.4.0	161	343	2	110	6	4275
sql12	24	209	2	50	1	6929

7.2 Evaluation

LocalMine has claims to create a generic language model that learns specific local features of a specific project. This translated into the fact that LocalMine when modeled as a binary classifier should efficiently capture the features of the natural language. We used the same metrics we have used to evaluate this language model on github commits dataset i.e F1 and Recall. However, we also report precision and F2 score here. These are only for references and verifying correctness of the model.

7.2.1 StackOverflow

We have 10 classes in this dataset as shown in Table 7.1. We will do all vs one classification here. For creating the dataset we follow following steps for one class at a time:

- Label all data for a class as 1 and others as 0, and shuffle the data.

Table 7.3 StackOverflow Dataset Metrics

project	precision	recall	f1	f2
academia	64%	95%	77%	87%
cs	59%	93%	72%	83%
diy	86%	93%	89%	92%
expressionengine	77%	90%	83%	87%
judaism	67%	80%	73%	77%
photo	83%	94%	88%	92%
rpg	68%	92%	78%	86%
scifi	79%	80%	80%	80%
ux	62%	86%	72%	79%
webmasters	75%	87%	81%	84%

- Split the data randomly in 80:20 for train and test, respectively.
- From the training data create a random dataset of 2:1 label counts i.e. one-third of data should be 1s and two-thirds as 0s and drop other data points. Shuffle the training data.
- Split training data in training and validation 80:20.
- Train on training data, validate on validation.
- Test on testing data and record metrics
- Repeat above steps 20 times and report average metrics.

Recall and F1 for StackOverflow dataset is as shown in Fig. 7.1. We see that on average we have approximately 89% recall and 80% F1 score. We find that average precision is less that brings down the F1 score. Depending on the use case we can assign relative importance to recall or not. Therefore, we have included precision and F2 score in Table 7.1. In particular, it can be seen that this model consistently has better recall than precision which was evident from github commits dataset too. We can see from the Table 7.4 that the classes with high recall and F1 produce much coherent topics (Topic 1 corresponds to topic related to the category). Hence, we establish that LocalMine is generic enough to learn features of natural language in texts like stackoverflow, which is loosely SE based texts.

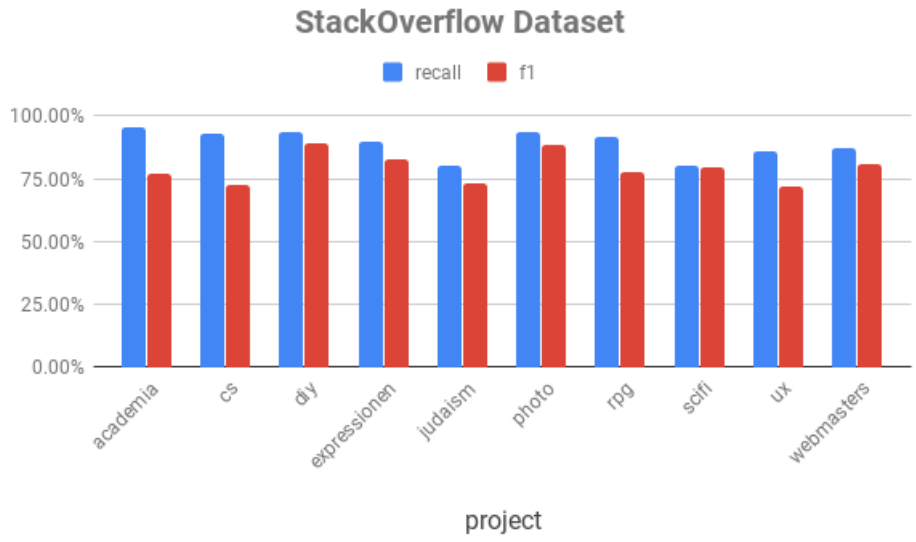

Figure 7.1 Performance on Stack Overflow dataset

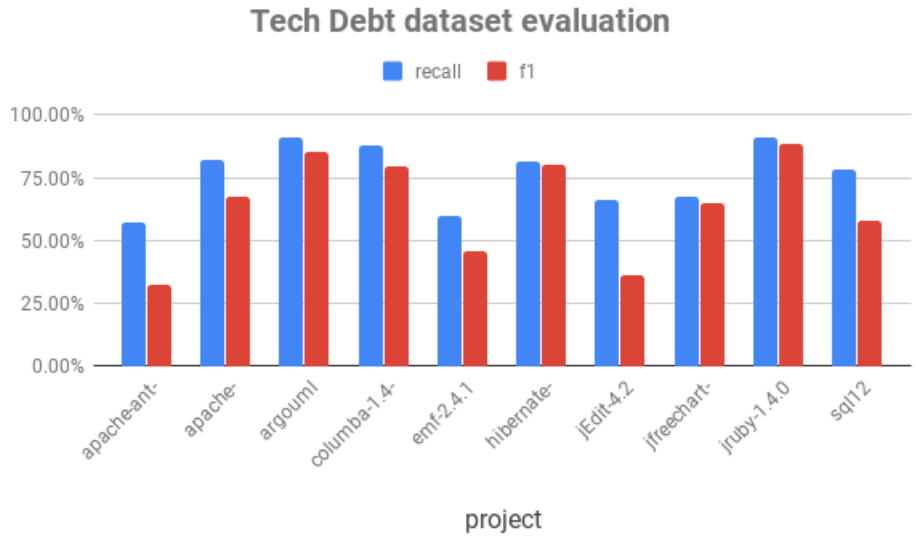

Figure 7.2 Performance on Technical Debt datasets

7.2.2 Technical Debt

Technical Debt data is merely comments from the code-base of the project. These comments are self admitted error, introduced due to rushed temporary code introductions. We use the same metrics as above i.e. F1 and recall to gauge the performance of LocalMine. The evaluation graph is as shown in Fig. 7.2. We see that recall is consistently high for all the projects than in baseline model. Table 7.6

Table 7.4 StackOverflow Topics - Top 10 keywords per topic

project	topic 0	topic 1
academia	use, page, site, like, user, would, need, way, set, one, websit, googl, look, imag	paper, research, one, student, would, work, univers, year, phd, question, time, know, ask, get, book
cs	use, page, site, user, like, would, com, work, imag, want, get, look, googl, websit, name	problem, one, time, would, algorithm, languag, number, question, find, know, say
diy	use, one, would, like, page, know, site, user, time, question, say, name, exampl, com, differ	water, wall, light, wire, hous, use, would, need, floor, instal, one, like, switch, replac
expressionengine	one, would, use, like, know, time, question, make, say, think, look, could	entri, page, site, field, channel, user, use, form, exp, file, categori, name
judaism	use, page, site, user, like, want, work, would, set, get, com, look, need	one, say, would, know, sourc, time, question, peopl, day, answer, make, torah, read
photo	page, use, site, user, name, com, file, one, would, websit, like, work, exampl, googl, question	camera, len, use, photo, would, one, like, get, light, take, canon, look, imag, pictur
rpg	use, page, site, user, name, would, like, work, imag, websit, com, googl, one, know, problem	would, charact, one, use, level, player, like, attack, spell, game, make, get, rule, power, book
scifi	use, page, user, site, would, like, need, work, want, problem, get, com, way, imag, websit	one, book, would, time, know, read, say, like, year, seem, stori, rememb, peopl, question, power, charact
ux	one, would, use, know, time, say, like, get, question, make, seem, read, take, look	user, page, use, site, design, button, websit, like, search, form, list, com, field, exampl
webmasters	one, would, use, time, like, know, question, say, make, get, seem, look, could, take, think	site, page, com, googl, websit, user, use, domain, link, http, search, file, content, imag, url

shows the words associated each of the topic. In 'jEdit-4.2' project we see a lot of overlap between these words i.e. technical terms. We can attribute this fact for this project to have one of the lower recall and F1 scores. This fact is also evident from the 'emf-2.4.1' and 'apache-ant-1.7.0' project. Also, it is worthy to note that the average recall is 76% and F1 is 66%. Average recall as established in the baseline model Huang et al [22] is 72% and average F1 is 59%. Although, both our approach and Huang et al. [22] approach is an approach that generalizes to any project. Huang et al. [22] used a Naive Bayes classifier on top of words transformed to a Vector Space Model. LocalMine uses a

Table 7.5 Technical Debt Dataset Metrics

project	precision	recall	f1	f2
apache-ant-1.7.0	23%	57%	32%	43%
apache-jmeter-2.10	57%	82%	67%	75%
argouml	81%	91%	86%	89%
columba-1.4-src	74%	88%	80%	84%
emf-2.4.1	38%	60%	46%	53%
hibernate-distribution-3.3.2.GA	79%	82%	80%	81%
jEdit-4.2	26%	66%	36%	48%
jfreechart-1.0.19	64%	67%	65%	66%
jruby-1.4.0	86%	91%	88%	90%
sql12	47%	78%	58%	68%

probabilistic framework by using LDA built on top of TF-IDF vector and the features from LDA are used to train a Random Forest classifier.

Table 7.6 Technical Debt Topics - Top 10 keywords per topic

project	topic 0	topic 1
apache-ant-1.7.0	file, ignore, attribute, property, set, directory, specified, create, need, name	checkstyle, ignore, visibilitymodifier, class, default, method, end, get, test
apache-jmeter-2.10	todo, used, set, test, file, check, default, use, name	nls, non nls, noop, panel, main, org, apache, see, apache jmeter, main panel
argouml	todo, class, end, set, needed, name, element, model, add, nothing	see, org, argouml, uml, object, java, lang, lang object,
columba-1.4-src	nls, non nls, message, folder, add, todo, create, get, new, author	columba, text, org, non javadoc, javadoc see, event, update, tooltip,
emf-2.4.1	value, byte, fill, object, non nls, nls, copied, non, interface, javadoc copied, javadoc	ignore, create, feature, nothing, command, class, add, file, new, content
hibernate-distribution-3.3.2.GA	todo, cache, check, add, column, alias, new, sql, object, use, one, node, key, note, method	collection, type, return, null, element, name, join, support, property, table, first, handle, select, array
jEdit-4.2	constructor, variable, type, line, case, instance, value, buffer, need, check, object, note, instance variable	method, class, member, private, private member, create, init, init method, find, package, workaround, inner class, inner
jfreechart-1.0.19	draw, argument, defer, defer argument, checking, argument checking, test, nothing, todo, paint	check, series, add, item, data, point, value, case, try, plot, independence, set, null, axis, fixme, check independence
jruby-1.4.0	line, fixme, switch, set, thread, constant, new, add, receiver, module, ignore, true, create, error, node, exception	method, value, args, class, block, scope, ruby, string, mri, array, name, variable, case, java, call
sql12	sql, error, todo, see, null, session, non, class, row, user, squirrel, oracle, nothing, javadoc, test, non javadoc	table, file, column, data, name, type, use, new, object, set, need, method, value, add, create, graph, text, expected, read

CHAPTER

8

SUMMARY

The research questions that we formulated in the introduction are answered in this chapter.

RQ1: *Does an automated probabilistic model perform better than generic dictionary based classifiers?*

Yes. As it is evident from Fig. 6.1, LocalMine outperforms Mauckza et al. [6] keywords based model in all projects. Moreover, LocalMine has the best F1-score and recall score on averag across all projects. LocalMine does this well because it captures local features of a project that are characteristic of the project. Only, in one project, 'abinit', Hindle et al. probabilistic and general model has better recall than LocalMine. However, in reaching such high recall this model has lots of false positive. Both Hindle et al model and LocalMine does better than keyword based model in terms of recall. LocalMine has smart feature generation, therefore goes on to have better F1 score too. It can be also seen from Fig. 5.1 that keyword based models perform better than Hindle et al model in 'libmesh' and 'mdanalysis' project. These models do have a better F1 score than all other models. It is interesting to note that project specific evolving dictionary Mauczka et al. model performs better than generic dictionary. Hindle et al constantly has comparable or greater recall for all three projects. Moreover,

Hindle et al model is automated and can be project specific. Both these conclusions points to the fact being project specific reduces a lot of false negatives. However, it also leads to a rise in false positive in case of Hindle. We tackle this problem by doing project level feature engineering by choosing LDA transformed features. We see that from Fig. 6.1 that LocalMine outperform Hindle et al. F1 score is also higher indicating that false positive has also decreased. We can also see from Table 6.2 that it is not necessary that only those words have higher weight that are mention in dictionary models like in Table 5.1 or Table 5.2. For instance, 'compilation error' is a two word phrase that is characteristic of a bug-fixing commit message. The dictionary based model do not account for phrases. Even if they do, it will become unfeasible to determine common phrases for different projects. Hence, we should not specify keywords that do not localize to a project as they do not capture important information that is a characteristic of the project for change message classification.

RQ2: *How many topics should be specified for LDA?*

Inspired by Ying et al [4] we decided to keep number of topics to be equal to number of classes (=2 for the case of LocalMine). From Table-6.2, we can observe that topic 1 actually capture the words corresponding bug fixes. The keywords discovered by LocalMine are also mentioned in Mockus et al [1] dictionary. Therefore, keeping 2 topics, fetches keywords that are comparable to special keywords that are prevalent in the literature, along with many keywords that are unique to commit messages of bug fixes for that project. Moreover, we note that apart from *Keyword search* model, no other model offers this comprehensibility other than LocalMine. In fact, LocalMine offers a better comprehensibility than *Keyword search* model. It is such because, LocalMine identifies keywords that are local to the software project. Not only does LocalMine capture localized keywords, but it also captures keywords that are similar to corrective category keywords used in *Keyword search* model. It can be seen in Table 6.2. Hindle et al. [11] used LDA but number of topics was arbitrary. Nonsensical topics were later weeded out by human intervention. However, LocalMine sets number of topics equal to number of classes. The objective that LocalMine has that each topic should provide with enough corresponding features in order to distinguish the two classes. When we look at Table 6.2, we can find that the two topics are distinctly different. Moreover, many of the top words match to the generic keywords given by Mauckza et al. [6] and Mockus et al. [1]. So, we can conclude keeping topics corresponding to classes helps in comprehensibility.

RQ3: *How does the model generalize to different natural-language software texts?*

We verified the generality of LocalMine by testing on two different software texts. On the stackoverflow set we find that average recall across projects is 89% and F1 as 80%. It is very comparable to baseline proposed in Majumder et al [23] implementation of baseline K-Nearest Neighbour model. Based on this comparison it can be claimed that LocalMine generalizes to stackoverflow dataset. However, LocalMine does require various hyper-parameter tuning. Comparing against the baseline on the tech debt dataset, LocalMine has outperformed the baseline in terms of recall and F1 score. We see from the table 7.6 that the projects with lower F1 score have higher correlated top words in the topics. In other words, those projects have correlated topics. Other projects where LocalMine generated distinct top words, i.e. LDA based feature generation is in tandem with the features required for classification, we have high recall and F1, and higher than the baseline Huang et al. [22] model. Hence, LocalMine generalizes to different kinds of natural-language software texts.

CHAPTER

9

THREATS TO VALIDITY AND FUTURE WORK

In this chapter we discuss some of the limitations for LocalMine. Also, we discuss how these limitations may be reduced and some enhancements that LocalMine can incorporate as part of future work.

9.1 Threats To Validity

It is quite evident from this study that LDA helps in comprehensibility of the model. However, the complexity of the model increases much more than simple dictionary models. This raises doubt if model is within reasonable limits. However, we see that the model is fairly robust across the projects and software texts. We can, thus, say construction of LocalMine increases complexity and increases the cost of other resources like time to train the model and computing requirements grow as the size of project increases. However, the comprehensibility increases as we are able to associate each topic

to one of the classes and able to justify the performance of LocalMine by understanding keywords like in dictionary models.

The results reported are generated over 20 sample runs. This is done to reduce the order bias. As data sets distributed in training and testing is completely random, that could have incurred order bias. Agrawal et al. [21] discussed that LDA, inherently, has an order bias. The algorithm automatically discovers words that are characteristic to a topic. These words could differ in different samples of data. However, keeping low number of maximum features for LDA training and running this experiment multiple times helps reduce that bias.

LocalMine decides 100 words from TF-IDF vectorized model to feed to LDA. This also means that a large portions of words in initial commit messages are dropped. As we can see from Table-3.1, approximately 3000 words are present in the vocabulary (after text processing) of commits message dataset per project. That means approximately 97% of words are not considered for final classification. There could possibly be some information loss here. However, through grid search over discrete values of 100, 500 and 1000 and LocalMine found 100 to be most optimal value with regards to the evaluation metrics.

Instead of relying on special words provided by Mockus et al [1], LocalMine discovers keywords that may better characterize changes messages of the project. After LDA, LocalMine fetches top N discovered keywords for each topic (N = 25, found after hyper-parameter optimization). The probability values of these words corresponding a topic are used to generate features when a word in the tokens of a commit message matches it exactly. This exact match could be misleading as a typographical error would not account for it or even a synonym. The match here could have been in terms of natural language, slopped to adjust for spelling variation and synonyms could be used keeping software engineering domain knowledge. Accommodating this domain knowledge may have an effect on LocalMine's keyword model.

LocalMine hyper-parameter optimization has been done over discrete finite space. Number of features for TF-IDF vectorizer was varied over the set of (100, 500, 1000). Number of words extracted as features per topic was varied over the set (10, 25, 50, 100). Random forest parameter for n_estimators and max_depth were varied over (50, 100, 200) and (10, 50, 100), respectively. A differential evolution technique may prove more useful here where the variation can be over larger space.

9.2 Future Work

Injecting Software Engineering domain knowledge into LDA and feature generation may perform better. As described in limitations section, feature generation using matching words can make use of SE domain knowledge.

The hyper-parameters of LocalMine were optimized with Grid Search. However, Agrawal et al. [16] shows that using differential evolution techniques, LDA has much better performance. We can also optimize various parameters in LocalMine apart from LDA, namely number of words to be used as features, number of estimators in Random Forest, maximum depth of Random Forest, etc.

There are other word embeddings models that can be employed. As commit messages are based on semantic natural language sentences, it might help to use word2vec or GloVe embedding instead of TF-IDF vectorization. Word2vec is known to capture semantic and syntactic relationships at low computational cost [24]. GloVe is unsupervised learning algorithm to obtain word vector representations [25]. However, this may lead to increased complexity in comprehensibility of the model. If these models improve our evaluation metrics, further questions can be raised on the comprehensibility and reason-ability of these models.

LocalMine is localized to a software project, and is yet a general model. It is a common practice in software engineering data mining that the model is trained on earlier releases of the software project and tested on more recent version. Can we train a model on older versions of project and predict for current version? However, a big task in change message classification is labeling of messages. Kim et al [9] suggests understanding bug tracking system and use keyword search heuristic to classify. However, recommendation is still made to get these change classifications manually verified. Software engineering domain knowledge is essential, but sample change message classification for each project is costly. Can we omit human in the loop and understand from earlier release notes for keyword labeling in a semi-supervised fashion? Or can a fully automated change message classification be built?

CHAPTER

10

CONCLUSIONS

In previous chapters, we have seen that LocalMine outperforms baseline models, both dictionary based models and probabilistic models. LocalMine is, also, suitable to be used for different software text mining task. In this chapter, we talk about the conclusions that can be drawn from experiments that we did for LocalMine and the results obtained. LocalMine is an automated model that generalizes on a variety of datasets. We see that LocalMine which is an LDA-based classifier works with github commit messages for change message classification, also with stack overflow dataset for topic classification and tech debt for comments classification. Though, we should note to achieve this performance on stack overflow and tech debt dataset, we had to tune some hyper-parameters associated with the components of LocalMine. We have kept number of features for TF-IDF as 1000 and top N words from topics to be 100. In change message classification model, these values were 100 and 25 respectively. We find that a small amount of tuning is required to create project specific model.

Since, the performance in each of these datasets LocalMine performs well within the range of the performance of baseline models and also outperforming in certain cases. It can be established

that LocalMine i.e. LDA based Random Forest classifier is very generic and robust to datasets. The property of this model to generalize and localize to a project is effective in extracting useful features from natural language SE texts. The projects that generally have lower F1 and recall scores have more correlation between the two topics. For our case, as a binary classifier, we have kept number of topics to be 2. It is our conscious decision that the two topics should correspond to the two classes in order to learn features of different classes. However, in case of high correlation between the topics we need to figure out a way to incorporate more feature extraction technique. We can either let LDA learn for a larger number of iteration rather than 20 that is default for LocalMine. We could also explore creating more topics and understanding topic perplexity to incorporate more granular features.

Inferring from answers of research questions we see that LocalMine is a probabilistic keyword model that generates comprehensible topics and features for effective software text mining. In particular, it performs better than state-of-the-art change message classification approaches that is based on keyword labeling. In addition, LocalMine can also be effectively used for other kinds software text mining tasks as seen from stack overflow and technical debt datasets. Hence, LocalMine is an effective probabilistic keyword model for software text mining.

BIBLIOGRAPHY

[1] Mockus, A. & Votta, L. G. "Identifying Reasons for Software Changes Using Historic Databases". *Proceedings of the International Conference on Software Maintenance (ICSM'00)*. ICSM '00. Washington, DC, USA: IEEE Computer Society, 2000, pp. 120–.

[2] Hattori, L. P. & Lanza, M. "On the Nature of Commits". *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering*. ASE'08. L'Aquila, Italy: IEEE Press, 2008, pp. III–63–III–71.

[3] Hindle, A., German, D. M. & Holt, R. "What Do Large Commits Tell Us?: A Taxonomical Study of Large Commits". *Proceedings of the 2008 International Working Conference on Mining Software Repositories*. MSR '08. Leipzig, Germany: ACM, 2008, pp. 99–108.

[4] Fu, Y., Yan, M., Zhang, X., Xu, L., Yang, D. & Kymer, J. D. "Automated classification of software change messages by semi-supervised Latent Dirichlet Allocation". English. *Information and Software Technology* **57** (2015), pp. 369–377.

[5] Hofmann, T. "Unsupervised Learning by Probabilistic Latent Semantic Analysis". *Mach. Learn.* **42**.1-2 (2001), pp. 177–196.

[6] Mauczka, A., Huber, M., Schanes, C., Schramm, W., Bernhart, M. & Grechenig, T. "Tracing Your Maintenance Work — a Cross-project Validation of an Automated Classification Dictionary for Commit Messages". *Proceedings of the 15th International Conference on Fundamental Approaches to Software Engineering*. FASE'12. Tallinn, Estonia: Springer-Verlag, 2012, pp. 301–315.

[7] Zhong, H. & Su, Z. "An Empirical Study on Real Bug Fixes". *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 1. 2015, pp. 913–923.

[8] Martinez, M., Duchien, L. & Monperrus, M. "Automatically Extracting Instances of Code Change Patterns with AST Analysis". *2013 IEEE International Conference on Software Maintenance*. 2013, pp. 388–391.

[9] Kim, S., Whitehead Jr., E. J. & Zhang, Y. "Classifying Software Changes: Clean or Buggy?" *IEEE Trans. Softw. Eng.* **34**.2 (2008), pp. 181–196.

[10] Osman, H., Lungu, M. & Nierstrasz, O. "Mining frequent bug-fix code changes". *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*. 2014, pp. 343–347.

[11] Hindle, A., Godfrey, M. W. & Holt, R. C. "What's hot and what's not: Windowed developer topic analysis". *2009 IEEE International Conference on Software Maintenance*. ICSM'09. 2009, pp. 339–348.

[12] Swanson, E. B. "The Dimensions of Maintenance". *Proceedings of the 2Nd International Conference on Software Engineering*. ICSE '76. San Francisco, California, USA: IEEE Computer Society Press, 1976, pp. 492–497.

[13] Hassan, A. E. “Automated Classification of Change Messages in Open Source Projects”. *Proceedings of the 2008 ACM Symposium on Applied Computing*. SAC ’08. Fortaleza, Ceara, Brazil: ACM, 2008, pp. 837–841.

[14] Antoniol, G., Ayari, K., Di Penta, M., Khomh, F. & Guéhéneuc, Y.-G. “Is It a Bug or an Enhancement?: A Text-based Approach to Classify Change Requests”. *Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds*. CASCON ’08. Ontario, Canada: ACM, 2008, 23:304–23:318.

[15] Yan, M., Xia, X., Shihab, E., Lo, D., Yin, J. & Yang, X. “Automating Change-level Self-admitted Technical Debt Determination”. *IEEE Transactions on Software Engineering*. TSE’18 (2018), pp. 1–1.

[16] Agrawal, A., Tu, H. & Menzies, T. “Can You Explain That, Better? Comprehensible Text Analytics for SE Applications”. *CoRR* **abs/1804.10657** (2018). arXiv: 1804.10657.

[17] Blei, D. M., Ng, A. Y. & Jordan, M. I. “Latent Dirichlet Allocation”. *J. Mach. Learn. Res.* **3** (2003), pp. 993–1022.

[18] Thomas, S. W. “Mining Software Repositories Using Topic Models”. *Proceedings of the 33rd International Conference on Software Engineering*. ICSE ’11. Waikiki, Honolulu, HI, USA: ACM, 2011, pp. 1138–1139.

[19] Grant, S., Cordy, J. R. & Skillicorn, D. B. “Using Topic Models to Support Software Maintenance”. *Proceedings of the 2012 16th European Conference on Software Maintenance and Reengineering*. CSMR ’12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 403–408.

[20] Mathew, G., Agrawal, A. & Menzies, T. “Trends in Topics at SE Conferences (1993-2013)”. *CoRR* **abs/1608.08100** (2016). arXiv: 1608.08100.

[21] Agrawal, A., Fu, W. & Menzies, T. “What is Wrong with Topic Modeling? (and How to Fix it Using Search-based SE)”. *CoRR* **abs/1608.08176** (2016). arXiv: 1608.08176.

[22] Huang, Q., Shihab, E., Xia, X., Lo, D. & Li, S. “Identifying Self-admitted Technical Debt in Open Source Projects Using Text Mining”. *Empirical Softw. Engg.* **23**.1 (2018), pp. 418–451.

[23] Majumder, S., Balaji, N., Brey, K., Fu, W. & Menzies, T. “500+ Times Faster Than Deep Learning (A Case Study Exploring Faster Methods for Text Mining StackOverflow)”. *CoRR* **abs/1802.05319** (2018). arXiv: 1802.05319.

[24] Mikolov, T., Chen, K., Corrado, G. & Dean, J. “Efficient Estimation of Word Representations in Vector Space”. *CoRR* **abs/1301.3781** (2013). arXiv: 1301.3781.

[25] Pennington, J., Socher, R. & Manning, C. D. “Glove: Global Vectors for Word Representation.” *EMNLP*. Vol. 14. 2014, pp. 1532–1543.