Approval Prediction for Software Enhancement Report using Deep Neural Network

By

Sadeem Ahmad Nafees

Fall 2017-MS-CS&E 00000203427

Supervisor

Dr. Mian Ilyas Ahmad

Department of Computational Science and Engineering

Research Center for Modelling and Simulation (RCMS)

National University of Sciences and Technology (NUST)

Islamabad, Pakistan

January 2020

Approval Prediction for Software Enhancement Report using Deep Neural Network

By

Sadeem Ahmad Nafees

00000203427

Supervisor

Dr. Mian Ilyas Ahmad

A thesis submitted in conformity with the requirements for the degree of *Master of Science* in Computational Science and Engineering

Research Center for Modelling and Simulation,

National University of Sciences and Technology (NUST)

Islamabad, Pakistan

January 2020

Declaration

I, *Sadeem Ahmad Nafees* declare that this thesis titled "Approval Prediction for Software Enhancement Report using Deep Neural Network" and the work presented in it are my own and has been generated by me as a result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a Master of Science degree at NUST
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at NUST or any other institution, this has been clearly stated
3. Where I have consulted the published work of others, this is always clearly attributed
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work
5. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself

---

Sadeem Ahmad Nafees,

00000203427

Copyright Notice

This thesis is dedicated to *my beloved parents*

Acknowledgments

"Education is simply the soul of a society as it passes from one generation to another".

Alhamdulillah, all praises to **ALLAH** for the strengths and His blessing in completing this thesis. Special appreciation goes to my supervisor, Dr Mian Ilyas Ahmad, for his supervision and constant support. His invaluable help of constructive comments and suggestions throughout the experimental and thesis works have contributed to the success of this research. Not forgotten, my appreciation to my general evaluation committee members, Dr. Muhammad Tariq Saeed and Dr. Shehzad Rasool for his support and knowledge regarding this topic. Last but not least, my deepest gratitude goes to my beloved parents who supported me financially and spiritually and also to my brother, sisters for their endless love, prayers and encouragement.

Sadeem Ahmad Nafees

Contents

List of Abbreviations

Abbreviations

CNN	Convolutional Neural Network
LSTM	Long Short Term Memory
POS	Part of speech
NLP	Natural Language Processing
Word2vec	word to vector
MNB	Multinomial Naive Bayes
SVM	Support Vector Machine
APER	Approval Prediction for Software Enhancement Report using Deep Neural Network
SAAP	Sentiment based Approval Prediction for Enhancement Report
AAP	Automatic Approval Prediction for Software Enhancement Requests

List of Tables

List of Figures

Abstract

Software applications obtain enhancement requests on a large scale to fulfil user requirements through a bug tracking system, which is an important application for keeping record of the bugs in the software development process. Conventionally, software developers used to manually check these requests. However, manual inspection of these requests turns out to be a boring, hectic and time-consuming activity. Therefore, an automatic prediction system is required which can reject or approve an enhancement report without manual check. Existing approaches target this problem through machine learning techniques such as Multinomial Naive Bayes and Support Vector Machine. In this work we utilize a deep learning based approach for approval prediction of enhancement reports that include the following steps: First, we preprocess each enhancement report using natural language processing techniques. Then we perform emotion analysis of each preprocessed enhancement reports using *Senti4SD* to compute its sentiment. Next we combine the summary and description of enhancement report as a sequence of words to learn the features with some attention mechanism using deep recurrent neural network (RNN). Finally, on the basis of features selection model and sentiment we train deep neural network based classifier, the convolutional neural network (CNN) to predict the approval prediction of enhancement report. We evaluate the proposed approach on an open-source bug tracking system. Results of numerical evaluation suggest that the proposed approach significantly outperforms the existing approaches and achieve the accuracy of 82.15%, precision 90.56%, recall 80.10% and f-measure as 85.01%.

Keywords: *Text Classification, Deep Learning, Software Engineering, Enhancement Report*

Chapter 1

Introduction

Software applications are designed to perform different tasks in order to fulfil the user requirements. Often software applications receive large number of bugs through different bug tracking systems, which are tools that keep the record of bugs in the process of software development. In this chapter we present the concept of software enhancement report and define the problem of its approval prediction. Also we specify the research objectives and contributions of our work.

1.1 Enhancement Report

There are different bug tracking systems that are used to register bug reports for a specific software application. A typical bug report highlights an error or imperfection in the working of a process or software application that may produce an unwanted result. The report contains several attributes like bug id, title, severity, reported date, environment, status and descriptions. A sample of bug report from bugzila [1] is shown in Figure 1.1.

The importance of bug report can be categorized into two main parameters, priority

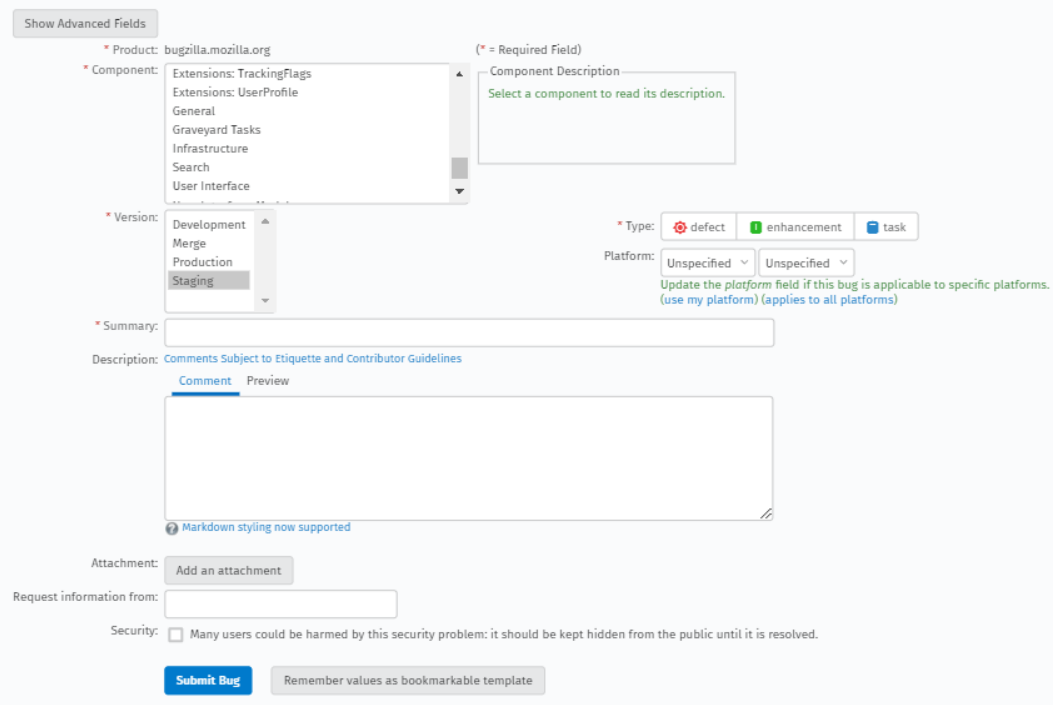

Figure 1.1: Sample of Bug Report.

and severity. These parameters are decided by the user who register the bug. One important type of severity is enhancement where the bug report suggest an enhancement in the software and therefore it also called enhancement report. Enhancement report suggest update/amendments of some features in the software application in order to resolve the issue. Therefore, for the success of software application it is necessary to update/amendments of feature enhancement with time in order to fulfil user needs. A sample of enhancement report used in [2] is shown in Figure 1.2.

Statistical analysis of enhancement report shows that considerable amount of reports were filed for enhancement. Thunderbird product (free email application) registered 1857 enhancement from 10,000 bug reports in between 01-02-2000 to 25-12-2005. Some of the enhancement report were not showing enhancement, because the user has less knowledge about the application or the quality of report is not good and this gives rise to misunderstanding between developer and users [3]. It has been observed that from

Figure 1.2: Sample of Enhancement Report.

35 different software applications, 75% of the enhancement reports were not approved [2]. This approval of the enhancement is decided by the developer manually. Since the number of enhancement reports (18.57% of bug repot) are continuously increasing, that is difficult for the developer to manually approve or reject the enhancement report. This means that an automated approach is required for approval/rejection of enhancement report before sending it to the developers. This process is shown in Figure 1.3. The benefit of an automated approach is it can help developers to save their time and efforts by prioritizing the useful and important enhancement reports from a large number of reports so that the more valuable reports could be managed efficiently.

In order to automatically predict the approval/rejection of enhancement report, authors in [2] proposed an approach that is based on machine learning classifier. They used the dataset that is extracted from open source application 'Bugzilla' which includes 40,000

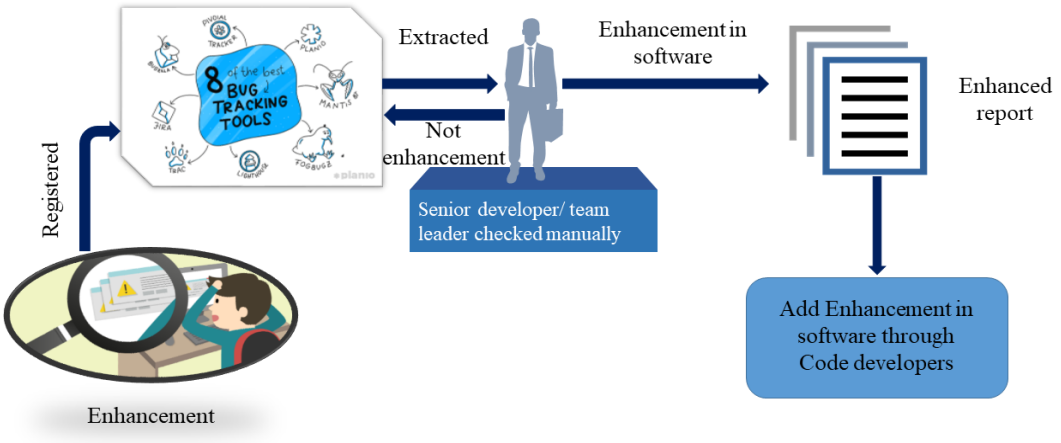

Figure 1.3: Process of Enhancement.

enhancement reports. Firstly, they performed preprocessing on each enhancement report by applying natural language processing techniques, extracted features from it and then converted it into vector form. Secondly, they fed each preprocessed enhancement report into the classifier and trained the classifier (Logistic Regression, Support Vector Machine, Multinomial Naïve Bayes and Random Forest). Results showed that the Multinomial Naïve Bayes achieved a higher accuracy than other algorithms. To the extension of this work, authors [4] proposed an approach that automatically predict the sentiment based approval of enhancement report. They utilized same dataset which included the 40,000 enhancement reports. To differentiate his approach from [2], they just added one extra feature which is sentiment and checked how sentiments effect the approval of enhancement report. They performed preprocessing on each enhancement report by applying natural language processing techniques and counted sentiments through senti4sd, converted it into vector form. After this they split the data into train/test and applied different machine learning algorithms like Logistic Regression, Multinomial Naïve Bayes,

Support Vector Machine, Random Forest and Bernoulli Naïve Bayes. Results showed that the Support Vector Machine performs better and also increase the performance in term of accuracy and f-measure up to *9.82%* and *53.66%* respectively.

In this thesis, a new approach based on sentiment analysis using deep learning classifier will be proposed. For this purpose, we will perform preprocessing using "natural language processing" techniques and handling the stack trace, URL, hex code, tokenization, stop word removal and lower case conversion on collecting enhancement reports and extract unique words from it. After this we will calculate emotions of each processed enhancement report using *senti4SD* classifier that are based on sentiment in order to observe the effect on the approval of an enhancement. After this we combine the summary and description of enhancement report as a sequence of words to learn the representation using deep RNN with attention mechanism. In last step, we train deep neural network based classifier to predict the approval prediction of enhancement report. Our results show that the deep learning model outperformed previously used machine learning approaches.

1.2 Problem statement

As discussed before, a major number of enhancement reports are rejected due to low quality of reports. This process of rejection is done manually by their developers, which is time consuming and boring. The problem is to analyze the description of enhancement report and automatically predict the acceptance and rejection of software enhancement. This is possible through a text classification procedure for which one can use machine/deep learning techniques. Since deep learning techniques often gives better results, our focus will be the use of deep learning classification for approval/rejection

prediction. This is shown in Figure 1.4.

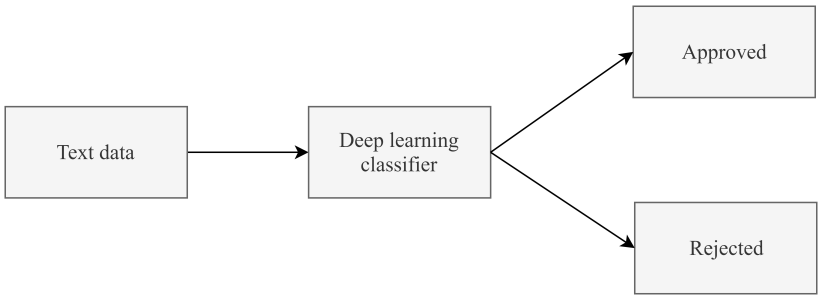

Figure 1.4: Problem statement.

1.3 Research objective

Some of the key research objective are listed below:

- The use of natural language processing techniques for extracting unique words from the description of enhancement report.
- The use of *senti4SD* classifier to calculate emotions of each processed enhancement report in order to observe the effect of emotions that predict approval/rejection of an enhancement.
- The use of long-short term memory with attention mechanism to learn the sequence of words and give weightage to words according to their importance.
- The use of deep learning based classifier on extracted features and sentiments to predict the approval prediction of enhancement report.

1.4 Motivation

Software industry spend more than 90% of software development cost on software maintenance and evolution activities [5]. A major factor in the maintenance of software is bug tracking and software enhancement. The study of enhancement prediction has the potential to reduce the time and effort of developers and cost of the software houses in term of new requirements. In particular the study helps the developers (employed or freelancers) and software houses (public and private) that are dealing with large-scale software systems. The main crux of this work is to use a text classification framework which can also be used in many other application. Some important application of text classification include classification of emails to spam/non-spam, language detection and client feedback detection.

1.5 Thesis Layout

In chapter 2, we discussed the immense research efforts that have been carried out in the field of bug/enhancement prediction and showed how our problem is related to the existing literature. In chapter 3, we present the proposed methodology to solve the problem of enhancement report prediction. In chapter 4, we show the results of the proposed methodology and compared them with existing approaches. In chapter 5, we describe future work and conclusion of the proposed work.

Chapter 2

Literature Review

In this chapter the problem of text classification and sentiment analysis is discussed using both machine learning and deep learning techniques. Also we present the approaches used for priority and severity prediction through machine/deep learning.

2.1 Text classification

Classification of text is the process in which tags/labels are assigned according to their content. Due to unstructured textual information (available in the form of chats, emails, social media and web pages) extracting knowledge from it manually can be difficult and time-consuming. Therefore, most of the businesses are shifted to automatic classification of text due to ease and cost-effectiveness. The task of text classification is to categorize a text document into two or more pre-defined classes.

Sentiment analysis plays vital role in software engineering to analyze the users attitude while writing the report, whether it is positive or negative. It is observed in [6] that a text document can be categorized into different classes of sentiments, such as SURPRISE, ANGER, SADNESS, HAPPINESS, DISGUST and FEAR. Two additional classes have

been added in [7], which are ANTICIPATION and TRUST.

A comparison between different sentiment analysis tools (Alchemy, SentiStrength and NLTK) is provided in [8], where two existing studies were replicated with different sentiment analysis tools. It has been observed that some of the existing results cannot be reproduced with the given tools.

In [9], qualitative and quantitative analysis of sentiment analysis tools are performed where 5600 manually registered JIRA reports are used. Their results showed that for detecting sentiments in text, SentiStrength-SE shows better performance as compared to existing domain independent tool such as Natural Language Toolkit, Stanford NLP and SentiStrength. In another paper [10], the same authors also developed another tool named DEVA for sentiment analysis in text. It also captures the emotional state of the report as well as sentiment analysis. For quantitative analysis of DEVA, they used 1795 JIRA reports, the evaluation results show that precision and recall is more than 82% and 78%.

Another important tool has been developed in [11] where a dataset of 2000 manually tagged review comments are used. In evaluation of their model, they executed a hundred 10-fold cross validation of different supervised machine learning algorithms (i.e. linear support vector machine, support vector machine with stochastic gradient descent, random forest, decision tree, adaptive boosting, multilayer perceptron, naïve bayes and gradient boosting tree). Assessment results showed that gradient boosting tree perform better than other algorithms, for which the mean accuracy, precision and recall are 83%, 67.8% and 58.4%.

Recently a new sentiment analysis tool *Senti4SD* has been developed in [12] for text classification. With a gold standard data set of Stack Overflow questions, answers, and

comments, they trained and validated their tool. Notably, the dataset was manually annotated for the polarity of emotions. Results indicate that the tool has removed the neutral and positive posts in the emotionally negative classified text by previous sentiment analysis tools.

2.2 Text classification through machine learning

With the rapid spread of the internet and the growth in online information, the technology has come to play a very important role in the automated classification of huge amounts of text data. Computer performance improved significantly in the 1990s, allowing to handle large amounts of text information. This contributed to using machine learning approach, which is a method of automatically developing classifiers from the text data given in a classification tag/label. This method provides excellent efficiency and ensures resource preservation. Most commonly used method in machine learning (Support Vector machine [4] and Multinomial Naïve Bayes [2]) for enhancement report prediction which contains the textual information. In machine learning, a document is represented as a n dimension vector in text classification task,

$$Documents = \{d_1, d_2, ..., d_n\} \tag{2.2.1}$$

Where d_1 , d_2 and d_n show the n number of document. There are two values for each feature of a document vector if a specific word appears in document, such as Bag-of-words.

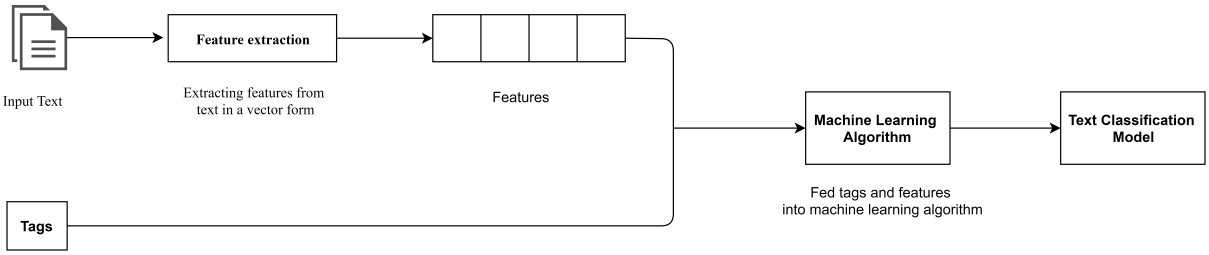

Figure 2.1: Text Classification with Machine Learning

2.2.1 Priority prediction

Usually, there are five levels of the priority for each bug report that can be represented as P_1, P_2, P_3, P_4 and P_5. P_1 shows the highest priority and P_5 shows the lowest priority. Prioritizing the bug report manually is time consuming and boring task. To assign the automatic priority of bug report, Kanwal and Maqbool proposed a machine learning based recommender [13]. They used the machine learning algorithm Naïve Bayes and Support vector machine to train their model on open source dataset of Eclipse and compare the results to find out the classifier which gives the highest accuracy. The evaluation results showed that on the text features (i.e., long description of bug report and summaries) SVM outperforms other algorithms. On the contrary, the evaluation on categorical features (severity, platform and component) shows Naïve Bayes to be better. In addition, they also experimented with the text and categorical features, combined, and showed that the highest accuracy is achieved with Support vector machine.

Authors [14] utilized Support vector machine,k-nearest neighbor, Naïve Bayes and Artificial Neural Network to automatically prioritize the newly reported bug into different categories from P_1 to P_5 . They evaluated the results through cross validation and showed that Naive Bayes shows best results with >70% accuracy, whereas, SVM, KNN and ANN achieve <70% accuracy on Eclipse dataset. The problem with the proposed approach is that its prediction accuracy highly depends upon the quality of summaries of

the bug reports. For priority, prediction of bug reports, authors [15] proposed a DRONE approach that are based on emotion analysis. Their approach focused on how emotion effects the priority prediction of bug report. For evaluation, they used Eclipse, which are open source project and suggested that proposed approach is outperform from the state-of-the-art.

2.2.2 Severity prediction

There are different types of severity levels in Bugzilla for a bug report which are categories into Normal, Major, Blocker, Critical, Trivial, Enhancement and Minor. Menzies and his colleague were the first researchers who worked on bug classification field on NASA project [16]. They proposed an automated method, named SEVERIS (Severity issue assessment). Their method predicted the severity of a reported bug. They scaled the severity of each bug report in the range of one to five. This method is the combination of machine learning and text mining techniques that are applied on bug reports. To extract the most relevant feature from each bug report text mining technique is used, for assigning the appropriate severity level using machine learning algorithms. Evaluation results showed that proposed approach classifies the bug into different categories with accuracies ranging from 65% to 98%.

Extended work of Menzies and his colleague, authors [17] introduced an automated method to predict the severity of a reported bug and also increased the performance of prediction by utilizing different open source projects like Eclipse, Gnome and Mozilla. Basically, they categorized the severity level into two groups, severe and non-severe. The non-severe group includes trivial and minor while major, critical and blocker are placed in the severe group. To categorize a given bug report into severe and non-severe

they used the Naïve Bayes algorithm and shows that the proposed methodology achieves good accuracy in term of precision and recall ranging from 65% to 75% and 70% to 85%, respectively. Authors in [17] only uses Naïve Bayes classifier to automatically predict the severity of a bug report.

Authors [18] extended his work by comparing Support Vector Machine, Multinomial Naïve Bayes, K-nearest Neighbor and Naïve Bayes on the similar dataset [17]. They utilized similar features to predict the severity of bug report and found that Multinomial Naïve Bayes classifier outperforms on open source project Eclipse, Gnome and achieve the accuracy between 48% to 93%. Drawback of this work is that they used small dataset and MNB perform well on smallest data.

As the extension of previously mentioned work, Roy and Rossi introduce a method that uses the text mining technique with bigram (arrangement of two organized elements from a string of text) to increase the performance of classification model [19]. For this purpose, they adopted the Naïve Bayes classifier on most commonly used dataset of Eclipse and Mozilla. They applied the χ^2 to get the most important features from the text on both single and bigram tokens. Their results show that the proposed model performs better than previous one.

2.2.3 Enhancement prediction

Nizamani was the first researcher who worked on the enhancement report classification and introduced a method based on machine learning to automatically predict whether a reported enhancement report will be approved or rejected [2]. For this purpose, they used the open source dataset of 'Bugzilla' which includes 40,000 enhancement reports of 35 different applications like Calendar, Firefox, Core and Thunderbird. Firstly, they

performed preprocessing on each enhancement report by applying natural language processing techniques such as stop word removal, lemmatization and stemming and extracted the features in a vector form. Secondly, they feed each preprocessed report into the classifier and train the classifier. They used the machine learning algorithms such as Logistic Regression, Support Vector Machine, Multinomial Naïve Bayes and Random Forest to predict the approval of the enhancement report. Evaluation results showed that the Multinomial Naïve Bayes achieves a higher accuracy than other algorithms.

As the extension of above mention work, authors [4] proposed a method that automatically predict the emotion based approval of enhancement report. They utilized same dataset that include the 40,000 enhancement report from 'Bugzilla'. To differentiate his approach with Nizamani's they just added one extra feature which is sentiment and check how much sentiments are effected by the approval of enhancement report. When the data is preprocessed he also counted the sentiments of the words through *Senti4SD* [12] corpus and converted it into vector form. After that they split the data into train/test and apply different machine learning algorithm such as Logistic Regression, Multinomial Naïve Bayes, Support Vector Machine, Random Forest and Bernoulli Naïve Bayes. They validated the results through cross application technique and their evaluation results showed that the proposed approach with SVM achieved good results in term of accuracy, precision, recall and f-measure score that is 77.90%, 86.28%, 66.45% and 74.53%.

2.3 Text classification through deep learning

Deep learning is an advanced discipline of machine learning that learns to perform the classification task from sound, images and text. In deep learning most commonly

used algorithms are convolutional neural network (CNN) and long-short term memory (LSTM), both are discussed in detail.

2.3.1 Convolutional neural network

Convolutional neural networks are very close to simple neural networks, that consist of multiple neurons with learning weights and biases. Each neuron takes some inputs, performs a dot product, and a non-linear function may follow it. A single differentiable score function is still expressed throughout the entire system: from inputs to output classes. The architecture of a convolutional neural network is shown in Figure 2.2 [20].

The sliding window, referred to as stride, is passed along the input with a defined step size [21]. A stride of 1 means moving the sliding window to include the next input instantly. This results in an overlap for any filter size, n, above 1.

Pooling layers apply a specified process along the input to be sub-sampled [21]. Pooling can be used to decrease the dimensionality on the output of convolutional layer. From the input matrix, Max Pooling requires the highest value within its filter size, $n_{maxpool}$. This is implemented as a sliding window, as the intention to decrease the size of the consequence by adding convolutional filters to the input of the convolutional layer, where the stride is typically set to be the same as $n_{maxpool}$.

A loss function is used as a measurement of how accurate or inaccurate mapping of a neural network [22]. Loss function provides a higher value when the network is supposed to have done worse. The weights of the network are then optimized to give as small loss as possible to the network. this is achieved with backpropagation and gradient descent.

Minimization of the loss function is done through gradient descent to estimate the weights during training [22]. The gradients are calculated by back-propagation for each

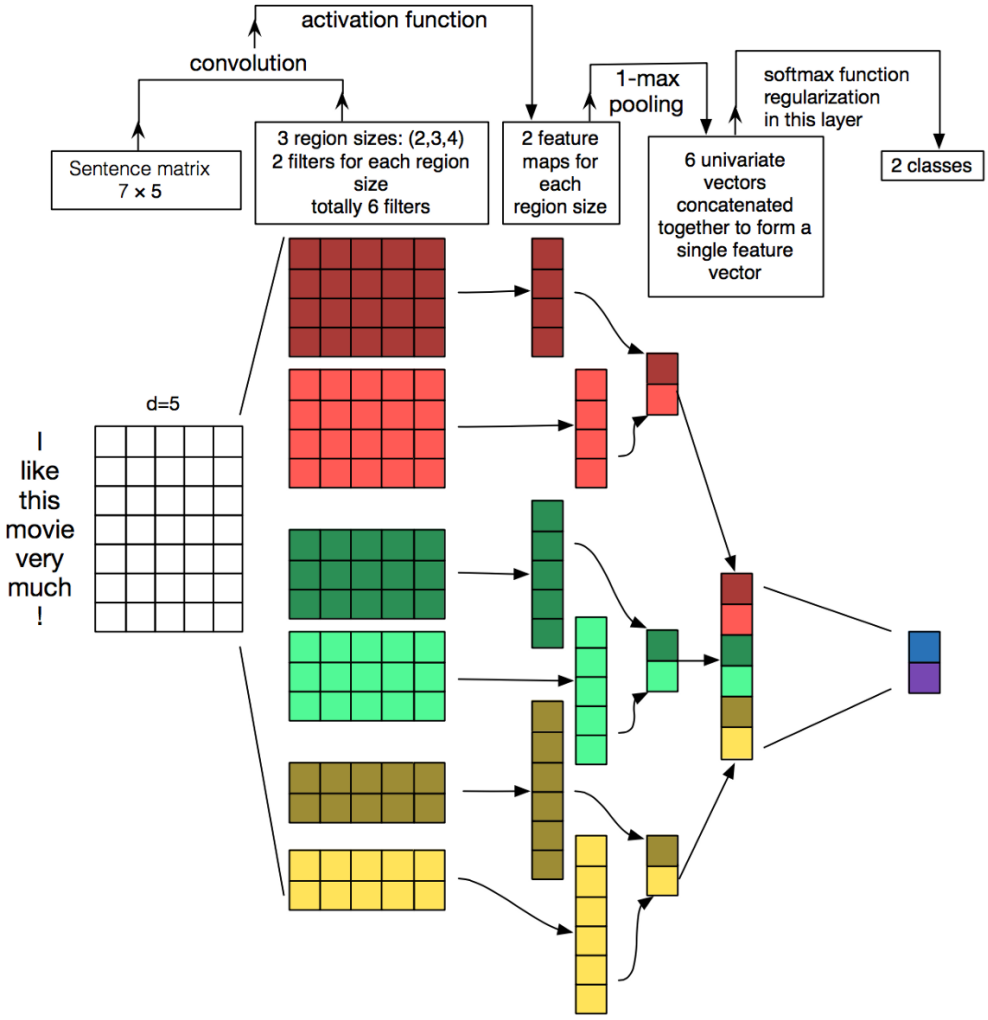

Figure 2.2: Architecture of CNN

layer, for example, beginning from the output (using the loss function) and moving back through the network. Then the weights are updated with a slight move in the gradient descent direction.

Multiple non-linear activation functions that can be used to calculate a node's final output in a neural network. Typically, a softmax activation function is used in the final layer for a classification task [22]. The tanh and sigmoid activation functions are other activation functions that are typically used within the hidden layers of a neural network.

These can be considered as smoother step-function versions. A relu activation function is typically used for convolutional neural networks. Figure. 2.3 shows the output of these activation functions.

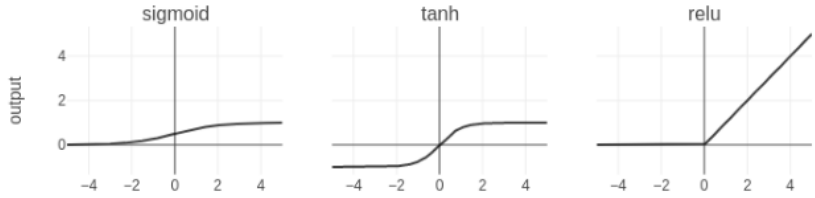

Figure 2.3: Output of Activation Functions

The concept of overfitting is a major issue when training a neural network [23]. To reduce the level of overfitting, a practically tested approach is dropout [24]. Each node in the neural network from a particular layer is removed with a probability p_{drop}, and the probability $1 - p_{drop}$ is retained. Only the weights that contributed to the outcome will be updated during backpropagation. I.e. Only the node weights that have been decided to be retained will be updated. Practically speaking, this can be seen as a network ensemble that uses an extremely elevated weight sharing level [24]. therefore, dropout refers to a model's robustness.

2.3.2 Long-short-term memory

LSTM is modified version of recurrent neural network and also fix the problem of vanishing gradient. Vanishing gradient problem occurs when the loss function gradient approaches to zero and making the network difficult to train. There is an internal state in each Long-short term memory cell. For each input, by adding and subtracting the information that are stored in internal state is updated [25]. The next state of LSTM is calculated by these formulas:

$$f_t = \sigma\left(w_f * x_t + b_f\right) \tag{2.3.1}$$

$$i_t = \sigma\left(w_i * x_t + b_i\right) \tag{2.3.2}$$

$$\tilde{c}_t = tanh(w_c * x_t + b_c) \tag{2.3.3}$$

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t \tag{2.3.4}$$

Where t represents the input index of the current state. f_t represents as forget gate, i_t represents as input gate and both are calculated at input index t. w_f and w_i are the corresponding weight matrices of the input gate and forget gate, although b_f and b_i are the corresponding biases. X is input for each LSTM cell consisting of the input index t state concatenation and the output of the previous LSTM cell, h_{t-1}. c_t represents the internal state of the LSTM. Hidden state of the LSTM is calculated by:

$$h_t = \sigma(w_o * x_t + b_o) * tanh(c_t) \tag{2.3.5}$$

Where w_o represents the output weight matrix and b_o is the biases.

In recent year, researchers are participating in deep-learning to solve software engineering tasks such as, predict defect in software [26], extract requirements from Natural Language Text [27] and generate source code [28], and achieved good results against machine learning algorithm.

To automatically predict the priority of bug report, authors [29] proposed an Artificial Neural network-based technique on international health care dataset of five different products of bug reports and validate the results through 3-fold cross validation. Their results suggest that the proposed approach performs well and produces good results in the form of f1-score, precision and recall. On the other hand, to automatically predict

the severity of bug report, authors [30] proposed a deep neural network based approach on the history data of bug report. They validate the results through cross product validation and results show that the proposed approach performs well and increase the performance in the form of f-measure upto 7.90%.

According to authors [31], different software engineering task such as, duplication of bug reports, bug report summarization, bug localization and bug triage have resolved effectively by utilizing deep learning techniques. These tasks motivate us to deploy deep learning techniques such as CNN, RNN, and LSTM to automatically predict the enhancement reports.

2.4 Proposed Research Work

In the existing literature, we know that the automatic prediction of enhancement reports has been successfully implemented using machine learning techniques and for some other applications deep learning techniques have performed better as compared to standard machine learning techniques. Therefore in this research, we propose a deep learning technique to automatically predict the approval/rejection of enhancement reports and compare the performance of the classifier with machine learning classifier. The details of our approach are discussed in chapter 3.

Chapter 3

Methodology

In this chapter, we presented the details of our proposed approach that categories an enhancement report into rejected and approved classes. The first step is to get the dataset of enhancement reports that contain status, summary and description of the enhancement. The data is then preprocessed through different natural language processing techniques that includes stop word removal, spell correction, tokenization, and lowercase conversion as well as removal of unnecessary information (special characters and URLs). Next is to compute the sentiment of each unique word in the enhancement report using *Senti4SD*. The same unique words that are used for sentiment analysis are also used to learn the features with some attention mechanism using deep recurrent neural network (RNN). Using the extracted features and sentiments of the text we train a deep neural network based classifier, the convolutional neural network (CNN) to predict the approval of enhancement report.

3.1 Data Preparation

3.1.1 Dataset

Bug tracking system allows users to report a bug when a problem is occurring by using different software applications. To maintain the quality of software applications, bug tracking system keeps the records of reported bug and provides platform for the developers to solve their bugs. We use dataset of enhancement report that is extracted from a real software application named Bugzilla. Enhancement report is usually formalized as

$$er =< t_d, r_s > \tag{3.1.1}$$

where t_d shows the textual description and r_s shows the resolution of the report.

3.1.2 Preprocessing

We preprocess the dataset of enhancement report using different natural language processing techniques which include stop word removal, lemmatization, spell correction, tokenization, and lowercase conversion. Fig 3.1. Shows all steps that involved in preprocessing.

In first step, we remove the noisy data from the text (i.e. hex code and URLs).

Tokenization

In this step, enhancement report is tokenized into separate words, by utilizing white spaces involved in the textual description. For example, “Add a copy Annotations function to the annotation service" after tokenization “Add", “a", “copy", “Annotations",

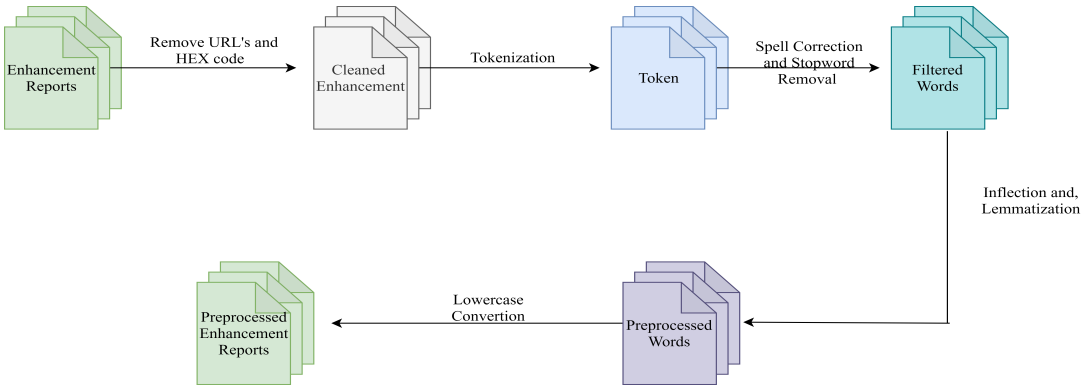

Figure 3.1: Preprocessing Steps

“function", “to", “the", “annotation", “service".

Spell checking

In this step of preprocessing, we perform spell checking and correct the mistakes found in the spellings. For example spell checker corrects the “quikly" with “quickly".

Stop-Word Removal

In this step, we remove all the words that do not carry meaningful information and those words are not required for the training of the model. Examples are, “is, am, are, he, she, it," ... etc.

Inflection

In this step, we perform inflection that transformed the plural words into their singular words. For example, “Bugs" into “Bug".

Lemmatization

In this step, we perform lemmatization that changes each word into its root form, because they carry the same meaning. For example, the words “playing", “plays" and “played" are transformed into “play".

Lower case conversion

In this step of preprocessing, we convert all uppercase characters of tokenized words into lowercase and store them as the preprocessed report. For example, “Add" into “add" and “Annotation" into “annotation”.

Final preprocessed enhancement report is formalized as,

$$er =< t_d^{'}, r_s > \tag{3.1.2}$$

where $t_d^{'}$ shows the description in the form of text and r_s shows the resolution of the report.

3.1.3 Emotion Calculation

In natural language processing to calculate the sentiment analysis of the writer is a primary task. To check the given opinion of a reporter about a report is positive or negative, we calculate the sentiments of the report from the given text [32]. Different tools are available to calculate the sentiment from a document like EmoTxt [33], SentiCR [11], Senti4SD [12], SentiStrengthSE [9], DEVA [10] and SentiWordNet [34]. We choose *Senti4SD* because it represents state of the art tool and performs well rather than SentiStrengthSE, SentiCR and SentiStrength to classify the documents. It also provides effective and efficient results. Senti4SD calculate the sentiment of the report when we

pass the description of report er to it and stored in the form as,

$$er =< t_d, s_e, r_s > \tag{3.1.3}$$

where t_d shows the description in the form of text, s_e shows the sentiment and r_s shows the resolution of the report.

3.1.4 Features selection

Bag of words is a simple representation of words which is used in natural language processing as a multiset (bag). It is a frequently used model for text classification based on frequency of words. Bag of words model extract features from each enhancement report in the form of words frequency. The main drawback of this model, it does not consider the semantic relationship between the words and order of the words in the given enhancement report.

To avoid the bag of words model problems Mikolov's proposed a method named word2vec based on skip-gram model [35]. Utilizing word2vec modelling, we convert each word into a fixed-length vector form that captures a large number of semantic and synthetic relationship between the words. Basically it is a neural network that predicts the words to nearby their context. It consider the words from given enhancement report that have same semantic meaning and context in the text. The drawback of word2vec model is, it learns the semantic relationship between the words on an individual basis rather than the sequence of words.

Discrepancies regarding bag of word model and word2vec model, it has been observed that remarkable enhancement related to words sequence, words syntax and words linguistic relationship is much needed. Solving these problem features selection model has

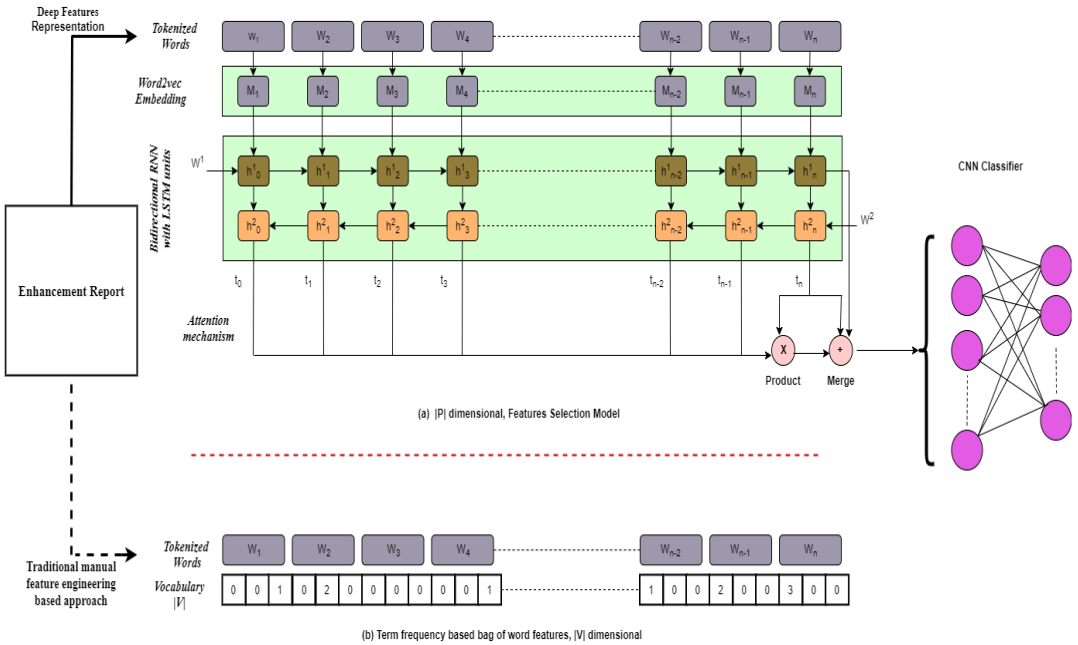

Figure 3.2: Features Selection Model

been introduced. Fig 3.2 shows the features selection model [36]. For this purpose, we use LSTM cell as a memory unit in the hidden layer [37] because these cells can memorize the word sequence and solve the problem of vanishing gradient [38]. Particularly, proposed model can memorize the word sequence in both directions forward and backwards in order to make the representation more valuable for the learning of features. In addition, proposed model deals with extracted words mentioned in above section for feature learning as an attention mechanism during classification [39].

Firstly, we use the repository to extract the |M|-dimensional representation to construct the features selection model for each enhancement report. Secondly, by using |M|-dimensional representation, we learn the |P|-dimensional word2vec representation. Finally, we use the |P|-dimensional representation to check the features with LSTM cells and it gives |R|-dimensional features of the given enhancement report. Moreover, proposed model has a sequence network (RNN) which consist of a hidden layer with n units

(h= h_1, h_2, ..., h_n). The RNN input is the word2vec representation (u = u_1, u_2, ..., u_n) and its output is the |R|-dimensional features (O = O_1, O_2, ..., O_n). In features selection model each hidden unit continuously performs the same function which alters the word u_i and a previous state t_{i-1} into the next state t_i and output word O_i.

$$f : \{t_{i-1}, u_i\} \rightarrow \{t_i, O_i\} \tag{3.1.4}$$

In addition, to learn the most important words from enhancement we utilize attention mechanism and the attention vector is derived from the weighted summation of all outputs O_i which can be defined as,

$$b_n = \sum_{i=1}^{n} b_i O_i \tag{3.1.5}$$

where b_i is the weight of each word u_i which defines the classification importance of u_i. Proposed model consists of 300 units of LSTM, 0.2 dropout, 0.001 learning rate, binary cross-entropy based loss function with adam optimizer and set the training to 100 epochs. In comparison, a term frequency based representation of BOW and word2vec, the size of features selection model is much smaller. When we select the repository size as 300 for BOW representation, the size of features selection model is recorded to be less than the word2vec representation (4 $|T|$) (< 1200) [35]. For example, if we use the 40,000 enhancement reports with 200,000 vocabulary words. Bag of word (BOW) model produces a feature matrix of size 40,000 * 200,000, at the same time features selection model produce a compact and dense vector of feature matrix of size 40,000 * 1200.

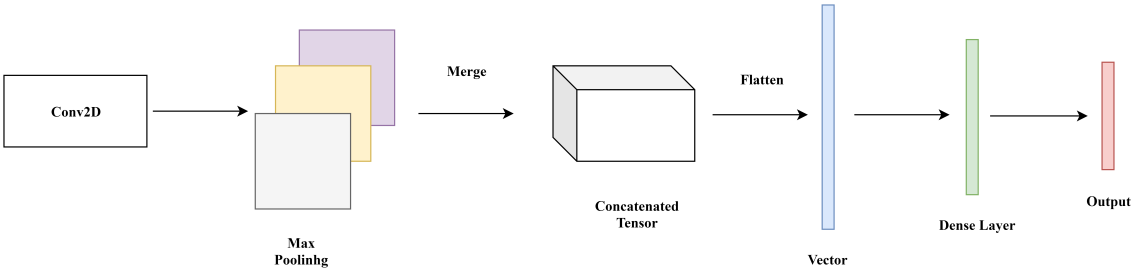

Figure 3.3: Overview of CNN Classifier

3.2 CNN for Text Classification

We use convolutional neural network to predict the approval of enhancement report. Figure. 3.3 shows the layout of CNN classifier. For the following two reasons, we choose CNN. First of all, CNN is able to learn the deep semantic relation between the words [40]. Second, different filter sizes can be used to avoid RNN's gradient problem [41]. Initially, we input two parts the features and sentiment of each enhancement to CNN. Deep learning based classifier includes 3 layers of CNN, 128 filters (neuron number), 1 kernel size (filter size), binary cross-entropy loss function, and activation (neuron final value) tanh. Second, to transform the input into a 1-dimensional vector, we move the output of the CNN to a flatten layer [42]. Importantly, with the same setting, we input the sentiment into a separate CNN and output is forward to a separate flatten layer. Finally, we fully connect the neurons between layers using the dense layer and map both inputs (features and sentiment) into a single prediction (output) using the output layer. The output predicts the given enhancement report status (approved/rejected).

Chapter 4

Results & Discussion

This chapter consist of implementation of our proposed approach “Approval Prediction for Software Enhancement Report using Deep Neural Network (APER)". Where we proposed the updated version of Umer's and Nizamani's approaches. We provided the complete comparison against existing approach with newly implemented approach and analyzed the performance of *APER* on 10 open-source Mozilla ecosystem applications. By investigating the following research question, we analyzed the *APER* performance.

4.1 Performance Analysis

Question 1: Does *APER* outperform than the existing approaches?

The proposed approach *APER* performance improvement against existing approaches is evaluated by the first question. For the following reasons, we selected two approaches, sentiment-based approval prediction for enhancements *(SAAP)* [4] and an automatic approval prediction for enhancements *(AAP)* [2]. First, both *SAAP* and *AAP* are proposed for the approval prediction of enhancements as our approach. Second, both approaches are proposed recently and are the only approaches that represent the state-

of-the-art.

Question 2: How do particular inputs (Sentiments and Features selection) affect *APER* performance?

Second question studies the impact of both inputs (Sentiments and Features selection). To investigate its impact on *APER* performance, we give one input at a time.

Question 3: How does preprocessing effect *APER* performance?

Third question evaluates *APER* performance with or without preprocessing to investigate the preprocessing effect on *APER*.

Question 4: In the approval prediction of enhancements, does the proposed deep learning classifier outperform machine / deep learning classifier?

A comparison between the proposed classifier and other classifier is provided by the fourth question. We use CNN as our classifier because it has been declared recently as the best machine learning algorithm for software documents [4].

4.2 Results Generation Process

To analyze *APSERDN* performance, we preprocess each reported enhancement report through different natural language processing techniques like stop word removal, POS tagging, spell correction, tokenization, and lowercase conversion and also remove unnecessary information like special characters and URLs from text. we count the sentiment of each enhancement report using *Senti4SD*. After this, we combine the summary and description of enhancement report as a sequence of words to learn the representation using deep RNN with attention mechanism. We carry out a cross-application evaluation to decrease validity threats by dividing all *er* enhancements into Q_i sections according

to their application where i = 1, 2, ..., 10. We gather *er* not from Q_i as training set (T_r) and enhancement Q_i as testing set (T_s), for the i^{th} cross-application evaluation. These steps are following for the cross-application evaluation. We gather T_r from e_r

$$T_{r_i} = \bigcup_{j \epsilon [1,10] \wedge j \neq i} Q_j \tag{4.2.1}$$

to train different machine and deep learning algorithms including Support Vector Machine (SVM), Long-Short Term Memory (LSTM), Convolutional Neural Network (CNN), also train the algorithms from SAAP [4] and AAP [2] on training set (T_r). We predict each testing set (T_s) of enhancement using trained algorithm CNN, SVM, SAAP and AAP and calculate the accuracy, precision, recall, f-score, Mathews correlation coefficient (MCC) and odds ratio (OR), respectively. To analyze the *APER* performance we used the well-known classification metrics (*accuracy*, *precision*, *recall* and *f-measure*). These equations are:

$$Acc = \frac{T^+ + T^-}{T^+ + T^- + F^+ + F^-} \tag{4.2.2}$$

$$Pr = \frac{T^+}{(T^+ + F^+)} \tag{4.2.3}$$

$$Re = \frac{T^+}{(T^+ + F^-)} \tag{4.2.4}$$

$$f - score = 2 * \frac{Pr * Re}{(Pr + Re)} \tag{4.2.5}$$

where T^+ means *true positive* which is the total amount of enhancement report that were correctly predicted by the classifier as approved. T^- means *true negative* which is the total amount of enhancement report that were correctly predicted by the classifier as rejected. F^+ is *false positive* denoting the total amount of rejected enhancement reports predicted by the classifier as approved. F^- is *false negative* which shows the total amount of approved enhancement reports predicted by the classifier as rejected.

We calculate MCC and OR respectively to verify the quality and efficiency of each classifier. The matthews coefficient of correlation is used as a measure of the quality of binary (two-class) classifications in machine learning. MCC is essentially a correlation coefficient between the binary classifications of observed and predicted class, it returns the value from -1 to $+1$. A $+1$ coefficient is an ideal prediction, 0 no better than a random prediction, and -1 suggests a complete inconsistency between prediction and observation.

$$MCC = \frac{T^+ * T^- - F^+ * F^-}{\sqrt{(T^+ + F^+)(T^+ + F^-)(T^- + F^+)(T^- + F^-)}} \tag{4.2.6}$$

$$OR = \frac{\frac{T^+}{F^+}}{\frac{F^-}{T^-}} \tag{4.2.7}$$

4.3 Results

4.3.1 Comparison with the baseline approaches

We compared *APER*, *SAAP*, and *AAP* performance results. We performed cross-application validation for this purpose and presented it in Table 4.1 the average analysis results of all techniques.

Table 4.1: Results of Different Approaches

Approaches	***Acc***	***Pr***	***Re***	***F-Score***	***MCC***	***OR***
APER	82.15%	90.56%	80.10%	85.01%	0.492	17.005
SAAP	77.90%	86.28%	66.45%	74.53%	0.487	16.189
AAP	70.94%	48.12%	52.59%	48.50%	0.355	12.491

As columns show the approaches result, rows show the results of *APER*, *SAAP*, and *AAP*.

We get the following conclusions, from table 4.1: APER perform better than SAAP in term of accuracy, precision, recall and f-measure up to 5.46%, 4.98%, 20.54% and 14.06%. SAAP also perform better than AAP in term of accuracy, precision, recall and f-measure up to 15.80%, 88.20%, 52.31% and 75.28%. MCC > 0 and OR > 1 average results are true for *APER* and verify the proposed approach quality and effectiveness.

Whereas the APER is correct, many false $+ve$ and false $-ve$ are observed. For instance, proposed approach predict the accepted report as a rejected "Create Bookmarks Widget with placement dependent on Bookmarks Bar status" and rejected report as accepted "Reloading resources handled by external applications". To explore the false $+ve$ and false $-ve$ we randomly selected 1,000 enhancement report from repository, noticed that due to limited (Threshold) frequency of important words that is 3, *APER* ignore some words from the given enhancement report. Considerably, because *APER* performs best at the frequency limit (Threshold) 3, we set the frequency limit 3. However, the basic reason for incorrect classifications was not fully understood. We should investigate the basic reason for incorrect classification in the future and find out solutions to decrease incorrect classifications.

4.3.2 Effect of multiple inputs

We compared *APER* performance results with and without different inputs, as features selection model and sentiment. Table 4.2 presents analysis results of *APER* by enabling and disabling some inputs. The first column provides input settings while the table rows show the *APER*s performance against each input.

We get following conclusions, from table 4.2: We observed that features selection model is enough for approval prediction of software enhancement. while the performance of

Table 4.2: Effect of multiple inputs

Input	Acc	Pr	Re	F-Score	MCC	OR
Features + Sentiment	82.15%	90.56%	80.10%	85.01%	0.492	17.005
Sentiment only	39.77%	51.38%	37.89%	43.62%	0.347	8.253
Features only	81.94%	87.53%	80.04%	83.62%	0.491	17.003

APER is reduced when we only used the features selection model in term of accuracy, precision, recall and f-measure up to 0.26%, 3.46%, 0.07% and 1.66%. But it helps *APER* to significantly improve the performance on *SAAP* and *AAP*. On the other hand, when we only use the sentiments it significantly reduces the performance of *APER* in term of accuracy, precision, recall and f-measure up to 106.56%, 76.26%, 111.40% and 94.89%.

We conclude that on the basis of previously done analysis when we disable the features selection model it significantly impact on *APER* performance. however, sentiment and features selection model are important to improve the performance for *APER*.

4.3.3 Preprocessing Impact

The textual data of enhancement report contains noisy information (e.g. hex code, punctuation and stop-word). Noisy information is irrelevant, which can directly impact on any machine learning or deep learning model performance. So, we perform preprocessing on textual data using NLP techniques to remove the noisy information of enhancement report. Through, preprocessing we not only increase the performance but also decrease the computational cost of the algorithms.

We compared *APER* performance results by using with or without preprocessing techniques. In Table 4.3, We also show the *APER* performance result with enabling or

disabling preprocessing techniques. Where, columns show the performance of *APER*, rows show the results with or without preprocessing.

Table 4.3: Preprocessing Impact

Preprocessing	Acc	Pr	Re	F-Score	MCC	OR
Enable	82.15%	90.56%	80.10%	85.01%	0.492	17.005
Disable	81.47%	87.92%	73.24%	79.91%	0.416	16.848

We get following conclusion, from Table 4.3. When we enable preprocessing techniques *APER* achieves significant performance in term of accuracy, precision, recall and f-measure up to 0.83%, 3.00%, 9.37% and 6.38%. While, when we check the *APER* performance on disabling preprocessing techniques. It decreases the Mathews Correlation Coefficient to 0.416 and Odd Ratio to 16.848. MCC (0.416) $>$ 0 and OR (16.848) $>$ 1 average results are true for *APER* and verify the proposed approach quality and effectiveness. we conclude that on the basis of previously done analysis when we disable the preprocessing techniques. it decreases the *APER* performance and show that preprocessing is crucial for approval prediction of software enhancement report.

4.3.4 Comparison with machine/deep learning algorithms

We compared APER performance with SVM because it's finest algorithm in machine learning for software engineering document [4] and LSTM is proved to be beneficial in natural language processing [43]. Specifically, for the comparison of chosen classifiers, we gave the same preprocessed enhancement report, features and their sentiment. We use Long-short-term memory with the provided setting (dropout = 0.2, Activation function = sigmoid and loss function = binary-cross entropy) and Support Vector Machine with

default Setting.

We show the classifier results in Table 4.4. Column shows the classifier and rows show the results of all classifier that we used.

Table 4.4: Comparison with machine/deep learning algorithm

Algorithms	Acc	Pr	Re	F-Score	MCC	OR
CNN	82.15%	90.56%	80.10%	85.01%	0.492	17.005
LSTM	80.73%	91.56%	74.81%	82.34%	0.484	16.917
SVM	77.95%	82.85%	71.58%	76.80%	0.465	14.984

We get the following conclusion, from Table 4.4: The CNN performs better than the LSTM classifier in term of accuracy, precision, recall and f-measure up to 1.76%, -1.09%, 7.07% and 3.24%. Because, CNN perform well on long input as our input text, does not require tedious and time-consuming feature modeling. CNN also useful for local and position invariant features. CNN also performs better than the SVM classifier in term of accuracy, precision, recall and f-measure up to 2.37%, 5.52%, 4.60% and 5.03%. SVM has to handle high dimension input size that required high computation, while features selection model significantly reduces the length of feature set. we conclude that on the basis of previously done analysis in the approval prediction of software enhancement report CNN performs better than other classifiers.

4.4 Threats to Validity

Although the labeling of Software Engineering reports is usually not accurate [4]. The first threat to construct validity is the possibility of inaccurate enhancement report la-

beling. So, we consider the enhancement report that are reused will be labelled correctly, but incorrect labeling can affect the *APER* performance. The selection of evaluation metrics is the second threat to construct validity. Because the confusion metrics that are selected for enhancement report classification are the most adopted and well-known metrics.

The threat to internal validity is we use the *senti4SD* for sentiment analysis of enhancement report because it performs better than the other available repository. Other repositories may be used to affect *APER* performance. The second internal validity threat is *SAAP* and *AAP* implementation. To minimize the threat, the implementation and outcomes of approaches are double-checked. Furthermore, some unfolding errors can affect *APER* performance. The third internal validity threat is related to the CNN hyperperameters that we used to train the classifier as mentioned in above section. Changing other default parameters can affect *APER* performance.

The performance of *APER* against other datasets is a threat to external validity. As mentioned in above section, we only test *APER* on 10 open source applications. The addition of other inter / intra domain of enhancement report can affect *APER* performance.

Chapter 5

Conclusion & future work

In this chapter, we have presented the conclusion and future work that acquired from our proposed work.

5.1 Conclusion

We proposed a combined natural language processing, sentiment and deep learning based approach that automatically predicts the approval of enhancement reports. The results are compared with existing machine or deep learning algorithms such as support vector machine (SVM), long short-term memory (LSTM), multinomial naive bayes (MNB) and convolutional neural networks (CNN). It has been observed that the proposed approach achieves higher accuracy, precision, recall and f-measure (82.15%, 90.56%, 80.10% and 85.01%) than the existing approaches. One of the important contribution in this research is to leveraged LSTM for sequence generation instead of previous feature learning techniques. We believe that our approach will help developers to save their time and address user-requirements in a more efficient manner.

5.2 Future Work

The main part of this work is to predict the approval of enhancement report automatically using combination of machine or deep learning techniques. An important future direction of our proposed work is to utilize deep learning classifiers on a large amount of inter/intra domain to further improve the overall performance. Another aspect is to enhance the performance of proposed approach by using different emotions. Furthermore, it will be interesting to see the effect of increasing the dataset on the approval prediction of an enhancement report. Also labels of the known dataset can be predicted through this approach, which is another important future direction.

References

[1] Link. https://bugzilla.mozilla.org/enter_bug.cgi?format=__default__&product=bugzilla.mozilla.org/. Accessed: 2019-10-10.

[2] Zeeshan Ahmed Nizamani, Hui Liu, David Matthew Chen, and Zhendong Niu. Automatic approval prediction for software enhancement requests. *Automated Software Engineering*, Oct 2017. ISSN 1573-7535. doi: 10.1007/s10515-017-0229-y. URL https://doi.org/10.1007/s10515-017-0229-y.

[3] Adrian Schröter, Cathrin Weiss, Rahul Premraj, Nicolas Bettenburg, Sascha Just, and Thomas Zimmermann. What makes a good bug report? *IEEE Transactions on Software Engineering*, 36:618–643, 2010. ISSN 0098-5589. doi: doi.ieeecomputersociety.org/10.1109/TSE.2010.63.

[4] Qasim Umer, Hui Liu, and Yasir Sultan. Sentiment based approval prediction for enhancement reports. *Journal of Systems and Software*, 155:57 – 69, 2019. ISSN 0164-1212. doi: https://doi.org/10.1016/j.jss.2019.05.026. URL http://www.sciencedirect.com/science/article/pii/S0164121219301104.

[5] Tao Zhang, He Jiang, Xiapu Luo, and Alvin TS Chan. A literature review of research in bug resolution: Tasks, challenges and future directions. *The Computer Journal*, 59(5):741–773, 2016.

[6] Paul Ekman. An argument for basic emotions. *Cognition & emotion*, 6(3-4):169–200, 1992.

[7] Robert Plutchik. Emotion. *A psychoevolutionary synthesis*, 1980.

[8] Robbert Jongeling, Proshanta Sarkar, Subhajit Datta, and Alexander Serebrenik. On negative results when using sentiment analysis tools for software engineering research. *Empirical Software Engineering*, 22(5):2543–2584, Oct 2017. ISSN 1573-7616. doi: 10.1007/s10664-016-9493-x. URL https://doi.org/10.1007/s10664-016-9493-x.

[9] Md Rakibul Islam and Minhaz F. Zibran. Sentistrength-se: Exploiting domain specificity for improved sentiment analysis in software engineering text. *Journal of Systems and Software*, 145:125 – 146, 2018. ISSN 0164-1212. doi: https://doi.org/10.1016/j.jss.2018.08.030. URL http://www.sciencedirect.com/science/article/pii/S0164121218301675.

[10] Md Rakibul Islam and Minhaz F. Zibran. Deva: Sensing emotions in the valence arousal space in software engineering text. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, SAC '18, pages 1536–1543, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5191-1. doi: 10.1145/3167132.3167296. URL http://doi.acm.org/10.1145/3167132.3167296.

[11] Toufique Ahmed, Amiangshu Bosu, Anindya Iqbal, and Shahram Rahimi. Senticr: A customized sentiment analysis tool for code review interactions. In *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, ASE 2017, pages 106–111, Piscataway, NJ, USA, 2017. IEEE Press. ISBN 978-1-5386-2684-9. URL http://dl.acm.org/citation.cfm?id=3155562.3155579.

[12] Fabio Calefato, Filippo Lanubile, Federico Maiorano, and Nicole Novielli. Sentiment polarity detection for software development. *Empirical Softw. Engg.*, 23(3):1352–1382, June 2018. ISSN 1382-3256. doi: 10.1007/s10664-017-9546-9. URL https://doi.org/10.1007/s10664-017-9546-9.

[13] Jaweria Kanwal and Onaiza Maqbool. Bug prioritization to facilitate bug report triage. *Journal of Computer Science and Technology*, 27(2):397–412, 2012.

[14] Gitika Sharma, Sumit Sharma, and Shruti Gujral. A novel way of assessing software bug severity using dictionary of critical terms. *Procedia Computer Science*, 70(Supplement C):632 – 639, 2015. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2015.10.059. URL http://www.sciencedirect.com/science/article/pii/S1877050915032238. Proceedings of the 4th International Conference on Eco-friendly Computing and Communication Systems.

[15] Qasim Umer, Hui Liu, and Yasir Sultan. Emotion based automated priority prediction for bug reports. *IEEE Access*, 6:35743–35752, 2018.

[16] Tim Menzies and Andrian Marcus. Automated severity assessment of software defect reports. In *2008 IEEE International Conference on Software Maintenance*, pages 346–355. IEEE, 2008.

[17] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals. Predicting the severity of a reported bug. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pages 1–10, May 2010. doi: 10.1109/MSR.2010.5463284.

[18] Ahmed Lamkanfi, Serge Demeyer, Quinten David Soetens, and Tim Verdonck. Comparing mining algorithms for predicting the severity of a reported bug. In *Proceedings of the 2011 15th European Conference on Software Maintenance and*

Reengineering, CSMR '11, pages 249–258, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-0-7695-4343-7. doi: 10.1109/CSMR.2011.31. URL http://dx.doi.org/10.1109/CSMR.2011.31.

[19] N. K. S. Roy and B. Rossi. Towards an improvement of bug severity classification. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 269–276, Aug 2014. doi: 10.1109/SEAA.2014.51.

[20] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015.

[21] Viktor Stagge. Categorizing software defects using machine learning. *LU-CS-EX 2018-17*, 2018.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. book in preparation for mit press. *URL¡ http://www. deeplearningbook. org*, 2016.

[23] Igor V Tetko, David J Livingstone, and Alexander I Luik. Neural network studies. 1. comparison of overfitting and overtraining. *Journal of chemical information and computer sciences*, 35(5):826–833, 1995.

[24] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[25] Christopher Olah. Understanding lstm networks. 2015.

[26] Rajni Jindal, Ruchika Malhotra, and Abha Jain. Software defect prediction using neural networks. In *Proceedings of 3rd International Conference on Reliability, Infocom Technologies and Optimization*, pages 1–6. IEEE, 2014.

[27] Hong Liang, Xiao Sun, Yunlei Sun, and Yuan Gao. Text feature extraction based on deep learning: a review. *EURASIP journal on wireless communications and networking*, 2017(1):1–12, 2017.

[28] Akshay Sethi, Anush Sankaran, Naveen Panwar, Shreya Khare, and Senthil Mani. Dlpaper2code: Auto-generation of code from deep learning research papers. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[29] Lian Yu, Wei-Tek Tsai, Wei Zhao, and Fang Wu. Predicting defect priority based on neural networks. In *International Conference on Advanced Data Mining and Applications*, pages 356–367. Springer, 2010.

[30] Waheed Yousuf Ramay, Qasim Umer, Xu Cheng Yin, Chao Zhu, and Inam Illahi. Deep neural network-based severity prediction of bug reports. *IEEE Access*, 7: 46846–46857, 2019.

[31] Xiaochen Li, He Jiang, Zhilei Ren, Ge Li, and Jingxuan Zhang. Deep learning in software engineering. *arXiv preprint arXiv:1805.04825*, 2018.

[32] Edward Loper and Steven Bird. Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ETMTNLP '02, pages 63–70, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1118108.1118117. URL https://doi.org/10.3115/1118108.1118117.

[33] Fabio Calefato, Filippo Lanubile, and Nicole Novielli. Emotxt: A toolkit for emotion recognition from text. *2017 Seventh International Conference on Affective*

Computing and Intelligent Interaction Workshops and Demos (ACIIW), pages 79–80, 2017.

[34] Stefano Baccianella, Andrea Esuli, and Fabrizio Sebastiani. Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining. In *in Proc. of LREC*, 2010.

[35] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 3111–3119, USA, 2013. Curran Associates Inc. URL http://dl.acm.org/citation.cfm?id=2999792.2999959.

[36] Senthil Mani, Anush Sankaran, and Rahul Aralikatte. Deeptriage: Exploring the effectiveness of deep learning for bug triaging. In *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, pages 171–179. ACM, 2019.

[37] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

[38] H Sak, Andrew Senior, and F Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, pages 338–342, 01 2014.

[39] Minh-Thang Luong, Hieu Pham, and Christoper Manning. Effective approaches to attention-based neural machine translation. 08 2015. doi: 10.18653/v1/D15-1166.

[40] W. Y. Ramay, Q. Umer, X. C. Yin, C. Zhu, and I. Illahi. Deep neural network-based severity prediction of bug reports. *IEEE Access*, 7:46846–46857, 2019. ISSN 2169-3536. doi: 10.1109/ACCESS.2019.2909746.

[41] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. 2012. URL http://arxiv.org/abs/1207.0580.

[42] Keras. Flatten layer. RetrievedNov01,2018fromhttps://github.com/keras-team/keras/blob/master/keras/layers/core.py#L467.

[43] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing [review article]. *IEEE Computational Intelligence Magazine*, 13:55–75, 2018.