Learning to Categorize Issues in Distributed Bug Tracker Systems

Automatisierte Fehlerticket Kategorisierung in verteilten Umgebungen

Master-Thesis von Kaushikkumar Gondaliya aus India

Juni 2017

TECHNISCHE UNIVERSITÄT DARMSTADT

Learning to Categorize Issues in Distributed Bug Tracker Systems
Automatisierte Fehlerticket Kategorisierung in verteilten Umgebungen

Vorgelegte Master-Thesis von Kaushikkumar Gondaliya aus India

1. Gutachten: Dr. Elmar Rueckert
2. Gutachten: Prof. Dr. Jan Peters
3. Gutachten: Mr. Christian Bernecker
4. Gutachten: Mr. Thorsten Busch

Tag der Einreichung:

Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
In der abgegebenen Thesis stimmen die schriftliche und elektronische Fassung überein.

Darmstadt, den 27. Juni 2017

(Kaushikkumar Gondaliya)

Thesis Statement

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.
In the submitted thesis the written copies and the electronic version are identical in content.

Darmstadt, June 27, 2017

(Kaushikkumar Gondaliya)

Abstract

A bug report plays an essential role in software development and maintenance. It is a key element of an efficient IT problem management. Due to the complexity of a software product, many teams such as design, usability, performance etc. work on a single product. Bug reports typically need to be routed to these different expert teams to be resolved. A manual routing process is time-consuming and less efficient. Therefore, the challenge is to find an approach that automate the routing process. With the advent of natural language processing methods, machine learning and deep learning algorithms, new automatic approaches were proposed to address this challenge.

In this thesis, bug reports from the business process manager (BPM) product of IBM Deutschland GmbH were used to compare and assess the performance of prediction models. Different combination of natural language processing methods (i.e., *lemmatization*, *pos tagger*, *N-gram* and *stopword removal*) and classification algorithms (i.e., *linear support vector machines (SVM)*, *multinomial naive Bayes*, and *long sort term memory (LSTM)*) were evaluated. The feature processing was done using the *term frequency-inverse document frequency (TF-IDF)* method. Our evaluation shows that the best obtained prediction model for bug triage system is a combination of the *bigram* method with a *linear support vector machine*. For the *long short term memory (LSTM)*, more than 8000 features would be needed. The bug triage tool was implemented in microservice architecture using docker containers which allows for extending of individual components of frameworks.

Zusammenfassung

Fehlerberichte spielen in der Softwareentwicklung, Wartung und Support eine wesentliche Rolle. Sie sind Schlüsselelemente eines effizienten IT Problemmanagements. Aufgrund der Komplexität einzelner Softwareprodukte arbeiten in großen Firmen viele einzelne Teams (Design, Performance, Konfiguration) an einem einzigen Produkt. Fehlerberichte müssen in der Regel an diese verschiedenen Expertenteams weitergeleitet werden. Ein manueller Routing-Prozess ist zeitaufwändig und wenig effizient. Dementsprechend ist die Herausforderung, ein Ansatz zu finden, der den Routing-Prozess automatisiert. Mit der Hilfe der Methoden maschinelles Lernen, Sprachverarbeitung und neuronalen Netzen wurden neue automatisierter Ansatz entwickelt, um diese Herausforderung zu bewältigen.

In dieser Abschlussarbeit wurden Fehlerberichte aus dem Business Process Manager (BPM) Produkt der IBM verwendet, um die Leistung der Vorhersagemodelle zu vergleichen und zu bewerten. Dabei wurden verschiedene Kombination von Methoden der Sprachverarbeitung (Lemmatisierung, Pos-Tagger, N-Gramm und Stoppwort) und Klassifizierung (Linear Support Vector Machies (SVM), Multinomiale Naive Bayes und Long Short Term Memory (LSTM)) untersucht. Die Merkmalsverarbeitung und Gewichtung wurde unter Verwendung des Term Frequency - Inverse Document Frequency (TF-IDF) durchgeführt. Das Ergebnis der Arbeit zeigt, dass eine Kombination der Bigram-Methode und der Linear Support Vektor Machine die besten Ergebnisse für ein Vorhersagemodell liefert. Die Ergebnisse für den Long Short Term Memory (LSTM) Ansatz zeigen das eine Mindestmenge an 8000 Merkmalen erforderlich ist. Das Bug-Triage-Tool wurde in der Microservice-Architektur mit Docker-Containern implementiert, die eine Erweiterung der einzelnen Komponenten von Frameworks ermöglicht.

Acknowledgments

This master thesis has put me on various challenges, however, at the end of this thesis, I feel enriched with a fulfilling experience. First of all, I would like to express my sincere gratitude to my host supervisor Mr. Christian Bernecker and my manager Mr. Thorsten Busch, who believed in me and provided me an excellent opportunity to work at IBM Deutschland GmbH.

I am very grateful to Professor Dr. Jan Peters for accepting my thesis in Intelligent Autonomous Systems (IAS) department. I would like to thank my supervisor, Dr. Elmar Rueckert, without whom this thesis would not be what it currently is. I am very grateful for his support and encouragement. Despite his busy schedule, he always found time for me and had always provided proper guidance.

My heartful regard goes to my parents. I salute them for their selfless love, care and sacrifice they did to shape my life. Although they hardly understood what I researched on, they always supported me in any decision I made. Also, I would like to express my heartful thank to my both the sisters for their constant love, care and belief on me. My special thanks would go to my friends for reviewing my work and giving me precious suggestions to make it better. Finally, I would like to thank the Lord Almighty for his blessing, mercy and unseen guidance.

Contents

Figures and Tables

List of Figures

List of Tables

1 Introduction

The demands of quality software products have rapidly increased and a significant amount of cost is spent on the quality and maintenance. Various testing methods are used to ensure the quality of a software. However, a software is almost never perfect and needs to be maintained continuously for the new issues [1, 5]. Here, the issues denote code errors (a bug, an error in a computer program that causes unexpected behaviour). According to the study conducted by a Ponemon Institute and Emerson Network in 2013, the overall average per-incident cost of a data center outage was 680,000 US dollars[1]. In the same year, IBM received more than 6.8 million service requests[2]. Traditionally, large software systems have several teams working on a single product as it is usual to discover thousands or even ten thousands of bugs. The software development community came up with web based applications to track and maintain a list of issues and to reduce an amount of efforts [6]. This kind of software is called an issue tracker (or a bug tracker) and one entry of a bug is referred as a bug report or ticket. A bug report usually contains mainly a title, a description, and an assignee for that bug report. Moreover, it also has other meta fields such as a priority assignment, a status and comments etc. The assignee in a bug report is a developer assigned to investigate and resolve the bug. Thus, bug tracking and resolving is an essential part of the software development, the maintenance and the support of such a system. It provides a platform for users to report the issues they faced while using the software products [7]. Timely resolution of these bugs increase the software quality and also keeps software alive [8].

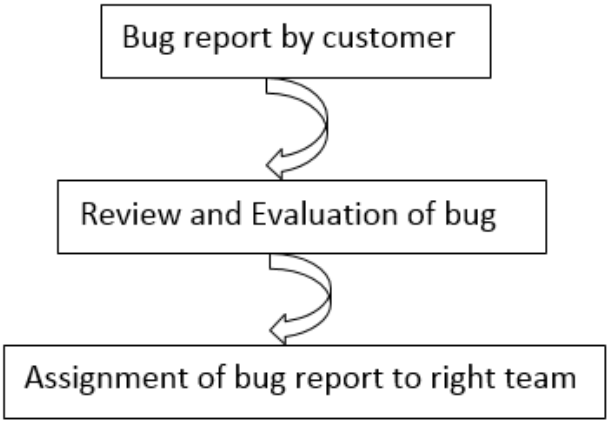


Figure 1.1.: Bug tracking process of bug reports. Customer creates a bug report which is reviewed and then forwarded to the right team for resolution [1].

In the literature of software maintenance, the term "*bug tracking*" used as a broad term referring to bug assignment, bug validation, marking duplicate bugs. In this thesis, we investigate automatic bug assignment (find right team or developer to resolve the bug) only. The process of bug a tracking case is shown in the Figure 1.1. If a customer encounters any issue in a software product, the bug report is written with a detailed description of an issue. After the bug report is received, in general, the first task is to analyze the bug report and then after forwarding it to the appropriate responsible team. This task is done manually by a triager, who has required knowledge and experience about the software product to analyze the bug report [1].

The process to analyze and assign thousands of ticket manually everyday is time-consuming and error-prone as it relies on the knowledge and experience of a triager. For example, the study on Eclipse project conducted by [9] reports that an average of 37 bugs per day are submitted to the bug tracking system (BTS) and three person-hours per day are required for the manual triage. Therefore, an automatic and an efficient bug tracking tool is required to reduce human efforts and to make a bug tracking process less time consuming.

In this thesis, we explore the use of machine learning, deep learning and natural language processing toward effective and automatic bug assignment along three dimensions:

[1] http://www.ponemon.org/local/upload/file/2013%20Cost%20of%20Data%20Center%20Outages%20FINAL%2012.pdf
[2] http://public.dhe.ibm.com/common/ssi/ecm/mt/en/mtw03011usen/MTW03011USEN.PDF

(i) the choice of classification algorithms

(ii) the choice of natural language processing for better features from the text

(iii) the efficiency–precision trade-off

Our thorough exploration along these dimensions have lead us to develop an application that achieves high levels of bug assignment accuracy and precision.

The traditional way of a software development is a monolithic approach where a single deployment unit contains all the responsibilities. It might be a beneficial for the small applications. However, it limits the scalability and flexibility of applications. The larger the code base becomes, the more complicated it becomes[10]. Therefore, it is challenging to apply any changes for the future requirements. It also slows down the release cycle of a product [10]. Therefore, today modern applications are moving from the monolithic approach to the distributed approach, where a large application is divided into the smaller parts according to their functions [11]. These smaller parts communicate with each other via synchronous or asynchronous techniques. The microservice architecture demonstrates such a distributed approach [11], where a specific component of software takes care of specific functionality. Docker containers [12] and supporting environment such as Kubernetes [13] facilitate the microservice architecture. Containers with microservice have the ability to enable autonomic strategies like re-execution of a failed service, changing workloads etc. Furthermore, decoupling in microservice has the ability to select and execute those services in a decentralized manner [3]. So there is no single point of failure for the large applications.

1.1 Motivation of IBM

The initiative for this study emerged from the IBM Deutschland GmbH that requested an automated bug assignment application to speed up its internal workflow. The approach we chose to resolve the problem uses machine learning (ML) algorithms to predict the correct assignee based on the past bug reports. The primary objective was to evaluate the chosen ML models and natural language processing (NLP) approaches to determine which one with what configuration is the best choice for the dataset provided by IBM. In this thesis, prototypes of a bug tracking tool were created for the business process management (BPM) product from IBM. The bug reports from this product are used as dataset to learn a prediction model. BPM product maintenance is handled by the seven different teams, representing seven categories of the bug reports. The idea of this project was to develop a multiclass classification model with the seven categories. *naive Bayes*, *support vector machine* and *long short term memory (LSTM)* algorithms were used to build such a multiclass classification model. For the natural language processing, *punctuation* and *stop word removal*, *lemmatization* , *pos tagger*, *bigram* methods were used. Those were compared to find the best NLP combination. We implemented the term *frequency - inverse document frequency (TF-IDF)* for the feature extraction as this method was very promising based on previous research. Various performance measures like accuracy, precision, recall, and F-measure were calculated and compared for all the machine learning algorithms [14]. We also created a web based tool for gathering real feedbacks about our prediction model from the IBM support employee. Those feedbacks can be used to extend our model with active learning in future. The tool was implemented in microservice architecture using the docker containers.

1.2 Related Work

Few attempts have been made to develop an automatic bug tracking system. In previous studies, text mining and machine learning algorithms used for detection of bug report [15, 16], bug prioritization [7, 17], bug categarization [18, 19, 20, 21, 22] and severity [7, 23]. However, our focus is only on the bug categorization in this thesis. In the following section, we briefly discuss some of the previous related works for bug categorization.

Supervised Learning Approach
Most of the previous works in bug categorization used a supervised learning approach. They experimented on different datasets and compared different classification algorithms.

In [18], the authors have proposed an automatic routing system to classify incoming tickets to the responsible maintenance teams. The idea was to develop a continuously running an automatic router with the lowest misclassification error. They gathered around 6000 tickets from a large software system for about two years which were classified into eight different categories by human experts. They compared several classification approaches like *Bayesian model*, *support vectors*, *classification trees* and *k-nearest neighbor* classification to find the best classification engine. Their empirical results showed that the probabilistic models, i.e., the *k-nearest neighbor classification* and the *support vector machines* outperformed the others. Furthermore, the precision of the prediction model showed an improvement with an increase

in the amount of training data, while it was not the same for other models, especially for the *vector space model*. However, for natural language processing they just used the *stopword removal* and *stemming method*.

In [19], the authors evaluated a text categorization approach but instead of a *naive Bayes* classifier, they used a *support vector machine (SVM)*. They used the datasets *Eclipse* [24] and *Firefox* [25] which had 8,655 and 9,752 bug reports, respectively [19]. For the *Firefox* dataset, the developer who submitted the last patch was used for labelling the bug reports. For the *Eclipse* dataset, the developer's name was used for labelling the bug reports, one who marked the bug report as "resolved". In a case of duplicates, a name of the developer who resolved the original bug report was taken for Eclipse and Firefox. They also tried *Global Compositae Checklist (GCC)* [26] dataset but the result was not so promising compared to *Eclipse* and *Firefox* datasets. They achieved 64% precision for *Firefox* dataset and 58% precision for *Eclipse* dataset. However, they obtained only 3% recall value for *Firefox* dataset and 10% recall value for *Eclipse* dataset. The *bag of words method* with a *stopwords removal* was used for a feature selection.

In both the previous studies [18, 19], a comparison of *stopwords removal* method with any other natural language processing methods were missing.

Supervised Learning Approach with Term Frequency-Inverse Document Frequency (TF-IDF)
Some of them used a *TF-IDF* method along with supervised learning approach to improve performance of the classifiers. In [27], the authors have proposed a model using *term frequency-inverse document frequency (TF-IDF)* and compared it with a normal *naive Bayes* approach. They used the tickets from the *bugzilla repository*[25] of bug reports [28]. They demonstrated their approach on a dataset of 10,000 tickets to classify them into security or non-security tickets. Their evaluation showed that their model achieved an accuracy of 93.9% and precision of 92.5% whereas when compared to that of *naive Bayes*, its accuracy and precision were 90% and 82.4% respectively. They only used *naive Bayes* for the comparison, the effects of *TF-IDF* on other algorithms were missing.

In [8], the authors compared more algorithms along with the *naive Bayes* algorithm. The used dataset was from *Mozilla* [25] with 1,983 bug reports in a total. They used a *term frequency - Inverse Document Frequency (TF-IDF)* weighted schemes for a feature extraction from the normal text. Here, the value of IDF was three which means that "terms" were removed from the feature list if it appeared in more then three documents. Furthermore, they compared different machine learning algorithms such as *naive Bayes*, *support vector machine*, *radial basis function (RBF) Network* and *Random Forest*. The best combination was *latent semantic indexing* and *support vector machine* with the result of 44.4% classification accuracy, 30% precision and 28% recall value. The dataset was very small so the results were low compare to the results of [27].

Both [27, 8] showed that *TD-IDF* method improved the results of classification algorithms. So we used *TD-IDF* method in this thesis. Furthermore, We applied other natural language processing methods such as *lemmatization*, *pos tagger* and *bigram*.

Semi-supervised Approach
All previously mentioned efforts used supervised learning approaches. In [20], the authors have used a semi-supervised text classification approach for bug categorization. The approach included clustering, active-learning and semi-supervised learning algorithms. The main benefit of using a semi-supervised technique was that it required very few labelled samples to achieve high performance. The used dataset consists of 500 bug reports from *Mahout* [29], *Lucene* [30] and *OpenNLP* [31] projects. The best model was able to achieve a 65% weighted precision, 67% recall, 62% F-measure and 71% of accuracy with only 50 manually labelled bug reports.

Content-based Recommendation Approach
In [21], the authors used a very interesting idea of content-based Recommendation (CBR). They evaluated the combination of an existing CBR with a collaborative filtering recommender (CF), referred as a content-boosted collaborative filtering (CBCF). Furthermore, they developed a topic model to reduce the sparseness and to enhance the quality of CBCF. They used the bug reports from *Apache* [32], *Eclipse*[24], *Linux kernel* [33] and *Mozilla* [25] which were created in a period of about 8 to 12 years. After removing all the bugs that were fixed by inactive developers (determined by interquartile range), resulted dataset contained 656 reports assigned to 10 developers for *Apache*, 47,862 reports and 100 developers for *Eclipse*, 968 reports and 28 developers for *Linux kernel* and 48,424 reports assigned to 117 developers for *Mozilla*. The result in terms of accuracy were 64%, 35%, 25% and 59% for *Apache*, *Eclipse*, *Linux kernel* and *Mozilla* projects, respectively.

DevRec Algorithm Approach
In [22], the authors employed the algorithm defined as DevRec based on a weighted sum of the multi-label *k-nearest neighbor classifier (ML–kNN)* and a topic modeling using *Latent Dirichlet Allocation (LDA)*. The algorithm used a composite of Bug report based (BR-based) analysis and Developer based (D-based) analysis. The BR-based analysis computes scores of the developers that are potential resolvers of a new bug report. The D-based analysis computes affinity of a bug to the developers based on their history (if they resolved some bugs in the past). The evaluation was based on the five different

datasets *GNU Compiler Collection* [34], *OpenOffice* [35], *Mozilla* [25], *Netbeans* [36], and *Eclipse*[24] containing 6,000, 15,500, 26,000, 26,000 and 34,400 bug reports, respectively. For top-5 recommendation, the respective recall values for *GCC*, *OpenOffice*, *Mozilla*, *NetBeans* and *Eclipse* were 56%, 48%, 56%, 71% and 80% which showed 10-15% increment for top-10 recommendation. Furthermore, the respective precision values were 25%, 21%, 25%, 32% and 25% which showed 7-10% decrement for top-10 recommendation. The results were law eventhough the datasets were large enough.

This chapter shows that there has already been a lot of attempts made in the past to automate a bug assignment process using various ML and text classification techniques. Although, it is hard to determine which approach is the most promising, mainly due to the differences in used datasets, NLP techniques for text processing, evaluation metrics and processes. Based on some of these previous findings, we chose to evaluate *SVM*, *naive Bayes* classification models. We also tried *long short term memory (LSTM)* deep learning model to compare its performance with the conventional ML techniques which was not done in previous works. For NLP, we used *lemmatization*, *postagger* and compare it with *N-gram* method, which was missing in previous works. We used *TF–IDF* as the feature extraction technique. In the next chapter, we discussed some ML and text classification basics as well as the background of the chosen techniques.

1.3 Outlook

The thesis is organized into five chapters, excluding the bibliography section. In the second chapter, ML methods and text classification are introduced, We provide the background of the classification models and techniques we used in our evaluations. It also contains the background information about the microservice architecture and docker containers. In the third chapter, the methodology of our work is explained. The forth chapter explains the results and contains a comparison of different classifiers. Finally, the fifth chapter concludes this study with a summary of our findings and give an outline of possible future work.

2 Background

Our thesis relies on several fundamentals such as machine learning, natural language processing and text classification. Furthermore, it is implemented in the micro-service architecture using a docker. In this chapter, we dive into these fundamentals to get a better intuition about their theory and inner workings. First, we will discuss the several basic principles in machine learning and text classification. Later, we learn about different classification algorithms which we used in this thesis. At last, the micro-service architecture and its benefits are discussed.

2.1 Machine Learning and Text Classification

Machine Learning is an area of computer science which teaches a computer to learn. The goal is to perform better in the future based on past experience. Learning can be anything here from performing complex tasks, to computing predictions or inferring intelligent behaviour. Past experience is some sort of observation or a data. The regular machine learning process is shown below:

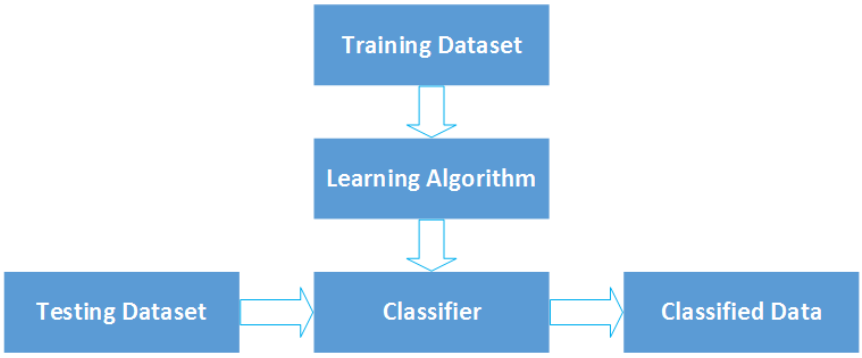


Figure 2.1.: Machine Learning Process. Training data are applied as input to learning algorithm to build prediction classifier, which is used to classify the test dataset.

As shown in Figure 2.1, the classifier model has been built using available training data and the machine learning algorithm. Once the classifier is ready, test data or new data is processed from it and will be classified or labelled accordingly. There are mainly three types of classification algorithm:

Binary Classification. When there exist only two choices/classes, it is called a binary classification. A sample can either be classified or not. The most typical example of binary classification is to predict whether an email is a spam or not.

Multi-class Classification. When there exist more than two choices/classes, it is called multi-class classification. One such a example is to classify the set of images of fruits into oranges, apples or pears.

Multi-label Classification. Set of target labels are assigned to the each document. It is possible to have multiple numbers of labels for a single document. For example, a single document could be labelled as religion, politics, or finance.

In this thesis, a multiclass classification has been used as we need to classify the tickets into seven different categories.

2.2 Natural Language Processing Methods

Natural language processing (NLP) is a way for computers to analyze, understand, and derive meaning from human language in a smart and useful way. NLP can be used to perform tasks such as automatic summarization, translation, named entity recognition, relationship extraction, sentiment analysis, speech recognition, or topic segmentation. Some of the most important NLP methods are:

Tokenization. Texts in this method are treated as a string and then chopped into the pieces called tokens. For example, the string 'winter is coming' is tokenized into 'winter', 'is', 'coming'.

Word Boundary. Remove the extra white spaces or punctuation from the text. Although the removal entirely depends on the domain and for that reason in most cases the regular expressions are used. For example, punctuation words like M.Sc. required being as it is in some cases.

Lemmatization or Stemming. The document could consist the same words in many forms such as infection, derivation etc. due to the grammatical reasons of a natural language. *Stemming* and *lemmatization* are both the natural language processing techniques to convert different word variants into similar canonical form. For example, the different form of words like 'work','works', 'worked', 'working' share a same stem 'work'.

Stopword Removing. The most commonly used words like 'a', 'the, 'of' etc. normally do not contain any meaningful information. So it is beneficial to remove these words from the texts. The list of stopwords can be created manually by adding more words to the default list.

Part-of-Speech Tagging. The *POS tagger method* reads the texts from the document and applies each word a part of speech tag like a noun, verbs etc. For example, the string 'Heat water in a large vessel'" will have the pos tags '(heat,VB), (water,NN), (in,IN), (a,DT), (large,JJ), (vessel,NN)'.

N-gram. Sometimes a group of words is more beneficial than just a single word. Here, N is the number of words in a group. N=1 is called as an unigram, which means an every individual word in a sentence. N=2 is called as a bigram and N=3 is called as a trigram. A example bigram of the string: 'winter is coming' is: 'Winter is', 'is coming'.

We used these NLP methods and compare their performance in this thesis.

2.3 Feature Extraction

Many ML algorithms like *Naive Bayes* do not scale to high data inputs. Thus, a subselection of features is crucial. A feature is a piece of valuable information with a great use for the prediction [37]. Once we finished the preprocessing, the most important task is the feature selection (FS) to get an important subset of data from the original document.
In NLP, the most widely used extraction methods are term frequency - inverse document frequency(*TF-IDF*) [38], *word2vector*[39] and *countvectorizer* [40]. The previous surveys suggest that the *TF-IDF* as weighting method was widely used and because of its simplicity and efficiency it delivered great results.[41]. Here, *TF* counts how frequently a term occurs in a document and *IDF* focuses how important the term is. If term appears in many or all the documents then it may not be a unique term. Now, *TF-IDF* is the product of term TF weight and IDF weight. It means *TF-IDF* gives most frequent but unique terms. A function for *TF-IDF* weight can be defined as $tf-idf(t)$, where t is term and d is document :

$$tf-idf(t) = tf(t,d) \times idf(t). \tag{2.1}$$

Now let us take an example to understand *TF-IDF*. We have document space as shown in first column of table 2.1. Here, d1 and d2 are documents in a training dataset, d3 and d4 are the documents in a testing dataset. After removing stopwords, the important words called as terms are shown in second column. We use the terms from training document sets and calculate the tf value as shown in column three. The last column shows the idf value for test document dataset.

Document Sets	Terms	TF Value	IDF Value
Train Document Set			
d1: The sky is blue	sky, blue	(blue, sun, bright, sky)	
d2: The sun is bright	sun, bright		
Test Document Set			
d3: The sun in the sky is bright	sun, sky, bright	[0,1,1,1]	[0.69,-0.40,-0.40, 0.0]
d4:We can see the shining sun, the bright sun.	sun, bright, sun	[0,2,1,0]	

Table 2.1.: *tf-idf* feature extraction example. First column is for documents sets used in the example. Second column is for the terms. Third column is the result of *tf* values for test document set using terms of train document set. The result of *idf* calculation for test document set is in the last column.

Now, for the training documents d_1 and d_2, we can have the index of glossary as $E(t)$ for terms t as shown in Equation (2.2).

$$E(t) = \begin{cases} 1 & if \quad t = "blue" \\ 2 & if \quad t = "sun" \\ 3 & if \quad t = "bright" \\ 4 & if \quad t = "sky". \end{cases} \tag{2.2}$$

In the Equation (2.2), the words such as "is" and "the" are not considered because they are the stop words. The term-frequency is how many times the term t occurs in the document d which can be defined as:

$$tf(t,d) = \sum_{x \in d} f(x,t), \tag{2.3}$$

here, $f(x,t)$ is a simple function which check word x in document is term t or not. It can be defined as follows:

$$f(x,t) = \begin{cases} 1 & if \quad x = t \\ \\ 0 & otherwise. \end{cases} \tag{2.4}$$

Previous two Equations (2.3) and (2.3) explain that $tf(t,d)$ returns the occurrences of specific term t in document d. For example, $tf("sun", d_3) = 1$ as term "sun" appears once in document d_3. Whereas the same word has tf value of two in document d_4. i.e. $tf("sun", d_4) = 2$ as it appears twice in document d_4. The vector can be created using this tf value as below:

$$\boldsymbol{u(d_n)} = \big[tf(t_1,d_n), tf(t_2,d_n), tf(t_3,d_n), ...tf(t_n,d_n)\big], \tag{2.5}$$

In Equation (2.5), each dimension represents the term of the glossary. For example, the $tf(t_2,d_1)$ means frequeny of the term 2 ("sun") in document 1. With use of Equation (2.5) we will have vectors for document d_3 and d_4 as follows:

$$\boldsymbol{u(d_3)} = \big[0,1,1,1\big], \tag{2.6}$$

$$\boldsymbol{u(d_4)} = \big[0,2,1,0\big]. \tag{2.7}$$

The Equation (2.6) shows occurrences of terms "blue", "sun", "bright", "sky" are 0, 1, 1 and 1 for a document d_3 respectively. The same occurrences of the terms are 0, 2, 1, 0 for a document d_4 as shown in Equation (2.7). However, we have a large amount of documents in a real time and it is easier to present them as matrix M_{tf} of the dimension $K \times F$; where K represents a number of documents and F represents a number of features. The matrix representation for document d_3 and d_4 will be as follows:

$$\mathbf{M_{tf}} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 2 & 1 & 0 \end{bmatrix}. \tag{2.8}$$

The matrix in an Equation (2.8) is a representation of the sparse matrix as there are many terms with value zero [42]. Now, for the a document space $D = \{d_1, d_2, \ldots, d_K\}$, where K is a total number of documents in the dataset, the inverse document frequency (idf) can be defined as:

$$idf(t) = \log \frac{K}{1 + |\{d : t \in d\}|}. \tag{2.9}$$

Where, $|\{d : t \in d\}|$ shows the cardinality of documents in which the term t is present, when the condition $\text{tf}(t,d) \neq 0$ is satisfied. 1 is added in the formula to avoid "divide by zero" situation. For our example, we will have $D_{train} = \{d_1, d_2\}$ and $D_{test} = \{d_3, d_4\}$. It means a total number of documents in this document space is $K_{train} = 2$ and $K_{test} = 2$. For each four terms, "blue", "sun", "bright" and "sky" in our example, the *idf* value $idf(t_1)$, $idf(t_2)$, $idf(t_3)$, $idf(t_4)$ can be calculated using Equation (2.9) as:

$$idf(t_1) = \log \frac{K_{test}}{1 + |\{d : t_1 \in d\}|} = \log \frac{2}{1} = 0.69314718,$$

$$idf(t_2) = \log \frac{K_{test}}{1 + |\{d : t_2 \in d\}|} = \log \frac{2}{3} = -0.40546511,$$

$$idf(t_3) = \log \frac{K_{test}}{1 + |\{d : t_3 \in d\}|} = \log \frac{2}{3} = -0.40546511,$$

$$idf(t_4) = \log \frac{K_{test}}{1 + \left|\{d : t_4 \in d\}\right|} = \log \frac{2}{2} = 0.0.$$

All the *idf* values can be written in a vector form as follows:

$$\boldsymbol{v} = (0.69314718, -0.40546511, -0.40546511, 0.0).$$

and the matrix representation is:

$$\mathbf{M}_{\text{idf}} = \begin{bmatrix} 0.69314718 & 0 & 0 & 0 \\ 0 & -0.40546511 & 0 & 0 \\ 0 & 0 & -0.40546511 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \tag{2.10}$$

Here the matrix multiplication is not commutative, thus the result of $X \times Y$ will be different from the result of $Y \times X$. Finally, we have value for $\mathbf{M}_{\text{tf-idf}}$ as:

$$\mathbf{M}_{\text{tf-idf}} = \mathbf{M}_{\text{tf}} \times \mathbf{M}_{\text{idf}}, \tag{2.11}$$

Where the matrix for $\mathbf{M}_{\text{tf}}$ and $\mathbf{M}_{\text{idf}}$:

$$\mathbf{M}_{\text{tf-idf}} = \begin{bmatrix} tf(t_1,d_1) & tf(t_2,d_1) & tf(t_3,d_1) & tf(t_4,d_1) \\ tf(t_1,d_2) & tf(t_2,d_2) & tf(t_3,d_2) & tf(t_4,d_2) \end{bmatrix} \times \begin{bmatrix} idf(t_1) & 0 & 0 & 0 \\ 0 & idf(t_2) & 0 & 0 \\ 0 & 0 & idf(t_3) & 0 \\ 0 & 0 & 0 & idf(t_4) \end{bmatrix},$$

Putting values from Equation (2.8) and Equation (2.10) in the above equation, we can have:

$$\mathbf{M}_{\text{tf-idf}} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 2 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 0.69314718 & 0 & 0 & 0 \\ 0 & -0.40546511 & 0 & 0 \\ 0 & 0 & -0.40546511 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$\mathbf{M}_{\text{tf-idf}} = \begin{bmatrix} 0 & -0.40546511 & -0.40546511 & 0 \\ 0 & -0.81093022 & -0.40546511 & 0 \end{bmatrix}. \tag{2.12}$$

In mathematics, the length of all vectors is defined as *norm*. Here, we will not apply normalization on the entire matrix at once. We will apply it to each row. The result after applying L2 norm on Equation (2.12) is:

$$\mathbf{M}_{\text{tf-idf}} = \frac{\mathbf{M}_{\text{tf-idf}}}{\|\mathbf{M}_{\text{tf-idf}}\|_2} = \begin{bmatrix} 0 & -0.70710678 & -0.70710678 & 0 \\ 0 & -0.89442719 & -0.4472136 & 0 \end{bmatrix}. \tag{2.13}$$

Equation 2.13 shows the normalized *tf-idf* values of the documents d_3 and d_4 (test set), which are literally set of unit vectors. In this thesis, The tf-idf method is used for feature selection.

2.4 Classification Algorithms

Machine learning algorithms can be broadly classified into following main categories.

Supervised Learning. The algorithm implied with a set of input along with the desired output(labels) and it learns to predict the correct output for a new input data. For example, for the fruit description as an input and fruit category (banana, apple, orange) as an output. This technique can be used with the methods like classification or regression. Some of the popular supervised techniques include *decision tree induction*, *naive Bayesian classification*, and *support vector machines*.

Unsupervised Learning. The algorithm must figure out the output by itself as no outputs are provided to it in this method. The purpose here is to explore the data and find out a pattern inside the data. This technique can be used with the methods like clustering or association. Some of the popular unsupervised techniques include *nearest-neighbor mapping*, *self-organizing maps*, *k-means clustering* and *singular value decomposition*. This technique is used to recommend items and segment text topics.

Semi-supervised Learning. This technique used both a small amount of labelled data and a large amount of unlabelled data as a training set. It is mostly used when a cost of labelling data is too high. For example, identifying the face of a person.

Reinforcement Learning. There are three main components in this learning process: the agent (the learner), the dynamic environment (everything the agent interacts with) and goals (action performed by an agent). The goal of this method is to learn the best policy. The algorithm learns that by trying repeatedly. This technique is used in the areas like robotics and gaming.

In this thesis, the supervised algorithms have been used as we already knew the prediction classes.

2.4.1 Naive Bayes Classifier

The *naive Bayesian classifier* was first described by [43] in 1973 and then by [44] in 1992. It is a generative linear model based on a *Bayes' theorem*. It relies on a simple assumption that all the features are independent of each other. For example, if a fruit is orange in color, round in shape and has 2.5 inches diameter then it can be classified as an orange even all those features are independent of each other and each commit independently. Although, this assumption is not always right but still it performs good in most of the practical scenarios [45].

In the context of a text classification, the probability that random text T belongs to a class C can be expressed as $P(C|T)$, where T is any of the values $\boldsymbol{x_i}$ where $i = 1, 2, 3, ..., L$ and C can take the values $\boldsymbol{y_j}$ where $j = 1, 2, 3, ..., M$. It can be calculated using *naive Bayes* as below:

$$P(C|T) = \frac{P(C)P(T|C)}{P(T)}. \tag{2.14}$$

Where, $P(C)$ is the prior probability of C, $P(T)$ is the prior probability of T, $P(C|T)$ is posterior probability of C given T, $P(T|C)$ is the likelihood which is the probability of T given C . Now, each text can be vectorized by the terms it has $\boldsymbol{x} = (x_1, x_2, ..., x_K, ..., x_L)$ [46]. Then the above equation can be interpreted as:

$$\begin{aligned} P(C|T) &= P(C|\boldsymbol{x}), \\ &= P(C|x_1, x_2, ..., x_L), \\ &= \frac{P(C)P(x_1, x_2, ..., x_L|C)}{P(x_1, x_2, ..., x_L)}, \end{aligned} \tag{2.15}$$

When leaving the denominator out, Equation (2.15) can be transformed as below:

$$P(C|T) \approx P(x_1|x_2, ..., x_L, C)P(x_2, ..., x_L, C),$$

Considering the assumption that features are independent for the given class the above equation can be further represented as:

$$P(C|T) \approx P(C)\prod_{k=1}^{L} P(x_k|C). \tag{2.16}$$

For a testing document, we can find the class corresponding the biggest $P(C|T)$ value among the n possible ones. It is also called maximum a posteriori or *MAP decision rule* [46]:

$$C_{map} = \underset{x \in C}{\operatorname{argmax}} P(C) \prod_{k=1}^{l} P(x_k|C) \tag{2.17}$$

$P(C)$ is the prior probability of class which can be calculated as follows:

$$P(C) = \frac{\#\ of\ instances\ in\ this\ class}{\#\ of\ instances\ in\ all\ classes}.$$

The second part of Equation (2.14), $P(T|C)$ is the likelihood which is the probability of predictor given class. It matters a lot in class label decision. From the various ways of calculating this value, we discuss *multinomial* and *Bernoulli naive Bayes* which are commonly used ways for document classification.

Bernoulli naive Bayes model is good when the feature vectors are binary (either 0 or 1) [46]. It is a simple model which only checks weather a feature occurs in a document or not. So calculation of $P(T|C)$ in equation 2.14 for the *bernoulli naive Bayes* model can be expressed as below:

$$P(T|C) = \prod_{k=1}^{l} P(x_k|C)^b (1 - P(x_k|C_j))^{1-b}, (b \in 0, 1),$$

$$P(x_k|C) = \frac{df_{x_k,C} + \alpha}{df_C + \beta}.$$

where, α and β are the smoothing parameters, $df_{x_k,C}$ is a number of documents in the training set which contain the feature x_k and belong to the class C, df_C is a number of documents in the training set which belong to class C. *Multinomial naive Bayes* model checks how often the term occurs in a chosen metric instead binary representation like *Bernoulli naive Bayes* [46]. So calculation of $P(T|C)$ in equation 2.14 for multinomial naive Bayes model can be expressed as below:

$$P(T|C) = \frac{\sum tf(x_k, d \in C) + \alpha}{\sum N_{d \in C_j} + \alpha V}.$$

where, α is a smoothing parameter, $\sum tf(x_k, d \in C)$ is the sum of raw term frequencies of word x_k from all documents in the training set that belong to class C, $\sum N_{d \in C}$ is the sum of all term frequencies in the training set for class C, V is a total number of unique terms in the training set.

In many instances, *multinomial naive Bayes* outperforms *bernoulli naive Bayes* when the vocabulary is quite big [46]. In this thesis, *multinomial naive Bayes* is used for multiclass classification.

2.4.2 Linear Support Vector Machine

Support vector machine (SVM) is an excessively powerful and popular supervised learning approach in the field of machine learning algorithms. *SVMs* were natively introduced by Vladimir Vapnik and colleagues but the first main paper was published in 1995 by Vapnik and Cortes [47]. *SVMs* can be used for both classification and regression problems though it is mostly used for classification problems. Text classification usually has a huge vocabulary so *SVMs* are really good choice. *SVMs* are applicable for both linear and nonlinear classification problems. Polynomial kernel, Gaussian kernel and sigmoid kernel are widely used kernels in *SVMs*. We will only discuss the *linear SVM* as texts are linearly separable. [38].

The main concept of *SVMs* is to find a hyperplane that divides a dataset into the different classes in a best way. *SVM* models the situation by creating n-dimensional feature space, where each dimension represents a "feature" of a particular object. In the text or document classification, each feature is a prevalence of a particular word.

For linearly separable training examples $(\boldsymbol{x_i}, c_i)$, $(i = 1, 2, ..., L)$, where each example has d inputs $(x_i \in \mathbf{R^d})$ and its correlative class label $c_i \in \{-1, 1\}$. We can find a hyperplane based on the training set which satisfies the following equation:

$$\boldsymbol{w}\boldsymbol{x} + b = 0. \tag{2.18}$$

where $\boldsymbol{w}$ is a vector orthogonal to the hyperplane and b is a perpendicular distance of the hyperplane to the origin. [48]. we can find the canonical hyperplane to separate data from hyperplane by at least distance of 1 and it will satisfies the following condition:

$$\begin{cases} \boldsymbol{w}\boldsymbol{x_i} + b \geq +1 & \text{for} \quad c_i = +1 \\ \boldsymbol{w}\boldsymbol{x_i} + b < -1 & \text{for} \quad c_i = -1 \end{cases}, \tag{2.19}$$

or more compactly:

$$c_i(\boldsymbol{w}\boldsymbol{x_i} + b) \geq 1 \quad \forall i \ . \tag{2.20}$$

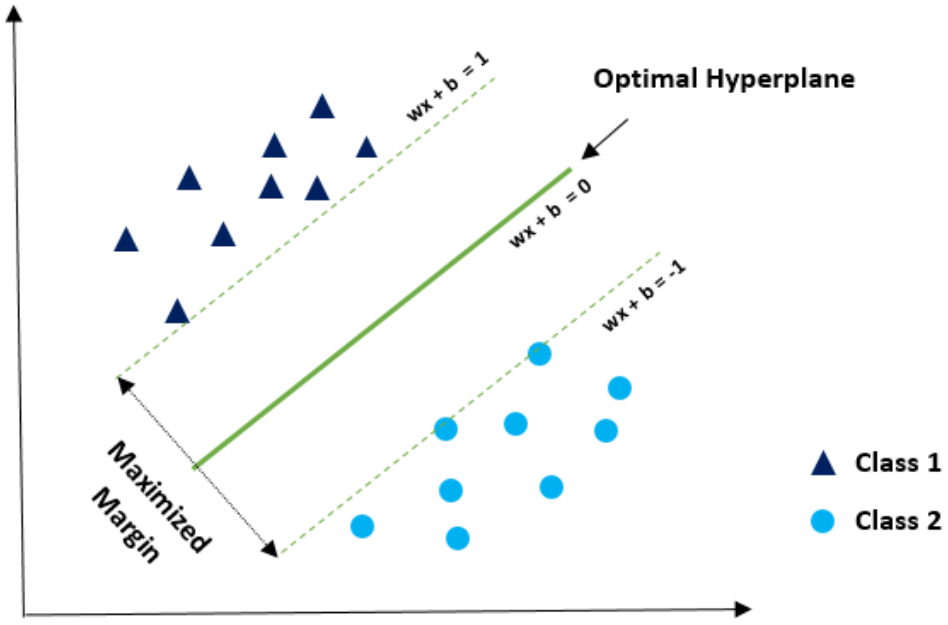


Figure 2.2.: Support vector machine-hyperplane with feature vectors for two classes. Green line represents a optimal hyperplane with maximized margin between two classes.

The mentioned function is to classify the text into positive or negative classes but as shown in Figure 2.2 there are multiple possible hyperplanes. It is required to find a maximum possible distance between two support vector to find the best possible classifier, which is also called as a *margin*. we must normalize by the magnitude of $\boldsymbol{w}$ to obtain this distance:

$$d((\boldsymbol{w}, b), \boldsymbol{x_i}) = \frac{c_i(\boldsymbol{w}\boldsymbol{x_i} + b)}{\| \boldsymbol{w} \|} \geq \frac{1}{\| \boldsymbol{w} \|} \tag{2.21}$$

SVMs were mainly proposed to deal with binary classification but nowadays, we mostly have a huge amount of data with more than two categories. So there is a demand of multi-class classification model with *SVMs*. There are numerous approaches[49] available however, the most frequent approaches are *One-vs.-Rest* method and *One-vs.-One* method.

One-vs.-Rest is also sometimes called as "one-versus-all" or "OVA" classification method. In this approach, each category is splitted out and all the remaining categories are merged to choose classifier with the greatest margin. It divides n class problems into n binary problems. In training the *i-th* binary *SVM*, samples belonging to the *i-th* class are used as the positive class samples, and all other training samples are used as the negative class samples.

One-vs.-One is also sometimes called as "one-versus-one" or "OvO" classification method. It divides n class problems into $\binom{n}{2} = n(n-1)/2$ binary problems. For any pair (i, j) satisfying $1 < i < j < n$, we can use the samples in the *i-th* class as positive ones, and samples in the *j-th* class as negative samples. For any test sample x, the comparison of each class is done one by one with all the remaining classes and votes are counted. The class with largest amount of votes is expected

to be the right class. If the real class of x is k $(1 < k < m)$ then we expect the k-th class will receive the largest amount of votes.

For small number of classes, OvA and OvO methods behave almost same in terms of accuracy so both the methods are applicable. However, OvO method requires excessively long time to train the classifier with high number of classes as number of binary classifier will be very large. For example, 1000 classes require 499,500 binary *SVM* classifiers. In this thesis, the *One-vs.-Rest* classifier is used for multi-class classification.

2.4.3 Long Short Term Memory

Long Short Memory Networks (LSTM) are a type of Recurrent Neural Networks (RNN), which were first introduced by Hochreiter & Schmidhuber (1997). *LSTMs* work as human memory function. Humans do not start to understand anything from the beginning instead use the previous knowledge. For example, every word of this thesis can be learned based on previous knowledge of words. The diagram for *LSTM* architecture is shown in Figure 2.3.

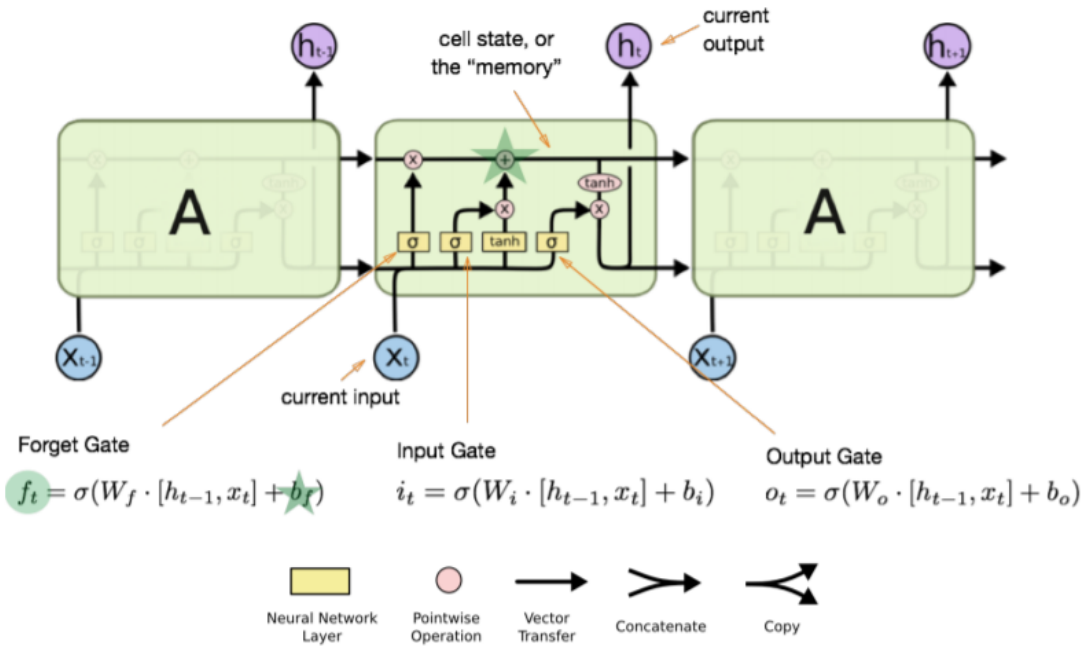


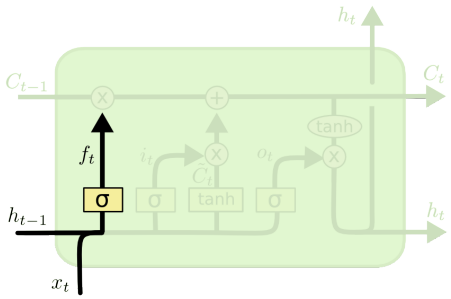
Figure 2.3.: An *LSTM* Network with memory block. It consists of input gate, forget gate and ouput gate. Sigmoid and tanh functions are used by the gates to determine output from the given input. [2]

Where, h_{t-1} is the previous hidden state, X_{t-1} is the previous output, X_t is the current output, X_{t+1} is the next output, W is a weight of matrix, and b is a bias.

LSTMs consist of memory cells, a horizontal line running through the top of the diagram, along with multiplicative gates which allow to store and access the formation for a long time. *LSTMs* can add or remove information to a cell using a carefully regulated structure called gates. There are three gates for this purpose which are a input gate, an output gate and a forget gate. These gates use the sigmoid functions to decide the amount of information to flow. Sigmoid function output in a range of [0,1] where 1 means to allow everything and 0 means allow nothing to pass. Every gate has an own weight which can be adjusted via gradient descent. The four layers of *LSTMs* can be explained in details as follows:

Forget Gate

It is not required to keep all the information in the network. For example, while moving from one music piece to next music piece into a dataset, unnecessary information from old music piece might be forgettable. *LSTM* has a forget gate which uses the sigmoid function between last hidden layer (h_{t-1}) and current input (X_t) to decide the information to forget from the network.

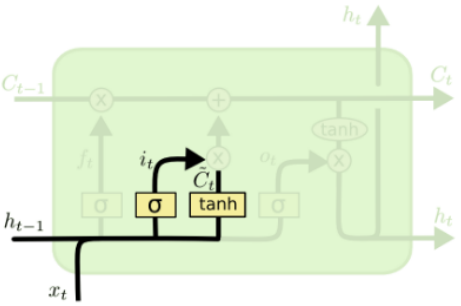


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 2.4.: *LSTM* Fotget Gate to determine unnecessary information using sigmoid function between last hidden layer (h_{t-1}) and current input (X_t) [2].

Input Gate

The next step is to determine the new information to store into a network which is done in two parts. First, it decides the amount of information to store using sigmoid layer. Second, tanh layer which creates a vector of new candidate values C_t to be added to the state.

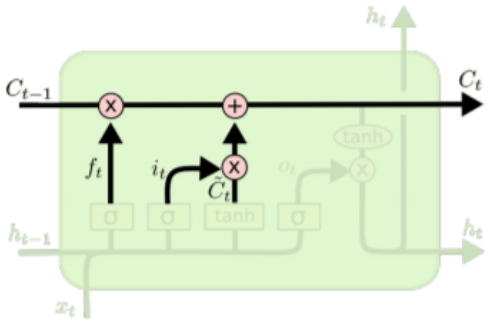


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figure 2.5.: *LSTM* Input Gate to determine information to used as input. It uses sigmoid layer to determine amount of infomation and tanh layer to create vector of that information [2].

Updating the Cell Memory

Until now the information to forget and the information to add into memory cell are determined. However, it is not being actually changed the memory cell. In this step we update old cell state C_{t-1} by multiplying it with vector t_t and then adding it to $i_t \cdot C_t$.

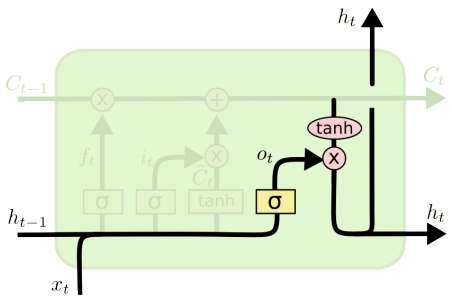


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 2.6.: *LSTM* update old memory cell with the information detemined in previous steps [2].

Output Gate

The last step is to determine an output from the memory cells. To do so, first, sigmoid function will be applied to the previous hidden state (h_{t-1}) and current input (C_t). Later, this result will be multiplied with the tanh (to force the values to be between -1 and 1) of cell state.



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 2.7.: *LSTM* output gate gives the output of memory block which is uesd as a input for next memory block [2].

LSTM has an actual memory built into the architecture where cell memory updated with new information ($i_t \cdot C_t$) by addition. It makes *LSTM* to maintain a constant error when it must be back propagated at depth [50]. Instead of determining the subsequent cell state by multiplying its current state with the new input, the addition prevents the gradient from exploding or vanishing [50].

2.4.4 Performance Evolution

This is the last step of classification process where we experimentally measure the performance of a text classifier. This is a very important step before we apply our model to a large dataset. The experimental evolution mostly focuses on the effectiveness of a classifier, i.e its decision capability for right category. Some of the terms which are used for a measurement are:

True Positives (TP): It is a calculation of the results which are correctly identified. For example, the percentage of bug reports which are correctly classified to the right team.

True Negatives (TN): It is the calculation of the incorrectly assigned tickets to this team.

False Positives (FP): It is the case when actual results are negative, but the system will provide positive results. Sometimes it is also referred as a "false positive error". For example, Anti-virus software detects a program as malicious one, although it is a harmless software.

False Negatives (FN): It is the case when negative results are wrong. In other words, you actually have positive results for the test, but the system gives you negative test results for the test. For example, the case when the anti-virus software fails to detect a virus.

Some of the most frequent measures for classification evolution are:

Accuracy. *Accuracy* is the most commonly used performance measure. It is the ratio of correctly classified documents to a total number of documents. Its formula is defined as [14] [51]:

$$(TN + TP)/(TN + TP + FN + FP).$$

Precision. *Precision* is a ratio of the correctly assigned tickets to the total inspection. Here, total inspection in the denominator is the sum of all true positives and false positives. Its formula is defined as [14] [51]:

$$(TP)/(TP + FP).$$

Recall. It is also called as *sensitivity*. It is the ratio of the truly predicted positive case. Its formula is defined as [14] [51]:

$$(TP)/(TP + FN).$$

F1 Score. *F1 score* is defined as a weighted mean of a *precision* and *recall*. It takes into account both false positives and false negatives. *F1 score* is more convenient than *accuracy* in case of unbalanced class distribution. Its formula is defined as [14] [51]:

$$2 * (Precision * Recall)/(Precision + Recall).$$

There are also some other measures used like *break–even point* and *interpolation*. In this thesis, we used *accuracy*, *precision* and *recall* to compare a performance of the different classification algorithms.

2.5 Microservice Architecture

Microservices are small autonomous services which work together to fulfill a business requirement [4]. In this section, we will discuss the microservices and will compare its beneficiary with monolithic architecture. The application with monolithic architecture has only one deployment unit, one codebase and one technology stack. It has some benefits such as a single unit of access to coding, building and deploying which provides a simple model for developers. Furthermore, a simple model for scaling by just running multiple copies behind the load balancer. But monolithic architecture has several limitations for larger applications such as:

- It has huge code base for developers which is very hard to handle. It becomes even larger when new features are added.
- There is a huge burden on development tools because of the refactorings, builds, tests which will take more time for whole application.
- Scaling is limited as creating multiple copies of the whole software consume lots of resources.
- Deployment frequency is limited as the whole system needs to be halted for it.

Microservice approach is a solution to most of these problems in monolithic architecture. Microservices can be defined as " Decomposition of a large application into small, autonomous and independently deployable components ". Microservices provide us more freedom to make different choices and capability to respond faster to changes.

The term "micro" itself refers to smaller service that must be manageable by the single development team(4-8 developers). Each service treated as an independent application with its own code base. It will be fine-grained and high cohesion. It has a resource representation and a data management. Each service represents one resource, for example, clients, carts, checkout, shop items etc. Loose coupling between each service is an important characteristic of the microservices. Each service is a separate entity and communications between these services should be using network calls. So service exposes an application programming interface (API), which is used by other services for communication. To model right service and obtaining the right APIs are keys to manage loose coupling. The key characteristic of microservice architecture are as follows:

Codebase
An own code base for each of the services is smaller which is easily maintainable by the developers. Tools for refactorings, builds, tests work fast on the smaller size. It takes only a few seconds to start services. There is no chance for an accidental cross-dependencies.

Technology Heterogeneity
With an application composed of multiple services, each service can have its own technology stack. It gives a freedom to choose a technology suitable to the task and based on the priority of the developers. So most optimal technologies and skills of the team can be utilized very well. Developers can experiment with new technology within single services as it will not affect the whole application. For example in a social network, we may use document-oriented database for posts and graph-oriented database for user interactions as shown in Figure 2.8.

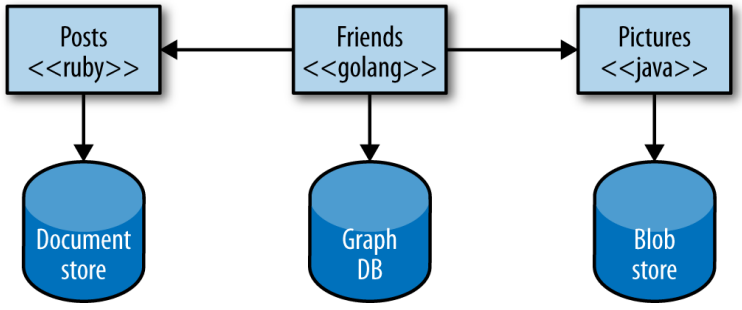


Figure 2.8.: Language-neutral in building microservices. Different database technologies such as *document*, *graph* or *blob* can be used for diferent features such as posts, friends or pictures. [3]

Improved Resilience
In microservice architecture, the failure of one service will not stop the whole running application. The failed service can be easily isolated leaving the rest of system to work with degraded functionality[3].

Optimizing for Replaceability
For a large monolithic application, it is very hard and risky to do any changes in implementation as it will affect whole application. However, services are small in size for micro services, it can be easily replaced by a new implementation or even can be deleted easily. When codebase in the services has only few line of code nobody gets emotionally attached. The service replacement is also a cost efficient.

Better Scaling
Each service can also be scaled independently according to the application requirement. It enables an efficient resource management as resources need to be provided only for the services required scaling instead for a whole application. The other services which do not represent bottleneck can remain simple and un-scaled.

Ease of Deployment
The services can be extended without affecting other services, which makes deployment process faster. For example, user interface (UI) can be updated without redeploying the whole system. If there occurs any problem, that service can be isolated immediately via making a fast rollback.

Improved Reusability of Components
The microservice architecture also provides key promises of functions reusability same as service-oriented architectures. Each service can be reused for the different purposes with different services. Services can be also shared among the projects.

We chose to use microservice architecture in our thesis because of all those benefits. To build such a architecture, we used docker which is explained in following subsection.

2.5.1 Docker

Docker is an open source platform which provides an ability to develop, ship and run software applications quick and efficiently. Docker uses client-server architecture as shown in figure 2.9. It uses set of commands from client component and interact with the docker host via docker daemon running on a host operating system.

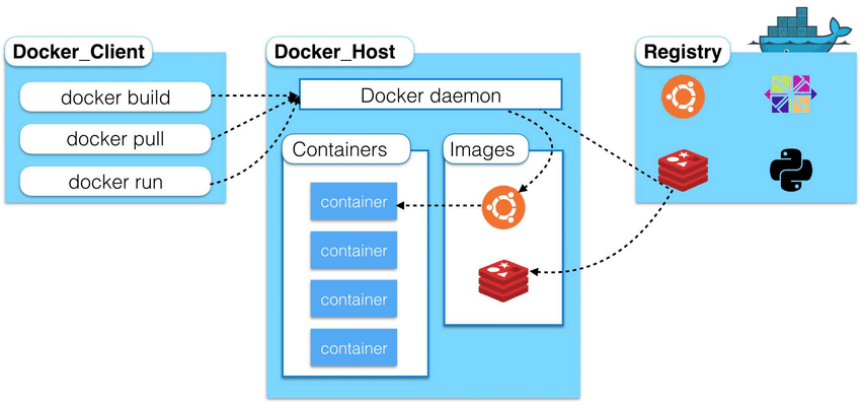


Figure 2.9.: Docker architecture overview. Client interact with docker host via docker daemon. Docker images are stored in docker registery. [4]

The concept of a lightweight container virtualization helps developers in deploying and managing application easier in microservice architecture [4]. Let us take an example of the ship containers, which are loaded into a ship, then shipped and unloaded to the different destination according to the requirements. Similarly, in docker containers, we can create an application, place them together with all the dependencies and can run inside a hardened box. A container can be created locally and stored as a docker image so that docker containers could be launched anywhere. Docker image is basically a read-only template which can be easily stored or transferred to other machines. Docker images are stored on shared repositories called docker registries, which makes a contribution of the docker images very easy and convenient. Docker Hub is one of the most famous docker repositories where people can download the previous popular images or upload own new images. In this thesis, we used a docker to build microservice architecture for our application.

3 Learning of Bug Tracking Classifiers

In this chapter, a detailed architecture of our prediction framework is presented. We explain the bug report, the processing of the raw data, the chosen feature vectors on the dataset and the prediction model using machine learning algorithms.

3.1 Problem Formulation

A bug report or a ticket is called a problem management report (PMR) in IBM . In this thesis, PMRs were used only from a business process management (BPM) product of IBM. Each BPM PMR involves only one product issue and consists of several fields: headers (preferred contact method, customer full name, telephone or mobile phone number, email address), problem details (product or service name, component id, operating system, problem title, problem description and business impact) and solutions provided by IBM software support engineers. All the PMRs are in the form of a plain text. The majority of PMRs are in English and very few are in Russian, Spanian, Chinese, Japanese, German etc. A sample PMR is shown the Figure 3.1.

```
*** Electronic submission by customer via SR tool, version 3.4.1
*** Customer problem tracking number: XX
*** Preferred contact method: IBM Service Request (SR) notification.
*** Customer contact full name: XX
*** Telephone: XX
*** Email: XX@XX.com

.
Problem Details
.
Product or Service: Business Process Manager Standard 8.5.0
Component ID: 5725C9500
.
Operating System: Linux
.
Problem title
Team Performance Report want to Override and remove "At Risk" category
from Report
.
Problem description
In implementing a recent application where many historical legacy
referrals/cases from a previous manual process were imported into the
system, this has caused the "Ave Time to Completion" calculation to be
skewed related to a previous process which was less efficient .

If there a way to override this calculation, or to remove it from the
report?  The "at Risk" cases is not accurate and provides inaccurate
values related to the current task SLAa and processes.
.
Business impact ( BusImpact )
It is causing the Team Performance reporting to not be used by
management, and giving a poor impression on the inability to override
or remove this erroneous information.  It also does not allow management
to share this info with others.
.
*** --- FOR SR USE ONLY ---
*** XRQXSRprNode5771446292888359
*** CAG400037 ENG Y
*** MTS
*** route=BPMSRID09:Coaches / Human or Ajax services
*** 5725C95:850/5725C9500:850 (ENT)
```

Figure 3.1.: Problem Management Record - 00151,442,000

Each PMR is identified by unique PMR-number. For example, the PMR in Figure 3.1 has PMR-number 00151,442,000. Here, last three digits 000 represent the country code. The middle three digits 442 represent a department within IBM. First five digits are an unique number automatically created by IBM PMR portal. For confidential issues, personal information about customers is kept non-public. In order to close a PMR, support engineers must reply customers at least once.

3.2 Overview of the Framework Architecture

In this section, we introduce the design pipeline of our framework and the detail description of each module in the framework. The application was divided into several separate modules to build a microservice architecture. Each container of the architecture is as shown in Figure 3.2 as filled disk.

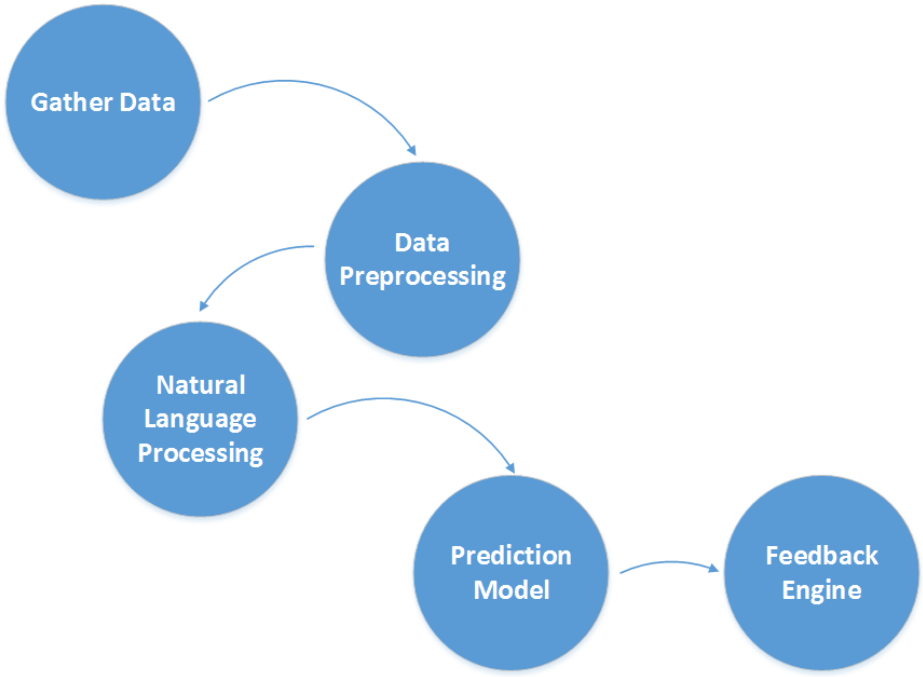


Figure 3.2.: Docker containers for bug tracking tool. It contains five containers i.e, gather data, data preprocessing, natural language processing, prediction model and feedback engine.

As shown in Figure 3.2, our framework consisted of five main containers. In the 'gather data' container, all the PMRs are gathered from IBM to use them as a dataset for our prediction model. Preprocessing was done on the gathered data such as removing duplicates, finding only closed PMRs etc in the second container. The third container was for natural language processing, where different NLP methods such as *stopword removal*, *lemmatization*, *pos tagger* and *bigram* were implemented. In the fourth container feature vectors were created from the text using *TF-IDF* to build a classification model. *Linear support vector machine*, *naive Bayes* and *long sort term memory (LSTM)* algorithms were implied to build prediction model and they were compared in order to obtain the best possible model. Finally, a feedback engine container was developed to get a real feedback from the IBM support engineers. In the following subsections, the functionality of every container is described in detail.

3.2.1 Gather Data

A dataset from IBM BPM product was employed to build a prediction model. Therefore primarily, a dataset was required to be prepared by gathering all the PMRs. When customers encounter any issues in the BPM product, they write a PMR to the IBM support department. Once a PMR is resolved by the support engineers, it is marked as a "closed". Seven different teams work in BPM product support to resolve the PMRs. Every team is responsible for a specific task and identified by the queue name as shown in Table 3.1.

So far, there is no repository which stores all the PMRs to use it directly as a dataset for our prediction model. Although, IBM has developed a java api called "retain api" which stores the PMRs until they are open and for few days once they are resolved. However, after few days, resolved PMR is removed from the retain api and there would not be any access

Queuename	Functionality
BPM2CH	Coaches component
BPM2CM	Installation and configuration
BPM2CO	Advance component, performance, cpu or memory, adapter
BPM2MO	Monitor component
BPM2PC	Process designer, design process
BPM2RT	BPM standard,BPM notation, UML lagugage configuaration
BPM2UE	User Interface or design

Table 3.1.: Functionalities of teams working on BPM product.

to it. A container was developed that gathers all the currently available PMRs and stores them into a specific directory. A folder was created for every team with a queuename then specific PMRs were stored into a specific folder. Everyday, the container runs continuously and gathers the new data. At the time the prediction model was learnt, there were in total 23597 PMRs.

3.2.2 Data Preprocessing

All the PMRs were gathered and stored into a directory. However, an improvement of the dataset was essential and some preprocessing was required before using it for the prediction model. The steps for the process are shown in Figure 3.3.

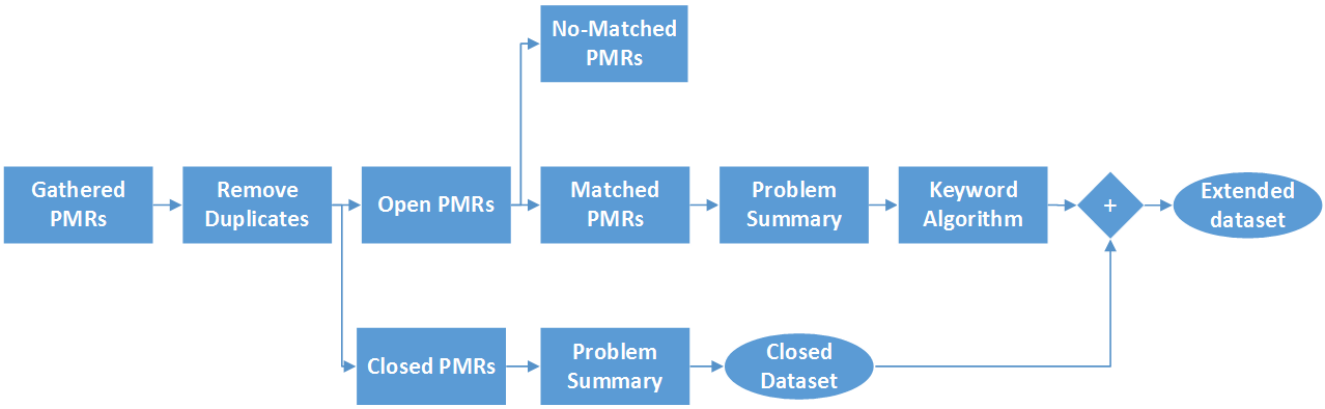


Figure 3.3.: Data preprocessing. The execution of a process on gathered data results in 'closed' and 'extended' dataset.

The gathered data contained several duplicate PMRs. Let us take an example to understand it better. A customer has an issue and he writes a PMR to support department which is forwarded to BPM2CH team by IBM PMR portal. Support engineer from BPM2CH would read the PMR and if he is not responsible for this PMR, he adds a reason and forwards it to an another queue (let us say BPM2UE) according to his knowledge. Now, the duplicate PMRs are in BPM2CH and BPM2UE queues. Therefore, only the latest PMR which is in BPM2UE queue is kept and the PMR in BPM2CH queue is removed considering it as a duplicate.

Closed Dataset

In next step, open and closed PMRs are separated. The string "$bpm\_pmr\_closure\_template\_end$" is added by support engineer when they resolve the PMRs. The existence of string gives an assurance of closed PMRs. However, those PMRs contained some extra content such as text added by support engineers while resolving or forwarding PMRs to another team along with useful content such as problem details, problem description. The whole text content of PMRs was not employed to build a prediction model. Whenever a customer writes a bug report for the first time, there will be only the problem title and the problem description. Both are defined as problem summary for simplicity. Such closed PMRs with only problem summary part was our first dataset, which is referred as 'closed dataset'.

Extended Dataset
In the beginning, the prediction model was built using only the closed PMRs but the number of PMRs were not enough to get good performance. So keyword algorithm was used to gather some more PMRs. If IBM portal could not find any queuename for the PMR from its content then it was forwarded to any random queue. Such type of PMRs are called as 'No-Matched' PMRs. First, 'No-Matched' PMRs and 'Matched PMRs' were separated from the remaining open PMRs. IBM portal adds "no routing done" string while doing such a random routing which was used to separate the 'No-Matched' PMRs. Then problem summary part was fetched for the 'Matched' PMRs and keyword algorithm was applied on them. The list of keywords for each queue was provided by experienced support engineers. The task of the algorithm was to count the frequency of keywords for each queue and to choose the queuename with highest counts. For example, if there are three keywords for BPM2CH and two keywords from BPM2UE, PMR with the highest number of keywords would be chosen i.e., BPM2CH queue. In the case, there are the same number of keywords for two queues then any one of the random queuename would be chosen. For example, PMR contains two keywords for both BPM2CH and BPM2UE queue then randomly either BPM2CH or BPM2UE being chosen. PMR is not taken into account if no keyword can be found or only one keyword found. Keyword algorithm obtained around 6754 PMRs which were added to 1215 closed PMRs. Instead of 7969 (6754+1215) PMRs, total 7346 PMRs were obtained in total because the common PMRs which were already present in a closed PMRs directory was not added. The dataset with those PMRs referred as 'extended dataset'.

Queuename	Gathered Data	Closed Dataset	Extended Dataset
BPM2CH	1515	96	387
BPM2CM	3651	167	1597
BPM2CO	4478	208	1209
BPM2MO	1214	56	247
BPM2PC	3706	143	1144
BPM2RT	5840	379	1689
BPM2UE	3193	166	1073
total	23597	1215	7346

Table 3.2.: Number of PMRs for each queue in gathered data, closed dataset and extended dataset.

As shown in the table 3.2, preprocessing container reduced 23597 PMRs to only 1215 PMRs for closed dataset and 7346 usable PMRs for extended dataset. Prediction model was built on both the dataset to choose dataset with better results.

3.2.3 Natural Language Processing

Our dataset contained only essential PMRs for each queue after preprocessing. Moreover, only the problem summary part of the PMRs was used. Direct use of the dataset to build prediction model did not lead to achieve promising results. Therefore, Natural Language Processing (NLP) methods were applied to remove unnecessary text from the PMRs. Two different approaches of NLP were used as shown in Figure 3.4.
For both approaches, first the text needs to be tokenized by converting all text strings to single word. Later, the punctuation characters such as 'l', '#', '$' were removed.

In the first approach, referred as the 'nlp1' approach, the *N-gram method* was applied. As explained in Subsection 2.2, a *N-gram* can be an *unigram*, a *bigram* or a *trigram* depending on the number of words we use as a pair. Here, a *bigram* was employed as it outperforms the *unigram* and *trigram* methods. However, the stopwords were removed from the text before applying *bigram*. A list of stopwords was created manually to add any extra words in a list as per requirements. For example, IBM, BPM words were very common in PMRs.

In the second approach, referred as the 'nlp2' approach, the *pos tagger* and the *lemmatization* methods were used. In the *pos tagger* method, each word was assigned in a grammatical form such as noun, verb, adjective etc. and by filtering, only the noun words and verb words were kept. Later, the *lemmatization* method was applied and each form of words were converted into their canonical form. At the end, the *stopwords removal* method was employed to remove all the stopwords in the same way as done in the nlp1 approach.

Here, the order of applying the *stopword removal* method was really important. In the case of the nlp1 approach, it was applied before using the *bigram* method. Whereas in the case of the nlp2 approach, it was applied at the end. The

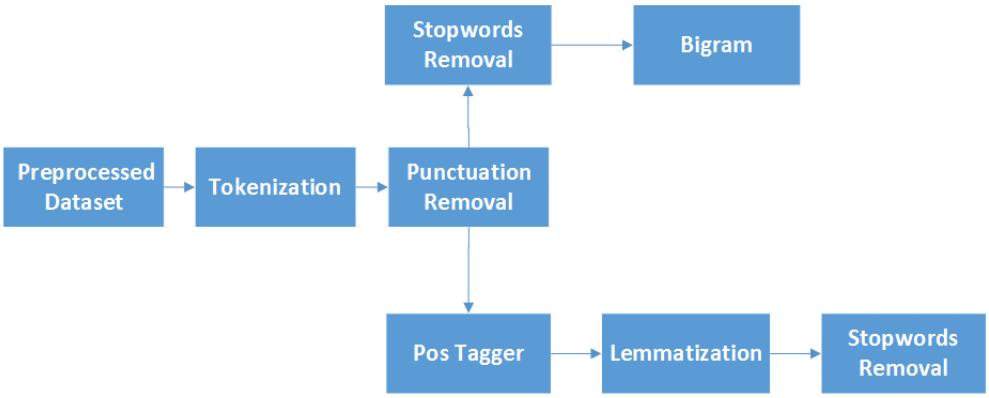


Figure 3.4.: Natural language Processing. Texts are tokenized and punctuation symbols are removed. In the first approach termed as 'nlp1', stopword removal and bigram method were used. In the second approach termed as 'nlp2', pos tagger, lemmatization and stopword removal method were used.

reason is that lets say, there are words such as "IBM", "IBMs", "IBM's" in the PMR and *stopword removal* is applied before *lemmatization* then it removes only "IBM" but not "IBMs" and "IBM's".

Both approaches, nlp1 and nlp2, were compared to find best suitable for BPM dataset and according to the obtained results, the nlp1 approach was considered to be slightly better than nlp2 approach with respect to the performance.

3.2.4 Prediction Model

Three different classification algorithms were used to learn a prediction model. The name of these algorithms were *support vector machines (SVM)*, *multinomial naive Bayes* and *long short term memory (LSTM)*. For the classification tasks, the dataset was randomly splitted into two parts- "training dataset (79%)" and "testing dataset (21%)". For closed dataset, 1215 PMRs were randomly splitted into 959 and 256 PMRs for training and testing dataset respectively. Out of 7346 PMRs of the extended dataset, randomly chosen 5804 PMRs were in training dataset whereas the remaining 1542 PMRs were in testing dataset. The training dataset was used to train the model. The testing dataset was used for model assessment and estimating the prediction error on a new data.

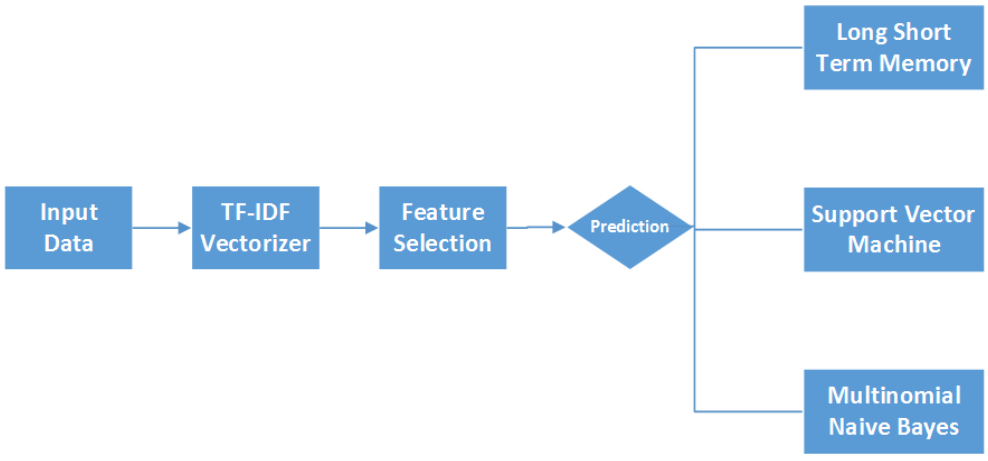


Figure 3.5.: Prediction model. Important number of features are fetched from the text using *TF-IDF* vectorizer method. Three classification algorithms, *LSTM*, *SVM* and *naive Bayes* are used to build prediction model.

Before applying a dataset to the classification algorithms, normal text was needed to be converted into vector form therefore, *TF-IDF vectorizer* technique was applied to create such a feature vector. IDF value in the method was 8, which means that words which appeared in more than 80% of the PMRs were removed from the list. The number of features were too high after applying the *TF-IDF* method. So certain amount was reduced in *TF-IDF* to get better results and to reduce prediction time.

3.2.5 Feedback Engine

In the last container in Figure 3.2, web framework was implemented to collect feedback about the prediction model from the BPM support engineers. A screen shot of the web page is shown in Figure 3.6.

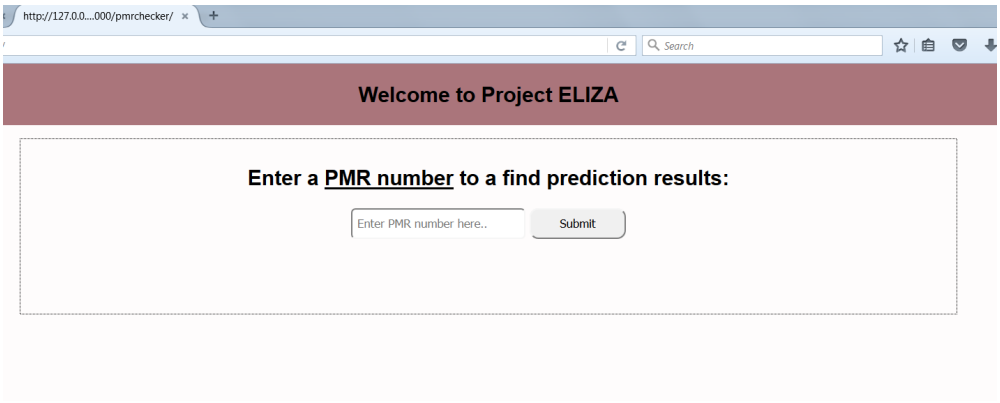


Figure 3.6.: Feedback engine web framework. Support engineers can enter the PMR-number to check prediction results.

The IBM support engineers can enter a PMR-number and check the results of a prediction model. When a support engineer presses the submit button, the backend process to display the results would be as follows: the PMR-number would be fetched from the input box and passed as an input to retain api for extracting text content of the PMR. For every request, we store the content into a text file, do natural language processing and pass it to already trained objects of *SVM*, *naive Bayes* and *LSTM*. Furthermore, the result would be fetched for a key word algorithm which was developed using the list of keywords provided by IBM support engineers.

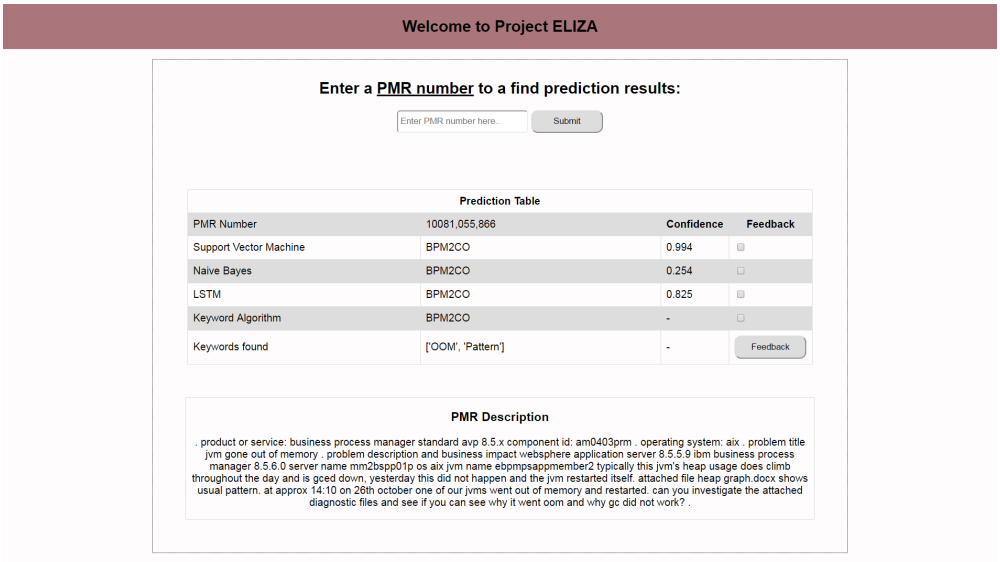


Figure 3.7.: Feedback engine web framework. Results of all the algorithms along with their confidence scores are displayed in tabular format. The problem summary of the PMR is also displayed. Support engineer can read the problem summary and give their feedback about the model.

The results of all four algorithms *SVM*, *naive Bayes*, *LSTM* and *keywords algorithm* will be displayed on the page in tabular format called "prediction table" as shown in Figure 3.7. In the prediction table, the first column displays the names of the algorithms. The second column shows predicted results in form of queue names. The third column is for confidence scores of the algorithms. Here, no confidence score occurs for keyword algorithm as it solely counts the occurrence of words into content. Keywords responsible for the result of the keyword algorithm are also shown in the table. The plan is to use the feedback for the active learning in the future.

In the Figure 3.7, the description text for the inserted PMR-number is displayed below the prediction table. After observing the description and the results, support engineers provide their feedback depending on their knowledge. The

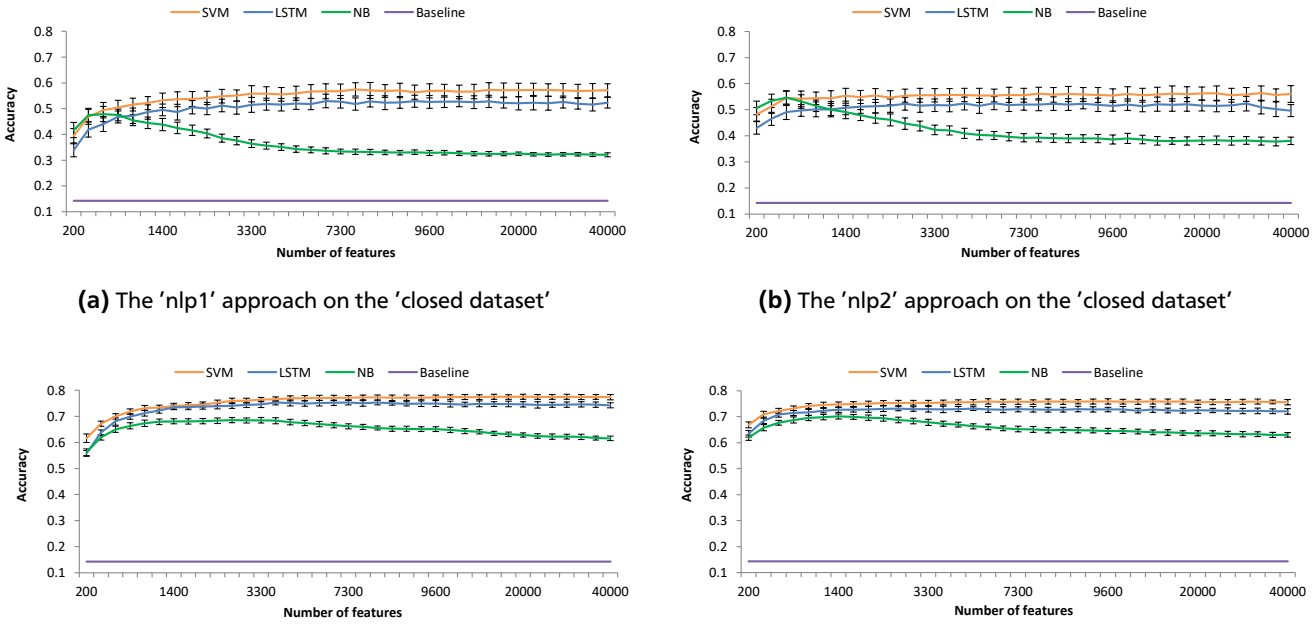
available checkbox in a last column of the table needs to be selected for the right prediction and feedback can be submitted using feedback button. If none of the prediction is right the button can be pressed without selecting anything. Feedbacks given by support engineers are stored into an excel data file which can be used in future to extend our model with active learning. In addition, those feedback are beneficial to examine the keyword algorithm assumption.

In this section, the architecture of our application was explained whereas, in the next section, this application would be evaluated.

4 Experiments

In this chapter, we present results of various prediction models on datasets of the business process management (BPM) product provided by a IBM Deutschland GmbH. We prepared the 'closed dataset' and the 'extended dataset' as discussed in a Section 3.2.2. The 'closed dataset' had 1215 resolved PMRs which were randomly divided into two parts, training dataset (79%) with 959 PMRs and testing dataset (21%) with 256 PMRs. The 'extended dataset' was prepared using keywords provided by BPM product support engineers. The dataset was called as 'extended dataset' because along with the resolved PMRs, it contained additional PMRs which were fetched using keyword algorithm. Several years of experience from the support engineers suggested that some keywords occur more often in PMRs for every queue. Although the assumption is not always right, it still might be useful to overcome the problem of smaller dataset. We implemented feedback engine that gathers feedback from support engineers to verify that assumption. This verification is shown in Figure 4.2. The 'extended dataset' had 7346 PMRs in total, which were divided randomly into training dataset (79%) with 5804 PMRs and testing dataset (21%) with remaining 1542 PMRs. The 'nlp1' and 'nlp2' approaches were used for natural language processing as explained in a Section 3.2.3. In the 'nlp1' approach, *stopword removal* and *bigram* methods were used. Whereas the *pos tagger*, *lemmatization*, *stopword removal* were used in the 'nlp2' appraoch. All the nlp methods were explained Section 2.2. The *term frequency-inverse document frequency (TF-IDF)*, explained in Section 2.3 was used for feature selection. The *linear SVM*, *multinomial naive Bayes* and *long short term memory (LSTM)*, explained in Section 2.4 were used for prediction models. The evaluation metrics such as accuracy, precision and recall were explained in 2.4.4. The accuracy defined as the correct class assignment to total number of samples would achieve an accuracy of $\frac{1}{7} = 0.142$ which is used as base accuracy. For every combination of the nlp approaches and prediction models, the accuracy values for different number of features were evaluated. Furthermore, the comparisons of accuracy, precision and recall values were done with obtained best number of features. For all the experiments, the mean values of 100-fold cross validation were used.

4.1 Specific Feature Selection



(a) The 'nlp1' approach on the 'closed dataset'

(b) The 'nlp2' approach on the 'closed dataset'

(c) The 'nlp1' approach on the 'extended dataset'

(d) The 'nlp2' approach on the 'extended dataset'

Figure 4.1.: For the 'nlp1' and 'nlp2' approach, effects of different number of features on the accuracy values for *linear SVM*, *multinomial naive Bayes* and *long short term memory*. The experiments were performed on the 'closed dataset' and the 'extended dataset'.

The accuracy values with different number of features for *linear SVM*, *multinomial naive Bayes* and *LSTM* are evaluated for the 'nlp1' and 'nlp2' approaches. The accuracy values for the 'closed dataset' and 'extended dataset' are shown in

Figure 4.1. The results in the graph are also shown in Table A.1 for the 'closed dataset' and in Table A.2 for the 'extended dataset' in the appendix. For the 'closed dataset', Figure 4.1a is for the 'nlp1' approach whereas Figure 4.1b is for the 'nlp2' approach. For the 'extended dataset', Figure 4.1c is for the 'nlp1' approach whereas Figure 4.1d is for the 'nlp2' approach.

According to Figure 4.1, we observe that *linear SVM* outperforms the *multinomial naive Bayes* and the *LSTM* in general. As shown in Figure 4.1a for the 'nlp1' approach, the maxima accuracy values for *multinomial naive Bayes*, *linear SVM* and *LSTM* are found to be 0.479, 0.574 and 0.529 with 600, 7900 and 6700 number of features on the 'closed dataset' respectively. Furthermore, for the 'nlp2' approach in Figure 4.1b, the maxima accuracy values for *multinomial naive Bayes*, *linear SVM* and *LSTM* are found to be 0.546, 0.561 and 0.523 with 600, 7900 and 2400 feature numbers on the 'closed dataset' respectively.

As shown in Figure 4.1c for the 'nlp1' approach on the 'extended dataset', the maxima accuracy values for *multinomial naive Bayes*, *linear SVM* and *LSTM* are found to be 0.686, 0.776 and 0.753 with 2400, 15000 and 8400 features respectively. Furthermore, for the 'nlp2' approach on the 'extended dataset' in Figure 4.1d, the maxima accuracy values for *multinomial naive Bayes*, *linear SVM* and *LSTM* are found to be 0.700, 0.759 and 0.731 with 1400, 9600 and 2400 features respectively.

Comparing the classification accuracies from selected features, we found that feature selection helps in improving prediction performance. The 'extended dataset' required more number of features than the 'closed dataset' to achieve higher accuracy because the 'extended dataset' was larger than the 'closed dataset'. Despite the higher performance, the behaviours of the prediction models were similar to the 'closed dataset'. For example, accuracy values for the *Multinomial naive Bayes* showed decrement with more number of features after achieving a higher accuracy values for both the 'nlp1' and 'nlp2' approaches. For *linear SVM*, initial increment in the accuracies until the highest values and then the accuracy values remained relatively similar showing tailing with increasing number of features. In the case of *LSTM* also the accuracy values remained similar after intial increment.

The 'extended dataset' relied on the assumption of keywords provided by the BPM support engineers and feedback engine was implemented for the verification. The feedback values obtained for 132 PMRs using the feedback engine are shown in Figure 4.2.

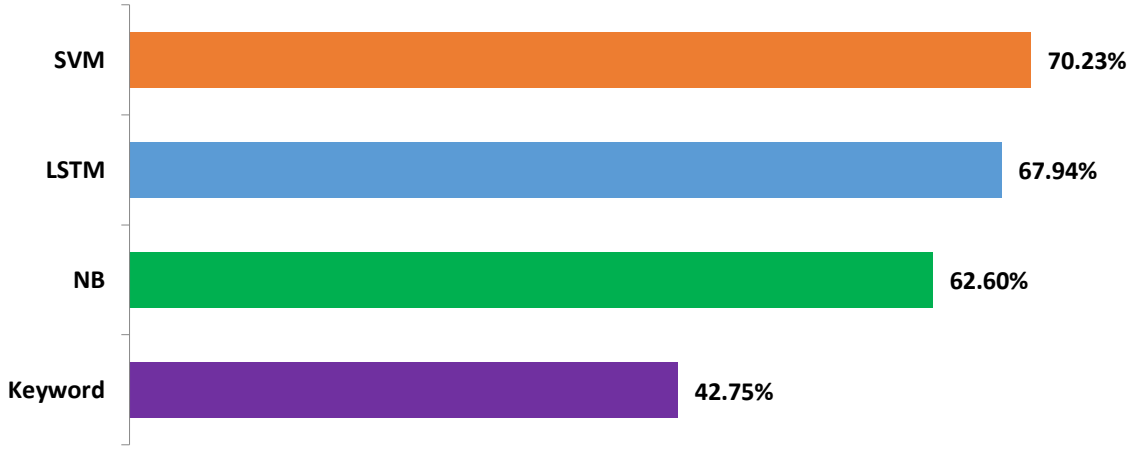
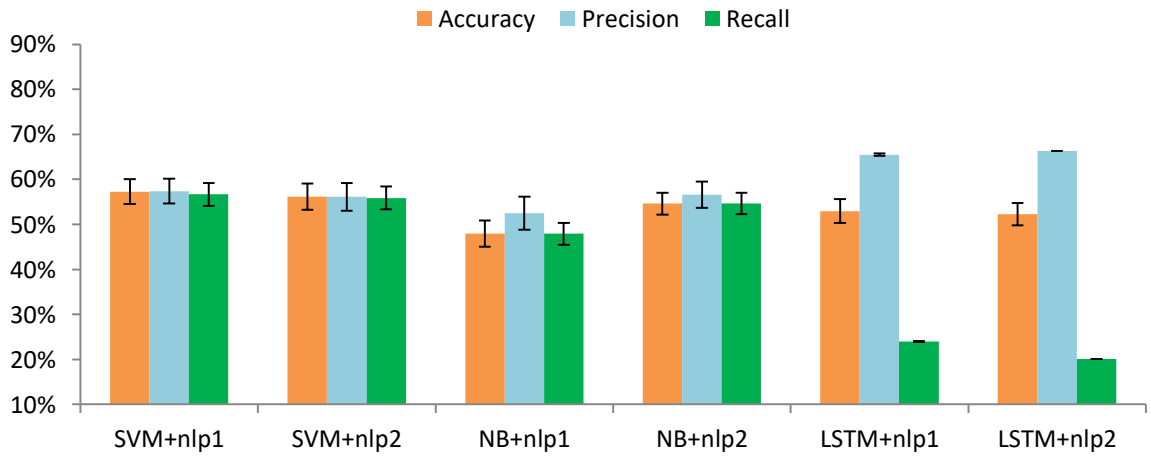


Figure 4.2.: The feedback values for the *linear SVM*, *multinomial naive Bayes*, and *long short term memory (LSTM)* and 'keyword algorithm' obtained from the experienced BPM support engineers.

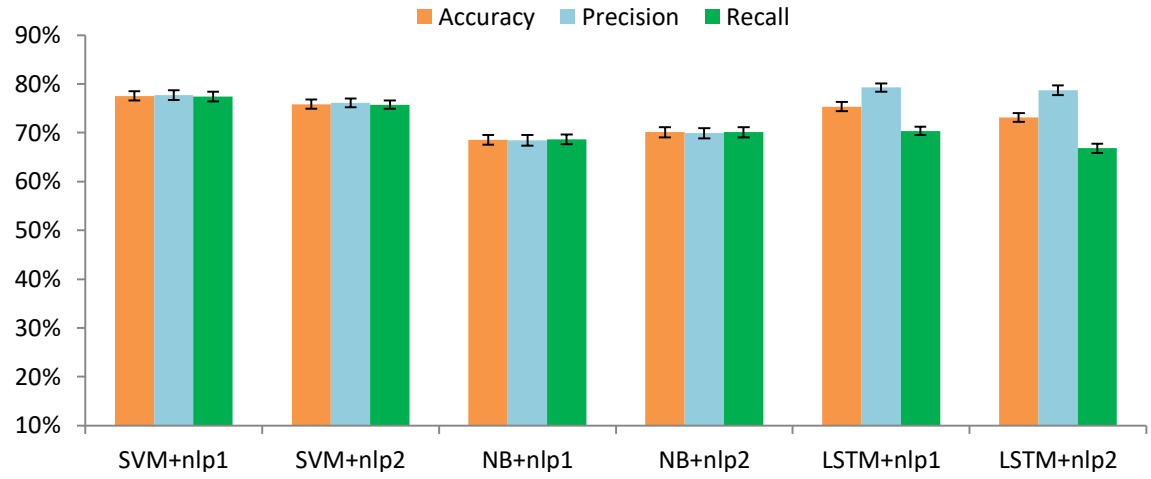
According to figure 4.2, the feedback results were positive enough to support the assumption of the 'extended dataset'. The most positive feedback values (70.23%) were achieved for the *linear SVM*. Whereas, the feedback values obtained for the *LSTM*, *multinomial naive Bayes* and 'keyword algorithm' were 67.94%, 62.60% and 42.75% feedback values respectively.

4.2 Comparison of the Prediction Models

We also calculated the precision and recall values for every combination of 'nlp' approaches and prediction algorithms using number of features at which the highest accuracy was obtained as in Figure 4.1.



(a) the 'closed' dataset.



(b) the 'extended' dataset.

Figure 4.3.: The accuracy, precision and recall values are shown for the combination of *linear SVM*, *multinomial naive Bayes*, and *long short term memory (LSTM)* with the 'nlp1' and 'nlp2' approaches. The results for the 'closed dataset' are shown in Figure 4.3a. Whereas the results for the 'extended dataset' are shown in Figure 4.3b

	SVM+nlp1	SVM+nlp2	NB+nlp1	NB+nlp2	LSTM+nlp1	LSTM+nlp2	Dataset
Accuracy	0.572	0.562	0.479	0.546	0.529	0.522	closed
Precision	0.573	0.561	0.524	0.565	0.654	0.663	
Recall	0.566	0.558	0.479	0.546	0.239	0.200	
Accuracy	0.776	0.759	0.686	0.700	0.753	0.731	extended
Precision	0.777	0.761	0.684	0.699	0.792	0.786	
Recall	0.744	0.757	0.686	0.701	0.703	0.668	

Table 4.1.: Accuracy, precision and recall values of every combination of 'nlp' approaches with *Multinomial naive Bayes*, *linear SVM* and *LSTM* on the 'closed dataset'.

The highest accuracy values and its corresponding precision and recall values for the 'closed dataset' are shown in Figure 4.3a whereas the values for the 'extended dataset' are shown in Figure 4.3b. For both the datasets, values are also shown in Table 4.1.

For the *linear SVM*, the precision values were similar to accuracy values, whereas recall values were marginally less than the accuracy values. For the *multinomial naive Bayes*, precision values were relatively higher whereas recall values were relatively similar to accuracy values. For *LSTM*, The precision values were very high. However, the recall values were very low. According to Figure 4.3b, *linear SVM* with the 'nlp1' approach achieved highest accuracy and recall among all other combination of 'nlp' and prediction algorithms on the 'extended dataset'. The precision values were highest for *linear SVM* on the 'extended dataset'.

Additional results are shown in the appendix. We used randomly chosen 79% PMRs for the training dataset and remaining 21% PMRs as the testing dataset for all the experiments. However, we also calculated the performances using different splitting ratios which are shown in Figure A.1 for the 'closed dataset' and 'extended dataset' in the appendix. Different number of layers were used in the *LSTM* for all combinations of prediction models on both the datasets. The accuracy values with different number of layers for the *LSTM* are shown in Figure A.2 in appendix.

5 Conclusion

The primary objective of this thesis was to find the best prediction model of an automatic bug triage system for IBM Deutschland GmbH. The historical bug reports of business process management (BPM) product were used to make smarter routing prediction for new bug reports. We prepared two datasets called as the 'closed dataset' and 'extended dataset' using PMRs of BPM product. Different natural language processing methods such as *lemmatization*, *pos tagger*, *N-gram* and *stopword removal* were implied to improve the text vocabulary. We processed the extracted vocabulary using *term frequency-inverse document frequency (TF-IDF)* method to generate feature vectors and to select the appropriate number of features. Finally, we used three classification algorithms, the *linear support vector machines (SVM)*, the *multinomial naive Bayes*, and a *long short term memory (LSTM)* network for the classification of bug reports. We have concluded that the best prediction model is the *linear support vector machine* with the 'nlp1' approach. The model achieved accuracy of 57.4%, a precision of 57.3% and recall value if 56.6% for the 'closed dataset'. For the 'extended dataset', the values were 77.6%, 77.7% and 74.4% for accuracy, precision and recall respectively. It was the optimal routing style for BPM product of IBM. We also implemented a web framework to collect feedback of the prediction models from IBM support engineers. It verified the assumption of keywords in the 'extended dataset' as discussed in Section 4.1. The bug triage tool was implemented in microservice architecture using docker containers.

5.1 Discussion

An already prepared dataset which could be used directly to build prediction models was not available for the business process management (BPM) product of IBM Deutschland GmbH. Therefore, we prepared the 'closed dataset' and 'extended dataset' using PMRs from BPM product and categorized PMRs into seven categories defining seven teams working in BPM support department of IBM. However, BPM support department in USA is using five resolver teams so categorizing PMRS into five categories. Therefore, even using only resolved PMRs in the 'closed dataset', the redundancy problem could not be ignored. Moreover, the 'extended dataset' was prepared using the keywords provided by BPM support engineers. It increased performance compared to the 'closed dataset' because of more number of PMRs in the dataset. However, it heavily relies on the assumption of keywords. Although, we developed the 'feedback engine' to collect the feedback results about the prediction models built on the 'extended dataset'. The collected feedback will prove the answer if this assumption was valid. The results of collected feedback are shown in in Figure 4.2. Furthermore, we can prepare the dataset using this feedback which will be more accurate than the 'closed dataset' and 'extended dataset'.

Higher number of PMRs in the 'extended dataset' helped to achieve better performance than the 'closed dataset'. However, the behaviours of the prediction models were similar for both the 'closed dataset' and 'extended dataset' i.e. the 'nlp1' approach achieved better performance than the 'nlp2' approach for *linear SVM* and *LSTM*. However, the 'nlp2' approach showed higher performance than the 'nlp1' approach for *multinomial naive Bayes*. In general, for the 'closed dataset' and 'extended dataset', the 'nlp1' approach required similar or more number of features than the 'nlp2' approach.

5.1.1 Benefits of Containers

We discussed the benefits of microservice architecture in Section 2.5. Here, the applicability of those benefits in our application is discussed.

Resource Optimization. Although, 'gather data' container runs continuously and collects new data, the new prediction model is not needed to be relearned everyday. It is usually done when there are enough new PMRs in the dataset (often in 2-3 months). Therefore, only the 'gather data' container requires daily computational power. The other containers, 'data preprocessing', 'natural language processing' and 'prediction model' require computation power once in 2-3 months.

Technology Independence. All the containers are independent, this has an advantage that any technology or programming language could be used inside the container. In our application, java was used for the 'gather data container' whereas python was used for the 'data preprocessing', the 'natural language processing' and the 'prediction model' containers. HTML, CSS and javascript technologies were used along with python for the 'feedback engine' container.

Replaceable. The 'feedback engine' container is a temporary container for the purpose of gathering a real feedback from the support engineers. If it is not required in the future, it could be shut down without affecting other containers.

Extensibility. In future, the idea is to integrate the application prototype with real IBM support system. One or more containers could be added to glue it with IBM support portal without applying any changes to other containers.

In this section, we discussed the experiments and their results. We conclude our results in the next section.

5.2 Future Work

This thesis is a first step for developing an automated routing system for IBM Deutschland GmbH. There are many opportunities for improvement. In this section, some suggestions and ideas for future work are presented.

Improved Dataset. In our thesis, accurate datasets of the BPM product were not available. We prepared the 'closed dataset' and the 'extended dataset' using PMRs of BPM product. However, there was a redundancy problem as discussed in Section 5.1. Furthermore, the 'extended dataset' relied on the assumption of collected keywords provided by BPM support engineers. However, we created a feedback engine where support engineers can give their feedback about the result of the prediction models. We can use the feedback in the future to prepare more accurate datasets.

Additional Feature Selection. In this thesis, the *term frequency-inverse document frequency method (TF-IDF)* was used for feature selection. As *TF-IDF* representation of features is based on the bag-of-words (BoW) model, which ignores the the position of each term in the context and does not capture the semantics of terms. In future, we can try to build a statistical language modelling architecture. Furthermore, feature reduction techniques such as *ChiSq selection* [52] can be utilized.

Ensemble Learning. In this thesis, we used three models separately. In future work, ensemble learning can be evaluated to combine several learning algorithms. It might be able to obtain a better performance than it could be obtained solely from any of the constituent learning algorithms.

Experiments on More Products. The experiments in this thesis were, for practical reasons, performed on a single product, the business process management (BPM) from IBM Deutschland GmbH. We prepared the 'closed dataset' and 'extended dataset' from the PMRs of BPM product and compared the results of different prediction models. Future work should evaluate the prediction models on other IBM products.

Improved PMRs. We used real historical PMRs of BPM product in this thesis. The PMRs were generated by customers after encountering the issues. However, many PMRs did not contain enough information about the issue. For example, the text such as 'The software is not working properly, can you please help me?' is not helpful to decide. Such PMRs affected the performance of the prediction models. In future, a guided system such as 'Question & Answer (QA)' should be provided to customers to gather as much as possible information about the issue. The proper information in PMRs would definitely be helpful to increase the performance.

Incremental Learning. In this thesis, our focus was to compare natural language processing approaches for the prediction models. We tried to compare the best performances using different parameters. In future, an incremental algorithm could be used where inputs are provided, one at a time, to the classifier and the classifier updates itself appropriately. We already provided the foundation for that by using mocroservice architecture. The first container, 'gather data' is already gathering new PMRs continuously. The 'prediction model' container can be enhanced to adapt incoming PMRs and building new prediction models automatically after every time slot.

Semi-supervised or Unsupervised Learning Approach. We chose to use a supervised machine-learning algorithms to categorize the PMRs. This choice means that we must provide a label for each PMR or not use the PMR for training or testing. In future, we could incorporate an unsupervised algorithm, such as *K-means clustering*, which does not require labelling for the PMRs. Alternatively, semi-supervised method like in [20], can be investigated. Here only few PMRs need to be labelled. It could be useful to handle the redundancy problem that we faced in our current approach as the labels are not required for all the PMRs.

Active Learning: Currently, we are storing feedback from the BPM support engineers. In future, an active learning approach can be implemented as a extension to our current model. If label assignments are wrong for some PMRs, it would be recognized from the feedback. So in the next phase correct labels would be used.

Bibliography

[1] G. Sharma, S. Sharma, and S. Gujral, "A novel way of assessing software bug severity using dictionary of critical terms," *Procedia Computer Science*, vol. 70, pp. 632 – 639, 2015.

[2] C. Olah, "Understanding lstm networks," 27 August 2015. Colah's Blog. Github `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[3] S. Newman, *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 1st ed., February 2015.

[4] D. Jaramillo, D. V. Nguyen, and R. Smart, "Leveraging microservices architecture by using docker technology," in *SoutheastCon 2016*, pp. 1–5, March 2016.

[5] B. Beizer, *Software Testing Techniques*. Dreamtech, 2003.

[6] D. Bertram, A. Voida, S. Greenberg, and R. Walker, "Communication, collaboration, and bugs: The social nature of issue tracking in small, collocated teams," in *Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work*, CSCW '10, (New York, NY, USA), pp. 291–300, ACM, 2010.

[7] K. Chaturvedi and V. Singh, "Determining bug severity using machine learning techniques," in *Software Engineering (CONSEG), 2012 CSI Sixth International Conference on*, pp. 1–6, IEEE, 2012.

[8] S. N. Ahsan, J. Ferzund, and F. Wotawa, "Automatic software bug triage system (bts) based on latent semantic indexing and support vector machine," in *Proceedings of the 2009 Fourth International Conference on Software Engineering Advances*, ICSEA '09, (Washington, DC, USA), pp. 216–221, IEEE Computer Society, 2009.

[9] J. Anvik, "Automating bug report assignment," in *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, (New York, NY, USA), pp. 937–940, ACM, 2006.

[10] D. Jaramillo, D. V. Nguyen, and R. Smart, "Leveraging microservices architecture by using docker technology," in *SoutheastCon 2016*, pp. 1–5, March 2016.

[11] L. Florio and E. D. Nitto, "Gru: An approach to introduce decentralized autonomic behavior in microservices architectures," in *2016 IEEE International Conference on Autonomic Computing (ICAC)*, pp. 357–362, July 2016.

[12] J. Turnbull, *The Docker Book: Containerization is the new virtualization*. No. v1.3.1, Oktober 2014.

[13] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, pp. 81–84, Sept 2014.

[14] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[15] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, (New York, NY, USA), pp. 461–470, ACM, 2008.

[16] A. Sureka and P. Jalote, "Detecting duplicate bug report using character n-gram-based features," in *Proceedings of the 2010 Asia Pacific Software Engineering Conference*, APSEC '10, (Washington, DC, USA), pp. 366–374, IEEE Computer Society, 2010.

[17] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "Characterizing and predicting which bugs get fixed: An empirical study of microsoft windows," in *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, (New York, NY, USA), pp. 495–504, ACM, 2010.

[18] G. A. Di Lucca, M. Di Penta, and S. Gradara, "An approach to classify software maintenance requests," in *Software Maintenance, 2002. Proceedings. International Conference on*, pp. 93–102, IEEE, 2002.

[19] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?," in *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, (New York, NY, USA), pp. 361–370, ACM, 2006.

[20] F. Thung, X. B. D. Le, and D. Lo, "Active semi-supervised defect categorization," in *2015 IEEE 23rd International Conference on Program Comprehension*, pp. 60–70, May 2015.

[21] J. woo Park, M.-W. Lee, J. Kim, S. won Hwang, and S. Kim, "Costriage: A cost-aware triage algorithm for bug reporting systems," 2011.

[22] X. Xia, D. Lo, X. Wang, and B. Zhou, "Dual analysis for recommending developers to resolve bugs," *Journal of Software: Evolution and Process*, vol. 27, no. 3, pp. 195–220, 2015.

[23] 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)*, pp. 1–10, May 2010.

[24] T. E. Foundation, "Eclipse dataset." `http://www.eclipse.org`.

[25] I. mozilla.org contributors, "Bugzilla dataset." `https://bugzilla.mozilla.org/`, 1998–2017.

[26] G. Team, "Global Compositae Checklist dataset." `https://gcc.gnu.org/bugs/`.

[27] D. Behl, S. Handa, and A. Arora, "A bug mining tool to identify and analyze security bugs using naive bayes and tf-idf," in *2014 International Conference on Reliability Optimization and Information Technology (ICROIT)*, pp. 294–299, Feb 2014.

[28] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE '09, (New York, NY, USA), pp. 111–120, ACM, 2009.

[29] A. C. P. Ltd., "Mahout dataset." `https://issues.apache.org/jira/browse/mahout/?selectedTab=com.atlassian.jira.jira-projects-plugin:issues-panel`.

[30] A. C. P. Ltd., "Lucene dataset." `https://issues.apache.org/jira/browse/lucene/?selectedTab=com.atlassian.jira.jira-projects-plugin:issues-panel`.

[31] A. C. P. Ltd., "OpenNLP dataset." `https://issues.apache.org/jira/browse/opennlp/?selectedTab=com.atlassian.jira.jira-projects-plugin:issues-panel`.

[32] T. A. S. Foundation, "Apache dataset." `http://httpd.apache.org/bug_report`.

[33] K. Bugzilla, "Linux kernel dataset." `https://bugzilla.kernel.org/`.

[34] T. G. Team, "GNU compiler collection dataset." `https://gcc.gnu.org/bugs/`.

[35] T. O. Team, "OpenOffice dataset." `https://wiki.openoffice.org/wiki/Issue_Tracker`.

[36] T. N. Team, "Netbeans dataset." `https://netbeans.org/community/issues.html`.

[37] C. M. Bishop, "Pattern recognition," *Machine Learning*, vol. 128, 2006.

[38] T. Joachims, "Text categorization with suport vector machines: Learning with many relevant features," in *Proceedings of the 10th European Conference on Machine Learning*, ECML '98, (London, UK, UK), pp. 137–142, Springer-Verlag, 1998.

[39] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation.," in *EMNLP*, vol. 14, pp. 1532–43, 2014.

[40] A. Tripathy, A. Agrawal, and S. K. Rath, "Classification of sentimental reviews using machine learning techniques," *Procedia Computer Science*, vol. 57, pp. 821–829, 2015.

[41] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of massive datasets*. Cambridge University Press, 2014.

[42] C. S. Perone, "Machine Learning :: Text feature extraction (tf-idf) ." `http://blog.christianperone.com/2011/09/machine-learning-text-feature-extraction-tf-idf-part-i/`, 2011. [Online; accessed 07-September-2016].

[43] R. O. Duda and P. E. Hart, *Pattern classification and scene analysis*. John Wiley and Sons, 1973.

[44] P. Langley, W. Iba, and, and K. Thompson, "An analysis of bayesian classifiers," in *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI'92, pp. 223–228, AAAI Press, 1992.

[45] I. Ahmed, D. Guan, and T. C. Chung, "Sms classification based on naive bayes classifier and apriori algorithm frequent itemset," *International Journal of machine Learning and computing*, vol. 4, no. 2, p. 183, 2014.

[46] C. D. Manning, P. Raghavan, and H. Schutze, "Text classification and naive bayes," *Introduction to information retrieval*, vol. 1, p. 6, 2008.

[47] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York, NY, USA: Springer-Verlag New York, Inc., 1995.

[48] D. Boswell, "Introduction to support vector machines," 2002.

[49] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE Transactions on Neural Networks*, vol. 13, pp. 415–425, Mar 2002.

[50] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, pp. 1735–1780, Nov. 1997.

[51] W. Zhu, N. Zeng, N. Wang, *et al.*, "Sensitivity, specificity, accuracy, associated confidence interval and roc analysis with practical sas® implementations," *NESUG proceedings: health care and life sciences, Baltimore, Maryland*, pp. 1–9, 2010.

[52] R. Baeza-Yates, B. Ribeiro-Neto, *et al.*, *Modern information retrieval*, vol. 463. ACM press New York, 1999.

[53] A. Graves, *Supervised sequence labelling with recurrent neural networks*. Studies in Computational intelligence, Heidelberg, New York: Springer, 2012.

[54] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '08/FSE-16, (New York, NY, USA), pp. 308–318, ACM, 2008.

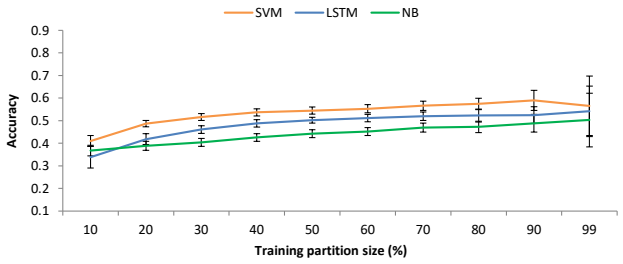
[55] A. Smola and S. Vishwanathan, "Introduction to machine learning," *Cambridge University, UK*, vol. 32, p. 34, 2008.

[56] C. D. Manning, P. Raghavan, H. Schütze, *et al.*, *Introduction to information retrieval*, vol. 1. Cambridge university press Cambridge, 2008.

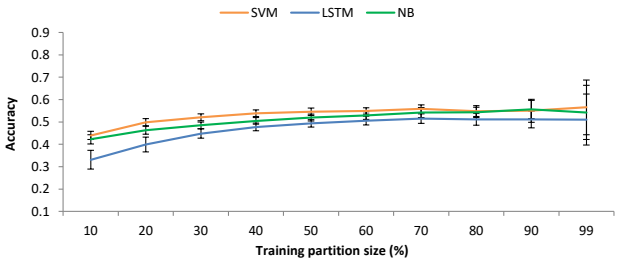
A Appendix

A.1 Accuracy Values for Different Splitting Ratios

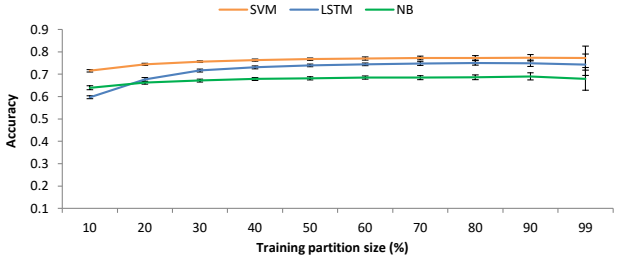
For the 'closed dataset' and 'extended dataset', we used randomly chosen 79% PMRs as training dataset and remaining 21% PMRs as testing dataset for all the experiments. However, we also evaluated accuracy values with different splitting ratios for training and resting dataset as shown in Figure A.1. In the case of 'closed dataset', the results of the 'nlp1' appraoch are shown in Figure A.1a and results of the 'nlp2' approach are shown in Figure A.1b. In the case of 'extended dataset', the results of the 'nlp1' appraoch are shown in Figure A.1c and results of the 'nlp2' approach are shown in Figure A.1c.



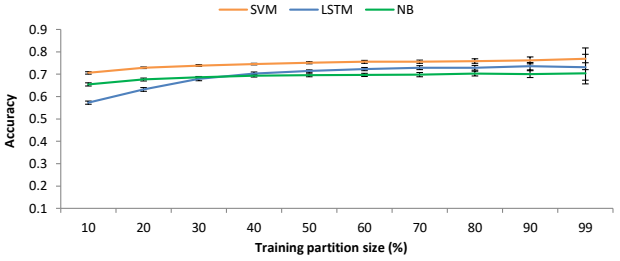
(a) The 'nlp1' approach on the 'closed dataset'



(b) The 'nlp2' approach on the 'closed dataset'



(c) The 'nlp1' approach on the 'extended dataset'



(d) The 'nlp2' approach on the 'extended dataset'

Figure A.1.: For the 'nlp1' and 'nlp2' approaches, the accuracy values with different splitting ratios for *linear SVM*, *multinomial naive Bayes* and *long short term memory*. The experiment was performed on the 'closed dataset' and the 'extended dataset'.

The accuracy values were seemed to be increasing with more number of training samples in general. The lowest accuracy values were achieved generally by using 10% of PMRs as training dataset for *linear SVM*, *multinomial naive Bayes*, and *long short term memory (LSTM)*.

A.2 Accuracy Values for Different Number of Features

As discussed in Section 4.1, We also evaluated accuracy values for different number of features by applying *term frequency-inverse document frequency (tf-idf)* method on the 'closed dataset' and 'extended dataset'. The values used in Figure 4.1 are show here in tabular format. The values for the 'closed dataset' are shown in Table A.1 and values for the 'extended dataset' are shown in Table A.2. The highest achieved values are shown as bold numbers.

	NLP1			NLP2		
No.of Features	NB	SVM	LSTM	NBC	SVM	LSTM
200	0.416	0.394	0.338	0.506	0.481	0.430
600	**0.479**	0.494	0.440	**0.546**	0.544	0.490
1000	0.455	0.516	0.472	0.514	0.543	0.501
1200	0.444	0.521	0.486	0.500	0.544	0.501
1400	0.438	0.532	0.495	0.490	0.553	0.507
1600	0.424	0.536	0.486	0.478	0.548	0.511
2000	0.404	0.541	0.500	0.461	0.545	0.515
2400	0.385	0.548	0.512	0.446	0.553	**0.522**
2800	0.375	0.551	0.504	0.437	0.555	0.516
3300	0.363	0.558	0.514	0.422	0.555	0.517
3900	0.356	0.558	0.517	0.420	0.556	0.516
4500	0.351	0.555	0.515	0.409	0.556	0.523
5300	0.342	0.559	0.519	0.402	0.554	0.514
6000	0.340	0.566	0.517	0.401	0.554	0.524
6700	0.336	0.567	**0.529**	0.396	0.556	0.518
7300	0.334	0.567	0.526	0.392	0.555	0.519
7900	0.332	**0.574**	0.518	0.393	**0.561**	0.519
8400	0.331	0.570	0.528	0.390	0.557	0.523
8700	0.330	0.568	0.523	0.389	0.560	0.519
9000	0.329	0.570	0.524	0.389	0.558	0.522
9300	0.331	0.563	0.528	0.389	0.556	0.518
9600	0.328	0.569	0.526	0.386	0.554	0.514
10000	0.329	0.569	0.527	0.389	0.560	0.519
11000	0.326	0.565	0.526	0.385	0.555	0.514
13000	0.325	0.566	0.525	0.380	0.558	0.520
15000	0.324	0.572	0.528	0.379	0.561	0.518
18000	0.324	0.572	0.522	0.380	0.559	0.520
22000	0.323	0.573	0.523	0.382	0.562	0.514
27000	0.323	0.570	0.525	0.381	0.558	0.523
30000	0.323	0.568	0.518	0.379	0.563	0.508
35000	0.321	0.569	0.515	0.377	0.556	0.501
40000	0.320	0.571	0.521	0.380	0.560	0.496

Table A.1.: Accuracy values of *linear SVM*, *multinomial naive Bayes* and *long short term memory* for every selected number of features on the 'closed dataset'. The left side values are for the 'nlp1' approach and right side values are for the 'nlp2' approach.

	NLP1			NLP2		
No.of Features	NB	SVM	LSTM	NBC	SVM	LSTM
200	0.562	0.616	0.558	0.619	0.668	0.634
600	0.650	0.700	0.681	0.676	0.721	0.708
1000	0.673	0.732	0.712	0.694	0.741	0.716
1200	0.679	0.733	0.723	0.698	0.745	0.723
1400	0.681	0.739	0.734	**0.700**	0.747	0.726
1600	0.680	0.743	0.736	0.699	0.748	0.727
2000	0.683	0.751	0.739	0.694	0.752	0.730
2400	**0.685**	0.759	0.741	0.687	0.752	**0.731**
2800	0.684	0.760	0.743	0.683	0.753	0.729
3300	0.685	0.764	0.746	0.678	0.752	0.729
3900	0.683	0.765	0.753	0.673	0.754	0.728
4500	0.678	0.769	0.751	0.699	0.755	0.729
5300	0.674	0.769	0.749	0.663	0.755	0.730
6000	0.670	0.771	0.750	0.660	0.758	0.728
6700	0.665	0.770	0.752	0.655	0.757	0.727
7300	0.662	0.772	0.752	0.652	0.757	0.729
7900	0.659	0.772	0.749	0.651	0.756	0.728
8400	0.655	0.772	**0.753**	0.647	0.758	0.726
8700	0.653	0.771	0.750	0.649	0.757	0.727
9000	0.652	0.771	0.748	0.648	0.758	0.729
9300	0.650	0.772	0.748	0.647	0.758	0.728
9600	0.651	0.772	0.749	0.644	**0.758**	0.727
10000	0.647	0.774	0.747	0.644	0.757	0.728
11000	0.643	0.774	0.745	0.643	0.757	0.723
13000	0.641	0.773	0.748	0.639	0.758	0.727
15000	0.635	**0.775**	0.747	0.639	0.758	0.734
18000	0.631	0.744	0.747	0.637	0.758	0.722
22000	0.625	0.774	0.744	0.636	0.757	0.724
27000	0.622	0.775	0.745	0.632	0.757	0.722
30000	0.621	0.774	0.745	0.632	0.758	0.722
35000	0.616	0.774	0.744	0.630	0.757	0.721
40000	0.615	0.773	0.742	0.630	0.755	0.720

Table A.2.: Accuracy values of *linear SVM*, *multinomial naive Bayes* and *long short term memory* for every selected number of features on the 'extended dataset'. The left side values are for the 'nlp1' approach and right side values are for the 'nlp2' approach.

A.3 Accuracy Values for Different Number of Layers in *Long Short Term Memory (LSTM) Network*

The accuracy values obtained with different number of layers for *LSTM* layers are shown in Figure A.2. The values were for the 'nlp1' and 'nlp2' approaches on the 'closed dataset' and 'extended dataset'.

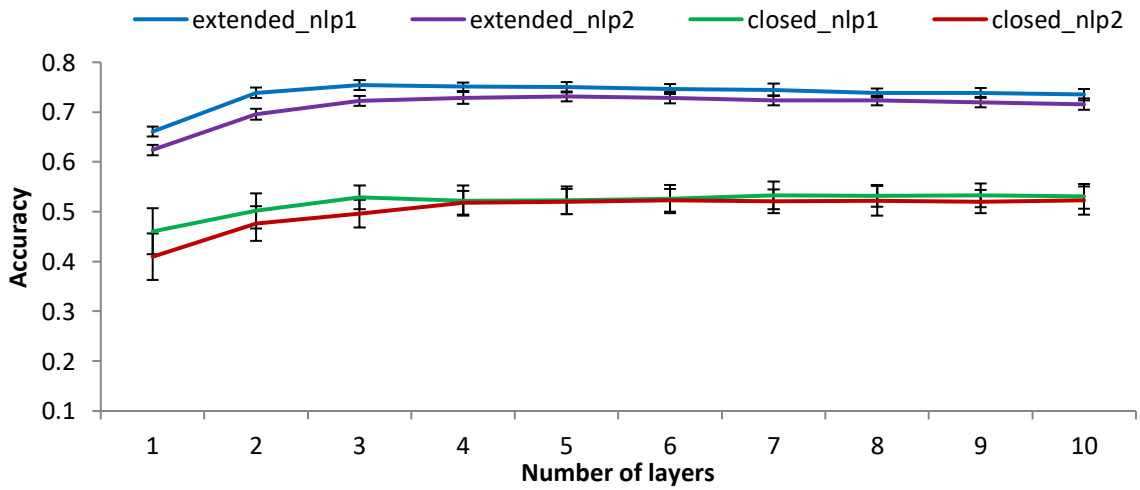


Figure A.2.: The accuracy values for with different number of layers for *long short term memory (LSTM)*. The results are for every combinations of the 'nlp' approaches with the 'closed dataset' and 'extended dataset'.

As shown in Figure A.2, more number of layers were required for the 'extended dataset' compare to the 'closed dataset'. After initial increment in accuracy values with number of layers, values remain similar with further increment.