Heriot-Watt University

Masters Thesis

Sentiment Analysis in Software Engineering

Author:
Aneesah Abdul Kadhar

Supervisor:
Dr. Smitha Kumar

A thesis submitted in fulfilment of the requirements for the degree of MSc. Software Engineering

in the

School of Mathematical and Computer Sciences

August 2021

Declaration of Authorship

I, Aneesah Abdul Kadhar, declare that this thesis titled, 'Sentiment Analysis in Software Engineering' and the work presented in it is my own. I confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: Aneesah

Date: 15/08/2021

Abstract

Sentiment analysis (SA) is the study of text to determine the sentiment or emotional tone (positive, negative, neutral) expressed in the words [76]. SA is increasingly being applied on software engineering text to study developer's sentiments, improve productivity and identify problematic features in APIs by analysing the negative sentiments expressed. Prior studies in SA for Software Engineering (SE) domain employed SA tools developed for open domain yielding poor results. Additionally, the state of art SA tools customized for SE domain yield varying results depending on the techniques and datasets employed. In this study, we aim to improve the performance of SA for SE domain by applying deep-learning techniques trained on StackOverflow [13] and Jira datasets[58] to build the sentiment classifiers. We developed and compared the performance of stacked CNN, stacked LSTM and stacked BiLSTM classifiers and found that the stacked BiLSTM classifier outperformed the other classifiers developed on the above-mentioned StackOverflow and Jira datasets. Additionally, this study investigated the impact of domain-customization on the performance of the classifiers by employing SE-specific word-embeddings learnt from 2 million StackOverflow posts and compared with the pre-trained open-domain word embeddings learnt from google news posts. The classifiers employing Google News (GN) embeddings outperformed the SE-customized based classifiers. The better performance of GN embedding classifiers is due to the large general training corpus of the GN word embedding model which can identify the sentiments expressed accurately when compared to SE-specific word embedding model. Furthermore, we observed that the stacked BiLSTM-GN classifier reduces the misclassifications of neutral sentiments samples as negative which has long been a challenge for SA in SE domain.

Acknowledgements

My sincere and honest thanks to Prof. Smitha Kumar, for suggesting this interesting project and for her invaluable guidance and support. This project would have not been complete without her generous help.

I would also like to thank my parents and grandparents for their guidance and unending support and my lovely sisters Nashwa and Nahla for always motivating me.

Contents

List of Figures

List of Tables

Abbreviations

SA	**S**entiment **A**nalysis
ML	**M**achine **L**earning
DL	**D**eep **L**earning
SE	**S**oftware **E**ngineering
NLP	**N**atural **L**anguage **P**rocessing
CNN	**C**onvolution **N**eural **N**etwork
RNN	**R**ecurrent **N**eural **N**etwork
LSTM	**L**ong **S**hort **T**erm **M**emory
BiLSTM	**B**idirectional **L**ong **S**hort **T**erm **M**emory
GN	**G**oogle **N**ews
SO	**S**tack **O**veflow

Chapter 1

Introduction

1.1 Background

Sentiment analysis (SA) to analyze developers' emotions is making a steady emergence in recent years. Study conducted by Guzman et al. in [29] indicates that the emotions expressed by the developers play an important role and have a large impact on the overall productivity, task quality and task synchronization at the early stage of software development. Identifying the negative sentiments early on at the project development leads to adoption of corrective measures and improve the understanding of the socio-factors of developers' sentiments [25, 29]. Emotions of developers are often expressed in crowdsourcing platforms, mailing lists, forums, commit comments of various open-source platforms, and issue tracking tools that handle their collaborative work [25, 29, 54]. The sentiments expressed can then be used to improve various software libraries, Software Engineering tools, code suggestion tools [29, 44]. Sentiment analysis of crowdsourcing websites like Stack Overflow has been used to give insightful suggestions on the improvement of source code and documentation [62]. Software Engineering (SE) text vary from spoken language and consists jargon, hence misclassifications of sentiment polarities is highly common. For instance, the study conducted by Novielli et al. [13] found that neutral sentence such as " what is the best way to kill a critical process" is erroneously classified as negative polarity in off-shelf lexicon tools.

1.2 Motivation

Given the wide variety of uses of Sentiment analysis in the SE domain, prior studies involved researchers using off-the-shelf SA tools for SE domain datasets achieving poor results [37, 38]. This prompted the need for a customized SE-specific sentiment analysis tool, as SE text differs from articles, books, spoken language and consists of technical jargon, code snippets, and URLs [4, 37, 44].

However recent techniques for SE domain sentiment analysis employ SE- specific lexicon methods [37, 33]and machine learning algorithms [4, 13] have reported an improvement, but are unreliable and provide varying results when applied on datasets other than the original datasets they have been trained on [66]. More recently, studies conducted by researchers have shown that deep-learning algorithms when used in Sentiment analysis outperform the machine-learning-based approaches when both are trained in open-domain datasets [16]. This provides the motivation to apply deep-learning algorithms to identify sentiment polarity in SE text.

1.3 Research Aims and Objectives

This study aims to classify the sentiments of manually annotated dataset to positive, negative, and neutral sentiment polarity. Deep-learning algorithms are used to identify sentiment polarity and reduce misclassifications. Additionally, the study aims to address the following two research questions.

RQ1: What are the best deep-learning algorithms to achieve better classification results on a software engineering dataset?

RQ2: What is the impact on the performance of the classifier when generic and software-specific word embedding are used?

The primary objectives of this project are:

1. Preprocess the dataset by identifying the most appropriate technique in literature.
2. Develop a deep-learning based sentiment analyser (SA) by evaluating various deep learning algorithms.

3. Evaluate the performance of the classifier with SE specific and generic word embedding.

1.4 Report Structure

The rest of report is arranged as follows: chapter 2 details important concepts of SA and deep-learning, along with the related work. Chapter 3 analyses the requirements needed to implement the models. Chapter 4 contains the methodology that is used in this project to build the models. Chapter 5 presents the implementation, source code and results of all experiments for the dataset. Chapter 6 discusses the results obtained. Chapter 7 presents the conclusion, limitations, and future work. Appendix A1 contains the Professional and Legal Ethics and Project Plan that was followed.

Chapter 2

Literature Review

In this chapter, some of the core concepts of deep learning is introduced in order to grasp the objectives of this dissertation. The chapter also discusses the various sentiment analysis approaches, text-processing methods, word-embedding techniques. Finally, previous work in Sentiment analysis for Software engineering domain is critically reviewed.

2.1 Sentiment Analysis

Sentiment Analysis also known as Opinion Mining, is a Natural Language Processing technique that classifies the sentiment polarity expressed in text as positive, negative and neutral. The study conducted by Garcia et al. in [25] found a strong correlation between emotions expressed by developers on issue tracker tools and their active contribution to open-source projects. It was found that developers expressing strong emotions, positive or negative are highly likely to become inactive to the open-source projects they contribute to.

Previous study conducted by Guzman et al. in [29] stresses the importance of analyzing developers' emotions to guarantee the overall success of the project. Moreover, developers use code repositories, issue tracking tools, mailing lists and forums to collaborate their work, and often sentiments are expressed in them [53, 54]. Analyzing developers' emotions help in identifying the bottle necks in the development of open-source projects and can help in taking corrective measures to address them in time [25, 29].

Sentiment Analysis has a wide range of applications in the SE domain, for example, extracting negative comments in an API review can help in identifying problematic API features [78], recommending software libraries and packages to developers [44]. Sentiment analysis classifies document, sentence and aspects in terms of polarity i.e. negative, positive, neutral and also performs subjectivity classification, determining the given input is subjective or objective [15].

2.2 Levels of Sentiment Analysis classification

Sentiment analysis is performed mainly on three levels of extraction, namely document-level, sentence-level and aspect-level. This project performs sentiment classification at the sentence-level.

2.2.1 Document-level Sentiment classification

In document-level sentiment classification, the sentiment polarity of a large text such as paragraph or a document is identified [45, 63].

2.2.2 Sentence-level Sentiment classification

At the sentence-level sentiment classification, the sentiment polarity of each sentence is determined. Initially, the sentence is identified as subjective or objective [63]. Then the identified subjective sentences are classified into positive, neutral and negative sentiment polarities [4, 21].

2.2.3 Aspect-level sentiment classification

In Aspect-level sentiment classification, the polarities of various aspects present in the sentence are analysed. For example, in the sentence “The sound quality of the speakers is amazing, but the battery -life is too short.”, conveys that there is a positive sentiment for the sound quality but negative sentiment regarding the battery-life. The Aspect-level

sentiment classification gives a deeper insight about the sentiments of the different aspects present [45]. The aspect-level sentiment analysis classification is more challenging than document and sentence level sentiment classification.

2.3 Sentiment Analysis Approaches

Currently there are three approaches to perform sentiment analysis of text.

1. Lexicon or Rule-based approach
2. Machine learning-based approach
3. Hybrid approaches

The figure 2.1 is an illustration of the various approaches to perform sentiment analysis of text.

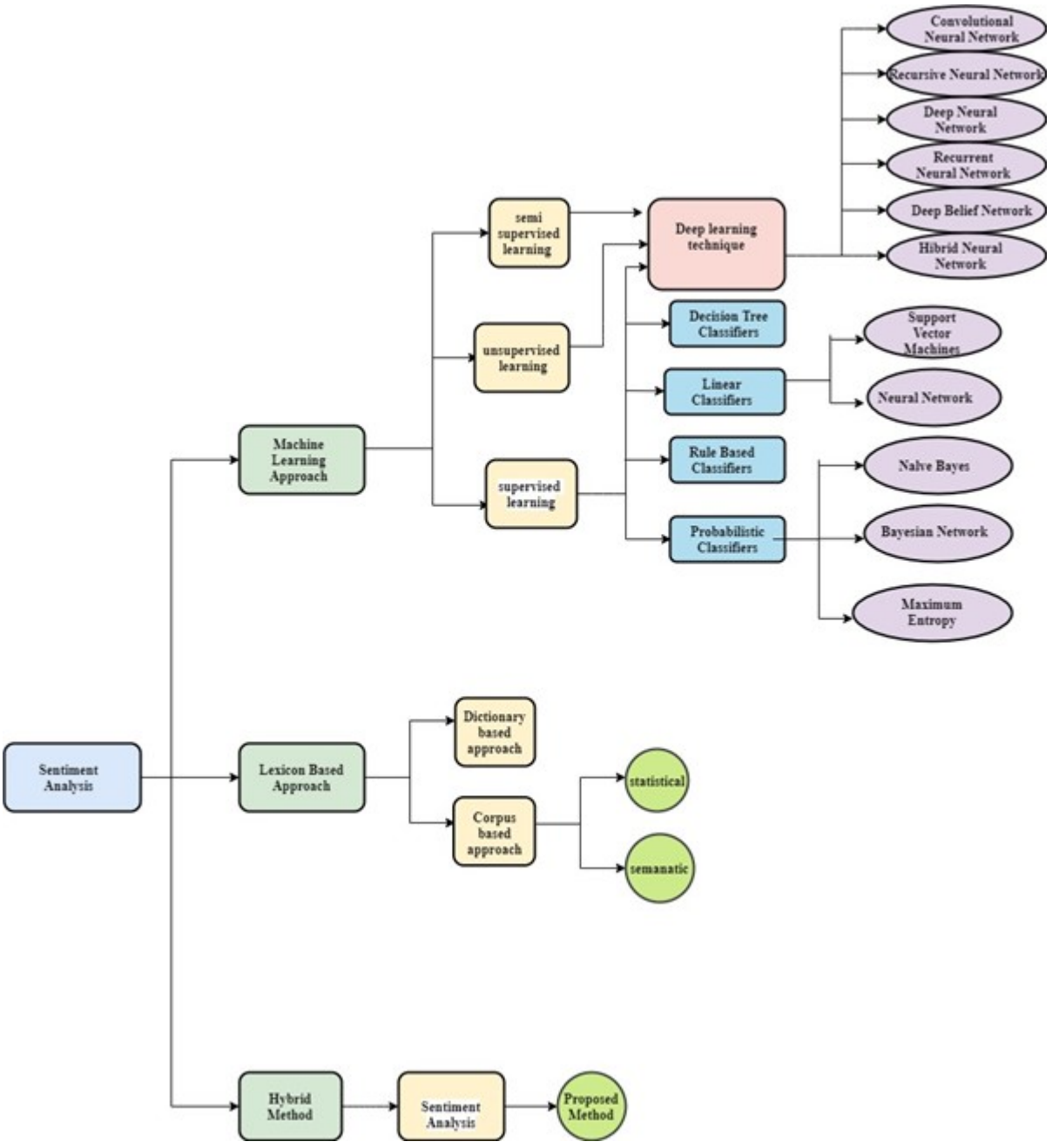


FIGURE 2.1: Sentiment Analysis approaches [48]

2.3.1 Lexicon Approach

Lexicon approach is also known as rule-based approach involves the use of a set of rules to identify the sentiment of a given text [29, 37, 69]. In rule-based approach, individual words are assigned a positive or negative polarity. The sentiment of the input text is

calculated by the number of positive/ negative words present. If a sentence has greater number of positive words, it indicates a positive sentiment sentence, else it is considered as a negative sentiment sentence [69]. To perform lexicon-based SA, the text is tokenized and then classified according to the set of defined rules. Examples of Lexicon sentiment analyzers are SentiStrength [69], SentiStrength-SE [37]. SentiStrength [69], a Lexicon Analyser used in [29] can identify sentiments in a short text. In the SentiStrength dictionary, positive words and emoticons are given a sentiment score ranging from +2 to +5, while negative words and emoticons are assigned a score of -2 to -5. Neutral words are assigned a score of +/- 1. For an input text, SentiStrength outputs two sets of sentiment scores ranging from +1 (not positive) to +5 (extremely positive) and -1 (not negative) to -5 (extremely negative). SentiStrength reports two sets of sentiment scores to convey mixed emotions contained in the input text. Additionally, SentiStrength reports binary (positive, negative), trinary (positive, negative, neutral) sentiment scores based on the algebraic sum [69]. However, SentiStrength when used to classify sentiments of SE text, it yields low accuracy. This is because the SE text contains technical jargons and customization for the SE domain is required [37].

Moreover, a SE domain specific lexicon classifier called as SentiStrength-SE was developed by Islam et al. in [37] achieved higher accuracy when compared to SentiStrength. The classifier included an SE-domain dictionary built on top of SentiStrength dictionary and achieved an overall an overall f-measure of 79.06%. Although, SentiStrength-SE achieves higher accuracy, there are limitations when employing the lexicon approach to classify sentiments.

2.3.1.1 Limitations of Lexicon approach

The lexicon classifiers are highly dependent on its sentiment dictionary to assign sentiment polarities and as a consequence the accuracy of the classifier is restricted by the fixed-size of the dictionary[34]. Moreover, lexicon analysers fail to identify the sentiment of sarcastic comments[37]. Furthermore, the text to be classified may contain words that have multiple meanings depending on the context used. For example, in the commit comment- "Updated in 1.2 branch. David; please download and try 1.2 beta when it is released in a week or so.." the lexicon-classifiers identify "please" as a positive sentiment word, although in this context "please" expresses a neutral sentiment as it

is used in a request. The lexicon classifiers assigns a positive polarity to the sentence when it is of neutral sentiment [37]. Thus, the lexicon-based analysers misclassify the sentiments of such sentences.

2.3.2 Machine Learning-based approach

To overcome the limitations of lexicon approach, various Machine Learning (ML) techniques are employed to perform SA and have been observed to achieve higher accuracy when compared to lexicon classifiers [4, 21, 13]. The ML approach require a labelled-dataset to learn and to identify the patterns present to optimize the future results. When using Machine learning approach the sentiment analysis task is modelled as a classification problem [46]. Machine learning approach is divided into supervised and unsupervised learning techniques. Supervised learning methods are used when there is a labelled dataset available, meanwhile unsupervised learning techniques are used when there is an unlabelled dataset to be classified [18]. The supervised learning techniques includes traditional machine learning algorithms such as the Naives Bayer, Support Vector Machine and also neural network based deep learning algorithms such as the Convolution Neural Network, Recurrent Neural Network.

A comparative study conducted by Ahmed et al. in [4], developed a supervised machine learning-based SA tool customised for the SE domain. The study compared the performance of the eight ML algorithms such as Naives Bayes, Gradient Boosting Tree (GBT), Linear Support Vector Machine, Random Forest, Decision Tree, Multilayer Perceptron, Support Vector Machine with Stochastic Gradient Descent (SGD) and Random forest. The tool developed-SentiCR, was trained and evaluated on a manually annotated Code-Review Comments dataset. Among them, it was found that GBT had the highest accuracy of 83.03%. Similar results were observed in [21] where an entity-level classifier-SentiSW was developed and trained on GitHub issue comments. Although, the study suggested that sophisticated rules are required in order to classify entities, comparison of the eight above-mentioned ML algorithms, found that GBT performs the best. Furthermore, SentiSW [21] and SentiCR [4] outperformed SentiStrength.

The accuracy of the classifiers incorporating traditional Machine learning techniques are highly dependent on the features extracted from the dataset, such as lexicon-features, keyword features, parts of speech and semantic features. Novielli et al. in [13] developed

an SE-specific classifier called Senti4SD. The classifier was trained using the SVM algorithm on the manually annotated StackOverflow dataset. Additionally, the classifier leveraged lexicon-based features, keyword-based features and semantic features based on word embedding. The classifier achieved an overall f-measure of 87%. On evaluation it was found that the combination of three extracted features provided best results. Hence, this study substantiates the claim that the accuracy of ML-based classifiers are highly dependent on the features extracted.

Traditional machine learning methods require Manual feature extraction techniques to reduce data complexity. Thus, requiring expertise and more time. Alternatively, in deep-learning models explained in section 2.4, all features are extracted automatically without the need to manually extract different features. Thus, outperforming the traditional machine learning based classifiers [40, 41, 6].

2.3.3 Hybrid approaches

The hybrid approaches combine the lexicon and machine learning techniques to perform sentiment classification. In the hybrid approach, the text is first classified using lexicon-based approach and the output produced is then fed into the machine learning based classifier as the training data [47]. Mukwazvure and Supreethi in [52] adopted the hybrid approach to develop a sentiment classifier to classify sentiments of news comments. The classifier employed lexicon approach to detect the sentiment polarity and then ML algorithms such as KNN and SVM were used to train on the labelled dataset. The labelled train dataset was obtained from the lexicon stage. This study observed that SVM classifier outperformed KNN, however suggested the use of a domain-specific lexicon dictionary to achieve better accuracy. Furthermore, previous study conducted by Filho et al. in [24] developed a hybrid classifier that incorporated Opinion Lexicon and Linear SVM trained on SemEval-2014 twitter dataset. The study observed that the accuracy of a well-tuned hybrid classifier gravitate towards the accuracy of ML based classifiers. Hence, the above studies lead to conduct the research in deep-learning approach.

2.4 Deep Learning approach

Over the recent years, Deep Learning (DL) has become increasingly popular to solve many NLP tasks. This is mainly due to the fact that Deep Learning algorithms can extract features automatically without the need for feature extraction techniques. Deep-learning methods use neural network architectures of many hidden layers to learn features directly from a dataset without having to perform manual extraction [27, 40]. The initial layers of the neural network architecture extract abstract features and the deeper layers capture meaningful information [26, 27, 40]. Hence, achieving a higher accuracy and performance. For the above-mentioned reason, this project will employ deep-learning algorithms to build the sentiment classifier. To build a Deep Learning based classifier, word embedding tools such as Word2Vec [49] and Glove [60] are used to convert the input sentence to vectors. These vectors are then fed into the neural networks as inputs. The neural network then learns the features and performs sentiment classification. The last layers of the neural network are a fully-connected layer and output the sentiment of input text.

In recent years, many studies have incorporated deep-learning techniques to identify sentiment polarity of text. This project examines the studies that have used Recurrent neural network, Convolution neural network, Long-Short term Memory, Bidirectional Long-Short term Memory as the deep-learning algorithms to perform SA.

2.4.1 Convolution Neural Network

Convolution Neural Network (CNN) is a feed-forward neural network generally used for image classification and in the computer vision field [27].

Recently, it has become popular in natural language processing (NLP) field due to its ability to extract features by applying the convolution filter.

The convolution neural network has the following four layers:

1. The input layer
2. Convolutional layer
3. Pooling layer

4. Fully-Connected layer

For the purpose of sentiment classification, at the input layer the input sentence to be classified is embedded at the word-level using the Glove, Word2Vec word embedding techniques [40, 41, 59]. This is represented as a matrix. The convolution filter is then slid over the word vectors that are generated to find the convolutions, producing feature maps [27]. Thus, the convolution layer captures important features for the feature maps. The pooling layer reduces the dimensional complexity, while preserving information of the convolutions or higher-level features [16, 27, 26]. An activation function such as Rectified Linear Unit (ReLU) is passed over the multiple feature maps that are generated. The weights of the neural network are optimized by the Loss Function [27]. Figure 2.2, illustrates the various layers of convolution neural network.

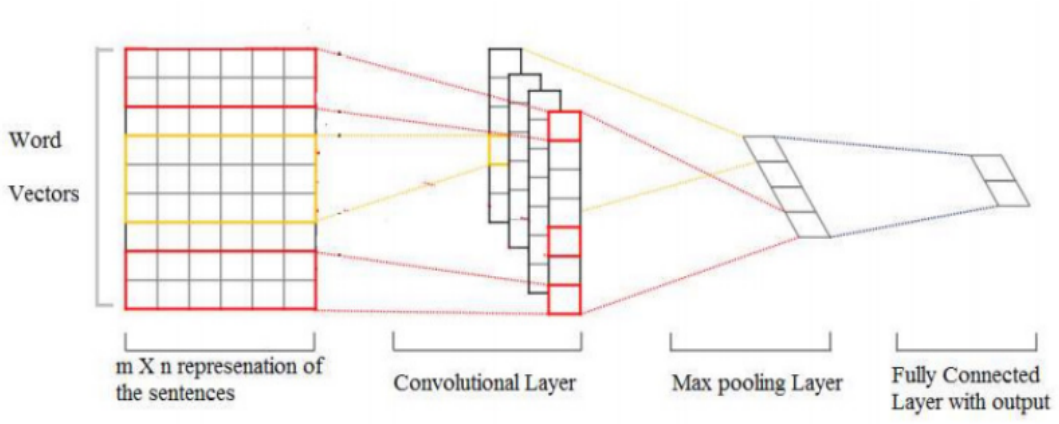


FIGURE 2.2: Architecture of a single-layer CNN [16]

Convolution Neural Networks have been proved to be very effective in solving SA tasks. Kim in [41] designed a one-layer CNN built on top of word embedding and achieved high accuracy results of 80% when tested on different datasets. The input layer of the CNN classifier comprised of word embedding vectors. The convolution operation involving multiple filters are applied to generate feature maps. This was followed by the max-pooling layer and finally a fully-connected layer with a dropout and a soft-max output. Similarly, a seven-layer CNN classifier was developed by Ouyang et al. in [59] employed word-embedding vectors generated via Word2Vec. The classifier consisted of three convolution layers where the input is processed, three max-pooling layers

that were arranged alternatively with respect to the convolution layers. The fully connected layer then outputs the sentiment. Although, CNN achieves high accuracy in the sentiment classification tasks [40, 41, 59], CNN does not share parameters due to its feed-forward/hierarchical neural network structure and hence cannot handle sequential data [27].

2.4.2 Recurrent Neural Network

Recurrent Neural Networks (RNN) is a type of artificial neural network with internal memory and so it can process sequential data [65, 27, 26]. Figure 2.3 [6] shows a diagrammatic presentation of Recurrent Neural Network. RNN is widely used in speech/voice recognition and in language translation tasks [26]. In RNN, the same task is performed for every element in the sequence such that the output of the current input depends on the current input and output of the previous input stage [27, 26]. Unlike the feed-forward models that allocate separate parameters for each value of the time-index and learns all the rules separately for each part, the RNN shares the parameters across several time-steps by adding feedback connections [27].

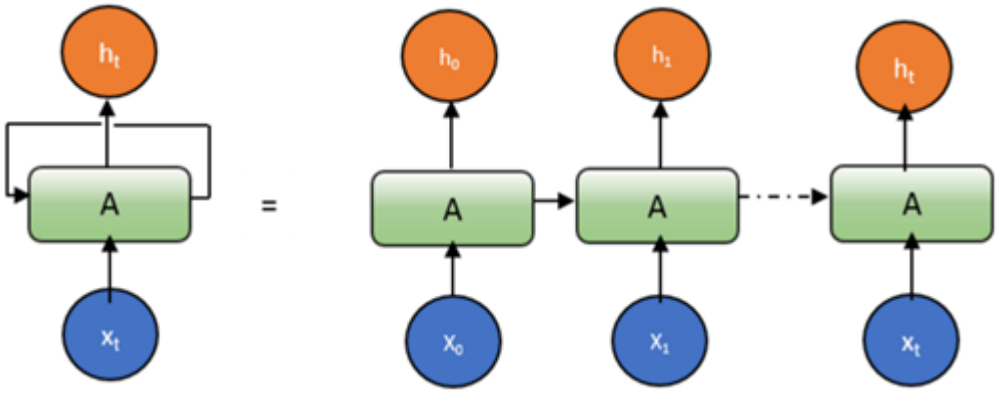


FIGURE 2.3: Unrolled Recurrent Neural Network [6]

Mathematically, the hidden state and output at time-step t is expressed by the equations (2.1) and (2.2) [23]. Where h_t is the hidden state vectors and x_t is the input vector and y_t is the output vector at time-step t. The activation function f is either RELU or tanh and w are the weights matrices and b is the bias [23].

$$h_t = \sigma_h(w_h x_t + u_h h_{t_{-1}} + b_h) \tag{2.1}$$

$$y_t = \sigma_y(w_y h_t + b_y) \tag{2.2}$$

The hidden layers of RNN remember the sequence of information through time to produce the output and so in text sentiment classification, RNN remembers the previous words to predict the next word in the sequence [50, 8, 6].

The chain structure of RNN makes it possible to handle sequential information. The study conducted by Patel and Tiwari in [43] provided the evidence that storing sequential information in a neural network can substantially increase the accuracy of the classifier. The RNN classifier was trained on IMDB movie reviews and achieved an accuracy of 87%. However, due to the vanishing gradient problem during backpropogation RNN is not suitable for handling long term dependencies [6, 23, 18].

2.4.3 Long Short-Term Memory units (LSTM)

To successfully train an RNN, many parameters need to be optimized such as the network structure, solution algorithm [73]. Moreover, the vanishing gradient problems of RNN make it unsuitable to handle long-term dependencies [6, 32]. LSTM is a type of Recurrent Neural Network that can remember inputs over a long period of time [32]. LSTM addresses the exploding gradient problem of RNN. In LSTM three gates are present as shown in Fig 2.4 determines the information to be retained and the information to be forgotten.

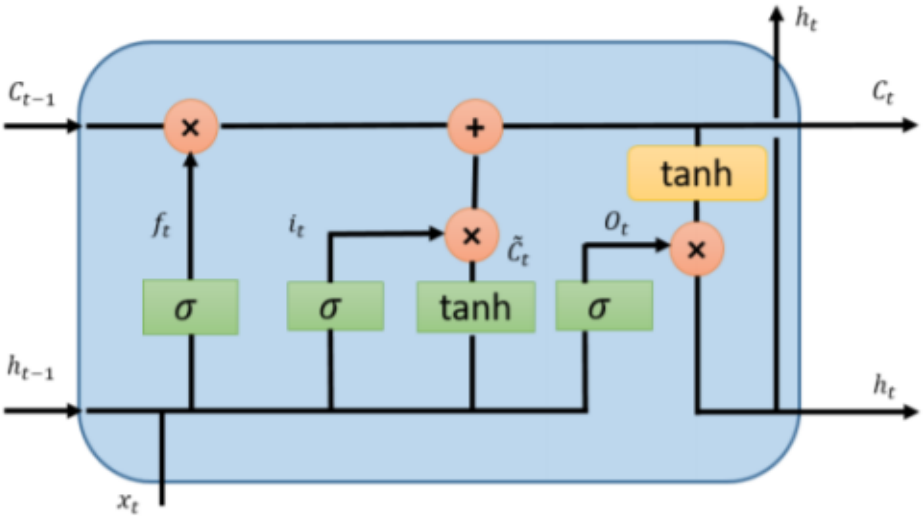


FIGURE 2.4: Long Short Term Memory [6]

1. Forget Gate

   The forget gate determines the information that can be forgotten or discarded by the block [32]. Information from the previous hidden state and from the current input is passed through sigmoid function. The forget layer is given by the equation (2.3), where $h_{t_{-1}}$ is the output from the previous hidden layer, x_t is the current input [6]. The equation outputs either 0 or 1 where 0 implies "forget" and 1 implies "keep" [6, 32]. The forget gate f_t controls the gradient passing through and is directly responsible for forgetting and updating the memory, hence addressing the vanishing gradient problem [32].

$$f_t = \sigma(W_f.[h_{t_{-1}}, x_t] + b_f) \tag{2.3}$$

2. Input Gate

   The input gate determines the relevant information to be added to the cell state. The tanh function creates a vector C_t which is added to the cell state. The cell state is updated by combining the below equation (2.4) and (2.5). Mathematically, the input gate is represented as follows [6]:

$$i_t = \sigma(W_i.h_{t_{-1}}, x_t + b_i) \tag{2.4}$$

$$\widetilde{C_t} = tanh(W_C.[h_{c_{-1}}, x_t] + b_c) \tag{2.5}$$

3. Output Gate

   The output gate decides the value of the next hidden state. The previous time-step cell state $C_{t_{-1}}$ is updated to C_t and this mathematically shown in the equation (2.6) [6].

$$C_t = f_t * C_{t_{-1}} + i_t * \widetilde{C_t} \tag{2.6}$$

   The output gate is given by equation (2.7)[6]. The LSTM runs the sigmoid function as a layer to decide the parts of the cell state to output.

$$O_t = \sigma(W_o.[h_{t_{-1}}, x_t] + b_o) \tag{2.7}$$

Finally, the tanh function of the cell state is taken and multiplied with the output of the output gate as given by the equation (2.8) [6].

$$h_t = O_t * tanh(C_t) \tag{2.8}$$

Given that LSTM remembers sequential data, Saeed in [61] developed an LSTM based classifier to classify IMDB movie reviews into positive and negative sentiments. The classifier achieved an accuracy of 89.9%. However, the study did not include identification of neutral sentiments. Similar performance results were observed by Kalaivani et al. in [39], the LSTM classifier was found to achieve the highest accuracy (approx. 80%)when compared with CNN and simple RNN. The model was trained on Gutenberg books dataset. The higher accuracy of the LSTM classifier is attributed to the remembrance of the sequence of the words. Additionally, the study conducted in [19] by Dashtipour et.al. observed high accuracy results when stacked LSTM classifier was used to identify sentiments of persian movie reviews. Significantly higher results were observed, due to the stacked architecture of the model, thereby making it deeper. Since LSTM processes the information only in sequential order, it is unable to access future information or context Bidirectional LSTM models are employed to increase the availability of the information in the network [75].

2.4.4 Bidirectional LSTM (BiLSTM)

Bidirectional LSTMs are an extension of traditional LSTMs. BiLSTM consists of two LSTMs that are applied onto the input data. One of the LSTMs takes input in the forward direction, while the other LSTM takes the input in the backward direction [75]. This increases the availability of the information to the network at any given time. The study conducted by Zhang et al. in [75] employed BiLSTM to identify sentiment polarity of chinese product reviews. The word embedding vectors were input to the BiLSTM layer and the softmax layer with sigmoid as the activation function were employed to map the vector representations of input text to its sentiment polarity. The classifier achievied a precision of 92.3%. However, the classifier performed only 2-class sentiment classification (positive and negative classes). Similary, the study conducted in [79] by Zhou et.al. employed a stacked BiLSTM model for Chinese microblogs sentiment classification. The classifier achieved an accuracy of 90.3% and 89.5% when skip-gram and

CBOW algorithms of the word2vec models were employed. Zhou et. al. attributed the improvement in the performance to the stacked structure of the model. Furthermore, the performance of CNN, LSTM, BiLSTM was evaluated by Barnes et al. in [9] on datasets of different domains. These classifiers were compared with the baseline models which were the L2- regularized logistic regression classifiers trained on bag-of-words (BOW) and average word vectors (AVE). The study found that the BiLSTM classifier achieved higher accuracies when compared to the baseline models on all datasets. Additionally, the BiLSTM classifier outperformed CNN and LSTM classifiers.

Sentiment classification when performed by adopting any of the above-mentioned approaches, requires the training dataset to be cleaned. SE-text consists of code snippets, URLs, whitespaces which need to be removed in order to increase the accuracy when the classifier model is introduced.

2.5 Text preprocessing

The accuracy of the sentiment classifier highly depends on the data that is fed into it. Cleaning the dataset helps in achieving a high accuracy. Since Software engineering text is different from articles, reviews and tweets a thorough preprocessing of the dataset must be adopted. The following are the preprocessing steps adopted in studies [4, 13, 12].

1. URL and code removal

   Clean the dataset by removing code snippets, HTML tags, URL links.

2. Removal of punctuations

   Remove punctuations such as ';', '/'. Punctuations play an important role in programming languages such as Java, C#, Python.

3. Convert dataset to Lowercase

   Convert all the text to lowercase, this avoids the deviations of the same word due to use of a different case.

4. Stemming and Stop-word

Stemming, a text-normalization technique and involves in getting the root word also known as stem of the word. Stop words include many semantic words such as articles and prepositions.

However, studies conducted by Novielli et al. in [13] has found that removal of stop words and performing stemming, omitted important sentiment information. However, we adopted a conservative approach to the removal of stop words and removed stop words that carried no sentiment.

5. Tokenization

   Tokenization is the process of splitting a phrase, sentence or a text document into smaller units or words called tokens. Feature extraction techniques like word-embedding are applied on the tokenized sentences to build a vocabulary of unique words [12].

2.6 Feature extraction

Sentiment analysis highly depends on the features extracted from the dataset to assign sentiment polarities to it. The studies conducted in [13], [20] indicates the feature extraction stage plays a crucial role in increasing the accuracy of the classifier.

2.6.1 Word Embedding

For a neural network-based classifier, all the text must be input as a vector of real numbers. Word embedding is a feature learning, feature extraction and a language-modelling technique where a word is transformed into a vector of continuous real numbers [12, 49]. In word embedding, a lower-dimensional dense vector is obtained from a high dimensional sparse embedding. Thus, reducing the dimensionality of text data. Word embedding captures the semantic relationship between the words, and so words that are related or have similar meanings are placed closed to each other in the representation space [13, 49, 12]. Hence when word embedding are used as input to the neural network based classifiers, the classifiers then generalizes the words and also finds the specific words present in the input text while identifying the sentiment polarity [12].

The Word2Vec tool [49] created by Tomas Mikolov, is the commonly used word embedding system, that uses neural networks to efficiently learn word embedding from text. It obtains the embedding using the Continuous bag of words (CBOW) and skip-gram models. In CBOW model, the target word is predicted from its context, whereas in the skip-gram model predicts the context words when given the target word. Software domain specific word-embedding developed by Efsathiou et al. [22] adopted the use of Word2Vec with the skip-gram model. Word2Vec tool was trained on StackOverflow dataset to obtain the word-embedding. Biswas et al. in [12] compared the accuracy of their neural network classifier when Google News embedding released in [49] and software-specific word-embedding was used. The software-specific embedding were developed by following the approach in [22]. It was found that the word embedding trained on Google News corpus performed better than SE-specific embedding. Our study aims to research the relation between the performance and the type of word embedding used.

Although sentiment analysis is useful in identifying developers' emotions, there are also many challenges and limitations faced when applying it to a domain as specific as the Software Engineering domain.

2.7 Challenges of Sentiment analysis

There are many challenges faced when applying sentiment analysis in software domain that inhibits the accuracy of which some of them are discussed.

1. Domain Dependency

   Software engineering texts consists of technical jargon that may be associated with a polarity in open-domain [37]. For example, in the text "The change you want would be nice ; but is simply not possible. The form data ... Jakarta FileUpload library." is misclassified as a positive sentiment polarity due to the presence of the word "would be nice", but the sentence in fact expresses neutral sentiment [37]. Similarly, words like bugs, error, default does not express any particular sentiment in the SE domain, but in open domain these words are given a negative polarity [13, 34, 37].

2. Negation Handling

It is important for sentiment classifiers to identify negation terms present in the sentence. Effective handling of negation terms increases the overall efficiency and the performance of the classifier, thereby reducing the misclassification of neutral sentiment samples as negative [4, 67]. The negation handling approach implemented in [4] involved in prepending a "not_" in the succeeding words of the sentence when a negation term was identified. Additionally, each word was tagged using NLTK parts of speech (POS) tagger and specific chunk grammar rules were created and used. This increased the recall in identifying identifying negative and neutral samples for complex code review comments. Thus, sophisticated techniques are required to effectively handle negations in sentences.

3. Misinterpretation of irony and sarcasm

   Identifying irony in the input text is a challenging task for the Sentiment Analysers. Sentiment Analysers misinterpret the irony present and assign an opposite sentiment polarity to it [13, 34].

2.8 Related work

This section discusses the various work related to SA in SE domain employing multiple approaches such as lexicon, machine-learning and deep-learning techniques.

2.8.1 Previous works on Lexicon Analysers

The studies conducted in [25, 29, 54] provided the evidence that software developers express their sentiments and emotions on crowdsourcing platform, code repositories, issue tracking systems. Garcia et al. in [25] and Guzman et al. in [29] studied the relation between the sentiments expressed and their overall contribution to the project. Guzman et al. in [29] used SentiStrength, a lexicon analyser to classify the sentiments. However, Jongeling et al. in [38] compared four domain independent sentiment analyzers (those of which are SentiStrength [69], Alchemy[1], NLTK [10], StandfordCoreNLP [68]) found that existing sentiment classifiers disagreed amongst themselves and the labelled sentiments. This study substantiates the claim that off-the-shelf sentiment analyzers

[1] http://www.alchemyapi.com/products/alchemylanguage/sentiment-analysis/

yield low accuracy and precision when used in SE domain. Hence, customization of Sentiment Analysis tools specific to that domain is necessary. To address the problem of domain dependency, Islam and Zibran in [37] proposed a SE-specific sentiment analysis built on top of SentiStrength called SentiStrength-SE. SE domain-specific dictionary was developed and built upon the SentiStrength dictionary. Evaluation conducted by Islam et al. reported that SentiStrength-SE outperformed the original SentiStrength [69] sentiment analyser. Thus, proving that incorporating domain-specific knowledge in the SA tool can substantially improve the classification result. However, the accuracy of the analyzer is restricted by the quality and the fixed size of the sentiment dictionary.

Alternatively, machine learning and deep learning techniques do not require a domain specific dictionary, but need a labelled dataset to train and identify patterns present to classify the sentiment polarity of the given text [4, 13, 62, 21].

2.8.2 Previous works on Machine Learning Analysers

Sentiment analyzers that are trained using machine learning algorithms on a SE dataset have higher accuracies when compared to lexicon-based sentiment analyzers [4, 13, 62, 21].The SentiCR tool developed by Ahmed et al. in [4] achieves an accuracy pf 83.03% when trained using the Gradient Boosting Tree algorithm. The classifier was trained on the Code Review Comments dataset and extracted features by computing TF-IDF (Term Frequency – Inverse Document Frequency). Additionally, the classifier employed negation handling techniques which increased the accuracy of the classifier. Moreover, SentiCR outperformed the existing off-the-shelf supervised machine learning based sentiment analyser tools such as NLTK [10] and Vivekn[55] thus proving that training ML-based classifiers in a SE-specific dataset enhances the accuracy of classifying sentiments. However, a limitation in SentiCR tool is that the tool does not discriminate between neutral and positive sentiments. Rather SentiCR assigns a polarity of -1 when negative sentiment is detected and displays a 0 when a non-negative sentiment is detected.

A machine learning based sentiment analyser to categorise the sentiment of the input text to positive, negative and neutral sentiments was developed by Novielli et al. [13]. The study aimed to reduce the misclassifications of neutral sentiment texts as negative polarity. This was achieved by extracting lexicon, keyword, and semantic features from the text. The classifier was trained on the gold-standard StackOverflow dataset. SVM

algorithm was used to train Senti4SD as it can generalize a high-dimension vector space and so achieved f-measure of 87%. Therefore, presenting strong evidence to the claim that extracting lexicon, keyword and semantic features boosts the accuracy of the classifier, by successfully reducing misclassification of sentiment polarity for the input text. Furthermore, Novielli et al. [13] and Ahmed et al. [4] compared the accuracy of the developed classifiers with SentiStrength and substantiated the claim that ML algorithms outperform off-the-shelf Lexicon Analysers.

However, a comparative study on SE-domain SA tools (SentiStrength [69], Senti4SD [13], Emotxt[14]), conducted by Islam and Zibran in [36] on three datasets from StackOverflow posts, Jira issue comments, and code review comments found the accuracy of the above mentioned tools varying across the datasets. It was observed that the tools performed best on their originally tested dataset and the overall accuracy decreased when tested on other SE-domain datasets. Similarly, the study conducted by Lin et al. [44] also confirms that current available tools are not ready for the SE-domain sentiment analysis. Hence there is a need for a reliable Sentiment Analysis tool.

2.8.3 Previous works on Deep-learning

Deep learning algorithms extract all the features present in the dataset automatically, without the need for explicit feature extraction techniques. The comparison study conducted by Kansara and Sawant in [40] compared traditional machine learning approaches with deep-learning approaches for SA. Machine learning models such as Naives Bayes, Logistic Regression and Random forests were compared with LSTM, CNN and a hybrid model of CNN+ LSTM as the deep-learning approaches. The machine learning and Deep Learning models were trained and evaluated on three datasets, the IMDB movie reviews, tweets, and hotel reviews datasets. The study concluded with the finding that the deep-learning models significantly outperformed the machine learning models. Additionally, deep-learning classifiers built in [41, 59, 43, 39, 61, 9] employed CNN, RNN LSTM and BiLSTM algorithms to build sentiment classifiers. These classifiers were trained on open-domain datasets and yielded high accuracies. The above-mentioned studies employed a single layer of CNN, LSTM, BiLSTM in their sentiment classifiers to extract word2vec features. However, the word2vec features are highly complex and often require more than one layer to extract the features [79]. The study conducted by Zhou

et.al. in [79], observed that stacking the BiLSTM layers decreased the prediction loss and increased the prediction accuracy and f-measure. The author attributes the better performance to the structure of stacked model that makes the model deep, thereby extracting the complex features word2vec more effectively. Similarly, studies conducted in [59, 77, 51, 7, 35, 19] employed stacked layers of CNN, LSTM and BILSTM algorithms in the sentiment classifiers and achieved f-measure results significantly higher than the classifiers employing single layers of CNN, LSTM and BiLSTM algorithms for the open domain. The above studies direct us to build a sentiment classifier for SE domain by stacking layers of the deep learning algorithms

Furthermore, it is observed that the dataset selected plays an important role in Sentiment Analysis. SE-domain datasets are highly imbalanced, and datasets usually contain more neutral sentiments samples when compared to the positive and negative sentiment samples [12, 44]. The impact of an imbalanced dataset is observed in [44]. The authors customized off-the-shelf tool StandfordCoreNLP. StandfordCoreNLP [68], a Recursive Neural Network based classifier was trained on 1500 sentences of manually annotated StackOverflow dataset. Out of the 1500 sentences, 178 were of positive polarity, 131 were negative and the remaining 1191 sentences were annotated as neutral sentiment polarity. The dataset was split into 90% train and 10% test set. The SE-customized classifier yielded a recall and precision below 40% for negative and positive sentiment sentences. It is observed that the classifier obtains low recall and precision mainly due to the skewed class distribution of the dataset. As noted earlier, the poor results yielded are due to the higher distribution of the neutral sentiment sentences.

However, to address the problem of skewed class distribution, Biswas et al.[12] implemented undersampling and oversampling techniques on the datasets. Their work developed a RNN classifier that was trained on the StackOverflow dataset. The dataset was procured from the study of Lin et al. in [44]. A balanced dataset was created by the simultaneous under-sampling of the majority neutral class and oversampling of the minority positive and negative classes. It was observed that this approach of undersampling and oversampling substantially increased the recall values for the negative (56%) and positive classes (46%). Moreover, the comparative study by Mabrouk et al. in [47] substantiates the claim that the performance of the classifier is higher when a labelled and balanced dataset is used when compared to unlabeled and imbalanced datasets.

Hence the above study direct towards the adoption of a dataset containing well-balanced class distribution of sentiment polarities.

The below table 2.1 summarizes the previous work done in sentiment analysis using Lexicon, Machine Learning and Deep Learning approaches. As observed, traditional Machine learning approaches adopted in [4] and [13] outperform the lexicon SE-specific Sentiment Analysis tool in [37]. Moreover, studies conducted in [40][19] observed that stacked Deep Learning classifiers outperformed traditional Machine Learning classifiers.

Paper	Proposal	Dataset	Feature Extraction	Algorithm	Precision/ Accuracy/ F-measure
[37]	Built a domain specific dictionary for SE to identify sentiments accurately	Gold-standard JIRA issue tracking dataset	Lexicon features	SE-specific dictionary was built to classify sentiments in the lexicon approach by computing the algebraic sum	Achieved f-measure of 79.06 %
[4]	Compared Machine Learning algorithms to build an accurate classifier	GitHub Code review dataset	TF-IDF	ADB, NB, GBT, Random Forest, Decision Tree, MLP, SGD, SVC	F-measure of 62% for the GBT classifier
[13]	Reduce misclassification of neutral sentiments as negative.	Gold-standard Stack-Overflow dataset, manually annotated	Lexicon, keyword and semantic features	SVM	Overall f-measure of 87%
[40]	Compared ML and DL techniques	1-IMDB Movie Reviews 2-Hotel Reviews 3-Tweets	For ML- Bag of Words (BOW) used For DL- All features extracted using Word2Vec to get the features as vectors.	CNN, LSTM, CNN+LSTM	CNN+LSTM obtained the following accuracy:88.28% for IMDB reviews, 81.29% for Hotel Reviews 74.92% for tweets.

[41]	1-layer CNN with hyper parameter can achieve high accuracy for SA.	Movie Review, Stanford Sentiment Treebank, Subjectivity dataset, TREC question dataset, Customer Review dataset, MPQA-Opinion dataset	Input text features extracted using Word2Vec	CNN	Accuracy of 89.6% for MPQA dataset
[59]	5-class fine grained Sentiment Analysis using CNN and Word2Vec to obtain vectors. The model was compared with Recursive Neural Network and Matrix-Vector Recursive Neural Network	Movie Reviews dataset from rotten tomatoes	Input text features extracted using Word2Vec	7-layered CNN	Accuracy of 45.4%
[61]	Develop LSTM based Sentiment classifier	IMDB movie reviews	Vectorization done by Doc2Vec	LSTM	89.9 %

[79]	Enhance the performance of sentiment classifiers by employing stacked deep learning models	Weibo dataset	Word2vec to capture semantic meaning of text	Stacked BiLSTM	90.3% accuracy for Skip-gram algorithm based word2vec model.
[44]	Build a software library recommender system by customizing off-shelf SA tool	Manually annotated Stack-Overflow dataset	StandfordCoreNLP tool was customized to SE domain	StandfordCoreNLP was developed using Recursive neural network	Precision of positive and negative classes below 40%
[12]	Improve the accuracy of the sentiment classifier for SE using neural networks	StackOverflow dataset from Lin etal.[44]	Vectorization of input text (features extracted) using Word2Vec. Compared generic and SE-specific word embedding.	RNN	Precision obtained 56% for negative and 46% positive and 89.3% for neutral classes
[19]	Employed stacked LSTM model to improve classification performance of Persian moview reviews	Persian movie reviews	fastText embedding	Stacked LSTM, Stacked BiLSTM	F-measure of 94%, 96% for stacked LSTM and stacked BiLSTM respectively

TABLE 2.1: Summary of Previous works

2.9 Rationale

Although, the above summarized studies employ various techniques for sentiment analysis, it has been observed that machine learning and deep learning techniques are far by the most popular. Additionally, it has been observed that the current SA tools for SE cannot effectively discriminate the positive, neutral and negative sentiments [44]. Thus, deep-learning algorithms such as CNN, LSTM and BiLSTM are employed to build the sentiment classifier. The developed classifiers will comprise of stacked layers of the above-mentioned algorithms. The stacked classifiers are used in sentiment analysis for open domains and various classification problems and have produced outstanding results, since they are able to extract the complex Word2Vec features [79]. Thus we want to compare the performance of the stacked classifiers developed using these algorithms (CNN, LSTM, BILSTM) for SE domain sentiment analysis. We believe, this is the first study to employ stacked deep learning models for the SE-domain sentiment analysis and to compare their performance. Additionally, this study explores if the use of SE-specific word embedding is required to guarantee the success of sentiment analysis. Finally, the performance of the developed classifier against the baseline methods specified in section 5.9 is compared.

Chapter 3

Requirements Analysis

This chapter details the functional and non-functional requirements in building a sentiment classifier for software engineering dataset. Additionally, the evaluation methods that will be used to evaluate the developed models are detailed.

3.1 Functional Requirements

The StackOverflow and Jira datasets developed by Novielli in [13] and Ortu et.al. in [58] will be used to train the classifiers. The datasets are further described in the section 5.1. Stratified Cross Validation technique is applied to ensure the equal distribution of the classes during training and testing.

3.1.1 Software Engineering Sentiment Analysis

The aim of this project is to build an accurate sentiment classifier to classify the sentiments of the developers expressed in crowd sourced platforms to positive, negative, neutral. Deep learning-based classifiers are developed to achieve the project goal.

3.1.1.1 Creation of SE-specific word embeddings

In this study, we compare the performance of the classifiers when SE-specific embedding and generic embeddings are used. To achieve this requirement, we build a SE-specific word embedding model using a large corpus of SE-text.

3.1.1.2 Building the classifier

Once the datasets are divided into train and test set, different deep-learning algorithms will be compared and evaluated. Three individual classifiers will be built that will employ stacked layers of CNN, LSTM and BiLSTM as the deep-learning algorithms. The precision, recall and F1-measure metrics are evaluated and compared for the developed classifiers to determine the most suitable algorithm for sentiment analysis for the SE domain.

3.1.2 Create a standalone system

A standalone version of the classifier must be developed where an SE-specific sentence is input and the sentiment of the sentence is displayed.

3.2 Non-Functional Requirements

3.2.1 Documentation

A thorough documentation of the code is important and will be implemented using Sphinx and Python docstrings. Python programming language will be used to build the classifier.

3.2.2 Handling of user inputs

The standalone system developed must be able to handle erroneous user inputs.

The below table 3.1 details the requirements for the project.

Requirement Type	MoSCoW	Name	Technique	Priority
Functional	Must	Acquiring Dataset	Gold-Standard dataset publicly available. Will use Stratified k-fold cross validation technique to ensure equal division of train and test sets.	High
Functional	Must	Integrate StackOverflow datadump from BigqueryAPI	Public dataset for StackOverflow datadump is available on Google's BigQuery API. The datadump will be used to build software-embedding	High
Functional	Must	Develop software-specific word embedding model	Word2Vec tool in CBOW algorithm used	High
Functional	Must	Use pre-trained word embeddings	Google News word embeddings are publicly available.	High
Functional	Must	Building Sentiment Classifier	Program written primarily in Python. Keras, Tensorflow libraries will be used.	High
Functional	Must	Evaluation and analyzing of the Results obtained for all DL techniques	Calculate F-measure, Recall, Precision.	High

Functional	Could	Compare the results obtained with the existing sentiment classifiers for SE	Compare the evaluation metrics for the developed classifier and the existing classifiers	High
Functional	Should	Create a standalone system for SA	Code written in Python	Medium
Functional (Optional)	Could	CLI application to train the classifiers on any given dataset.	Code written in python	Medium
Non-Functional	Should	Documentation	Thorough documentation of the code by applying comments and sensible variable names.	Medium
Non-Functional	Should	Handling erroneous user inputs.	This will be handled by the usage of exceptions in the program.	High

TABLE 3.1: Requirement Analysis

3.3 Cross Validation

Stratified k-fold cross validation is a resampling technique that is used to evaluate deep-learning models when only limited data is available [74, 42]. Additionally, this technique is applied to ensure the dataset is split into k different groups of equal sizes and containing equal class distribution in each fold. Hence this reduces the possibility of the training and testing set not being represented [74]. In stratified k-fold cross validation technique, the deep-learning classifiers are trained using the data present in the (k-1) groups and are tested on the remaining group and this is repeated for 'k' number of

times [42]. The performance of the classifier is the average of all 'k' executions of the model [42].

3.4 Evaluation Criteria

To evaluate deep-learning classifiers, multiple factors must be considered, obtaining high accuracies alone does not warrant the developed classifiers are working as expected. In fact, in a multi-class classification problem such as the one in our study, a classifier tested on a testing set that contains 90% of the data belonging to certain class would obtain an accuracy of 90%, however this classifier would not perform well when exposed to samples belonging to other classes. Henceforth, the performance for each of the classifier is measured in terms of precision, recall and f-measure. Moreover, the above-mentioned evaluation metrics adopted in our study is in accordance to the standard evaluation metrics adopted in SA benchmarking studies for SE [57, 56, 17]. The confusion matrix must be detailed for the purpose of explaining the evaluation metrics.

3.4.1 Confusion Matrix

Confusion Matrix is a NxN matric used for summarizing the performance of a classification model, where N is the number of classes [72]. As shown in the figure, the rows represent the actual values, whereas the columns represent the predicted values. The terms 'True' and 'False' indicate whether the samples are correctly or incorrectly classified. The confusion matrix reports the values for True Positive, True Negative, False Positive and False Negative which are defined as follows [72]:

True Positive- The classifier accurately predicts the positive class.

False Positive- The classifier incorrectly predicts the samples as positive class.

True Negative- The classifier correctly labels the samples belonging to the negative class.

False Negative- The classifier incorrectly labels the samples belonging to the negative class.

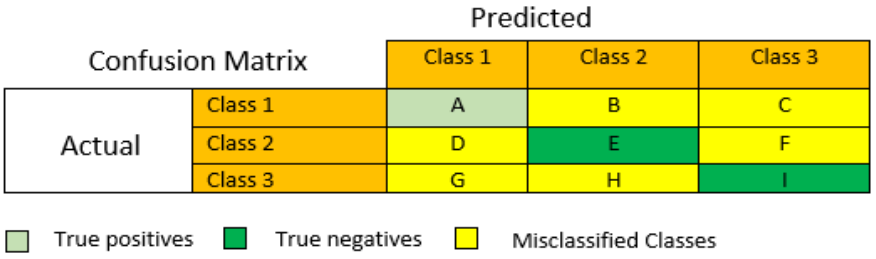


FIGURE 3.1: Confusion Matrix[5]

3.4.2 Precision

Precision measures the number of the correct predictions made by the model by calculating the number of relevant items from all the retrieved items. In this case, precision is the model's ability to identify positive, negative and neutral sentences and it is the ratio of correctly predicted positive samples to the total number of samples [28] and is given by equation 3.1.

$$\frac{TruePositive}{TruePositive + FalsePositive} \tag{3.1}$$

3.4.3 Recall

Recall is the model's ability to detect samples belonging to a certain class accurately. Recall is calculated as the ratio of the number of true positives to the total number of true positives and false negatives and is given by equation 3.2 [28].

$$\frac{TruePositive}{TruePositive + FalseNegative} \tag{3.2}$$

3.4.4 F-measure

F-measure is the harmonic mean of precision, recall and it is given by equation 3.4 [51]. F-measure will be used in place of accuracy to measure the overall performance of the classifier. This was done because the accuracy of an imbalanced class distribution in a

dataset would lead the accuracy of the model to tend to the majority class. Hence not painting the accurate picture of the performance of the classifier. Thus, f-measure will be used.

$$\frac{TruePositive}{TruePositive + \frac{FalseNegative + FalsePositive}{2}} \tag{3.3}$$

Chapter 4

Methodology

The methodology this project adopted is the Waterfall model. It is a sequential development process in which we analyze, design, implement and evaluate the different algorithms for our model. This methodology demands full planning and documentation in advance, thus many mistakes are avoided and the whole process is more efficient. The methodology is divided into the following phases, each of which is detailed in the subsequent sections.

1. Requirement phase
2. Design Phase
3. Implementation
4. Evaluation

4.1 Requirements phase

In this phase, the requirements for the GUI application is analysed. It is required that the GUI application takes an input from the user and output the identified sentiment based on the classifier and Word2Vec model selected.

4.2 Design phase

Python programming language is used to develop the application. The MVC design pattern is followed to maintain the software developed easily. Additionally, the application uses Tensorflow to develop the deep-learning classifier models such as CNN, LSTM and BiLSTM. The study is conducted using an Intel i7-8750H CPU @ 2.21 GHz, 16 GB memory, Nvidia GTx1060 Max-Q, and 64-bit Windows 10 Operating System. Jupyter notebooks and Pycharm IDE are used to build classifiers and conduct evaluations. Various libraries are used to develop the classifiers and GUI application, which are detailed below.

4.2.1 Tensorflow

The Tensorflow[1] library developed by Google is an open source library to create the deep learning models that is used in building the sentiment classifier.

4.2.2 Genism

The Genism[2] Library is an open source python library for Natural Language Processing. Genism Library enables to generate word embedding trained on a specific corpus using CBOW and skip-gram models.

4.2.3 NLTK

NLTK[3] short for Natural Language Toolkit, is a python library that is useful for tokenizing, semantic reasoning, word-embedding,classification and tagging. It is an open-source library developed by Steven Bird, Edward Loper and Ewan Klein.

[1] https://www.tensorflow.org/
[2] https://pypi.org/project/gensim/
[3] https://www.nltk.org/

4.2.4 Keras

Keras[4] is an open-source neural network library in python. It was developed in 2015 by Francois Chollet.

4.2.5 NumPy

NumPy[5] short for "Numerical Python" is an open-source library in Python that is useful for working on high-dimensional arrays. Employing NumPy makes it easier to reshape, splice, sort an array data. It was developed in 2005 by Travis Oliphant.

4.2.6 Pandas

Pandas[6] short for "Python Data Analysis Library" is a library in Python that is useful for analyzing and manipulating data. Pandas are open sourced and are under BSD License. Pandas make it easier to read CSVs, TSVs and MYSQL data and present it as data frames. Employing Pandas, one can use various functions like sort, filter, group by etc.

4.2.7 Tkinter

Python Tkinter[7] is used to build the GUI application.

4.2.8 ArgParse

ArgumentParser[8] is the command-line parsing module in Python.

[4] https://keras.io/
[5] https://numpy.org/
[6] https://pandas.pydata.org/
[7] https://docs.python.org/3/library/tkinter.html
[8] https://docs.python.org/3/library/argparse.html

4.2.9 BigQuery

Google's BigQuery[9] API is used to query through massive data tables by writing SQL queries at lightning-fast speed. The bqhelper package which is under Apache 2.0 license is used to further simplify the execution of queries and read-only BigQuery tasks.

4.3 Implementation phase

The implementation of application and classifiers are discussed in more details in chapter 5.

4.4 Testing phase

The testing phase involves in writing unit tests to validate preprocessing process for an input sentence. Additionally, the program is run multiple times to make sure no requirements were forgotten and also to take notes of any new problems that may arise.

The figure 4.1 depicts the waterfall model adopted in the development of this project.

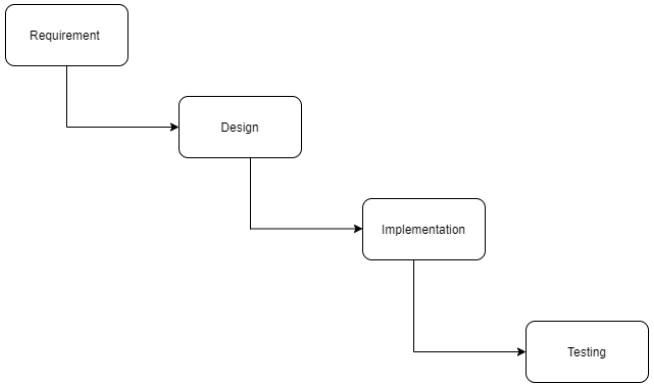


FIGURE 4.1: Waterfall methodology

[9] https://cloud.google.com/bigquery

Chapter 5

Implementation

This chapter presents the implementation details regarding the preprocessing of the dataset, creation of the word embedding model, training of the classifiers and the experiments performed with the multiple models.

5.1 Dataset Preparation

To train and evaluate the developed deep-learning classifiers, SE-related datasets from different platforms are used. The StackOverflow and Jira datasets developed by Novielli et al. and Ortu et al. in [13, 58] are released online and are annotated with sentiments for the purpose of SE-sentiment classification. The datasets are detailed thoroughly in the following sections.

5.1.1 StackOverflow Dataset

The StackOverflow dataset consists of 4423 user contributed StackOverflow questions, answers and comments from July 2008 to September 2015. To ensure the creation of a balanced dataset, SentiStrength a lexicon sentiment analyzer was used to check the presence of sentiment lexicon in the post. Subsequently, each of the post was then manually labeled as positive, neutral and negative by three coders. Posts that were annotated with conflicting sentiment polarities were ignored and the final labels of each of the sample was determined by majority voting. The dataset consists of 34.5% positive,

38.3% neutral and 27.6% negative samples. The dataset consists of three columns namely id, which represents the id number assigned to the sample, a label column which contains the sentiment labels for each of the samples, and the sentence column that contains the SO post as shown in fig 5.1.

	id	Label	Sentence
0	t3	positive	Excellent, happy to help! If you don't mind, can you accept my answer?
1	t53	positive	Paradise, if you submit that as a separate response I'll make you the winner!
2	t54	negative	Now when I load it to the device, an image won't show up. This is all kinds of horrible!
3	t74	neutral	If I understand your question correctly, the exception is because variable is not found inside the function. I'd suggest learning scoping rules in Python or read this excellent to-the-point chapter on the topic (or search for this topic on the Web!).The other thing I'd recommend is not to handle all exceptions as you've shown in your snippet. Here's why.The fixed code could look something like this:
4	t89	negative	Everytime I execute my app in debug mode, the execution breaks inside some class file of an external library that I'm using. Even though it's a NullPointerException, for some reason I can hit resume and continue the execution. In case you're curious, I've put the stacktrace below. My problem though is that I simply want to stop that. It's extremely annoying and happens every now and then during the execution. But I can't find any setting anywhere which will prevent breaking inside class files. How do I avoid this?

FIGURE 5.1: Example of samples in StackOverflow dataset

5.1.2 Jira Dataset

The dataset developed in [58] contains 5992 Jira issue comments divided into Group-1, Group-2, Group-3 comprising 392, 1600, 4000 issue comments accordingly. We followed the approach adopted by Chen et al. in [17]and excluded the issue comments of Group-1. The dataset is manually annotated with emotions as labels and as per the previous study conducted in [57], the issue comments labeled as love, joy are assigned a positive sentiment label, whereas the issue comments labeled as sadness and fear are awarded a negative sentiment polarity and the comments labeled as neutral are given a neutral sentiment label. Furthermore, issue comments containing surprise and conflicting emotion labels are excluded. The final dataset comprises 2571 issue comments from Group-2 and Group-3. The dataset contains 42.3% positive, 29.8% negative, and 27% neutral sentiment labeled issue comments. The dataset consists of two columns containing the issue comment and the sentiment label assigned to each of the issue comment as depicted in fig 5.2. The labels -1, 0, 1 imply negative, neutral and positive sentiment respectively.

	Sentence	Label
0	Thanks for fixing it!	1
1	WOW....you are right...I'm deeply sorry to bother you guys (and I'm also a bit embarrassed).	-1
2	Somehow it seems that this problem is more on the rubygems side. I didn't have time to investigate it any further but looks like rubygems is using @gems internally sometimes as a hash and sometimes as an array. You could try doing the following change in rubygems/source_index.rb (line 410): 410c410 < @gems.replace(new_index.gems) --- > @gems = Hash[*new_index.flatten]	0
3	Also your username was stupidly named and confusing so I deleted it and created a new one more appropriate.	-1
4	User specified directory is not created by agent	0

FIGURE 5.2: Example of samples in Jira dataset

Text data is the most unstructured form of data; hence preprocessing is considered as a vital task in SA to remove absurd information from raw text and has a direct impact on the performance of the deep learning classifiers.

5.2 Data Preprocessing

The preprocessing of the datasets was done by following the approaches adopted in [13, 12, 22, 17]. The preprocessor was written using the Regex library of python and is divided into the following steps. The figures 5.3, 5.4 depict the above-mentioned datasets before preprocessing.

	id	Label	Sentence
5	t108	positive	If you only need the count (like you express in your original question), you could do: Hope it helps!
6	t126	neutral	What will you be doing with the images? Displaying them in the browser?
7	t128	negative	It is really painfull to get all code form TFS whenever you make a build using NANT. Is there any settings in ccnet.config which will only take the files which are changed form last time which can speed up the process.
8	t183	neutral	Added bounty to see if I can get a full answer (ideally: the code snippet that, when added above, answers the question).

FIGURE 5.3: StackOverflow dataset before preprocessing

	Sentence	Label
6	Sorry that I think I missed some discussion in the mailing list. The implementation and test and BreakIterator is attached.	-1
7	Sorry in rushing to commit I forgot to add author during the commit.	-1
8	Direct link to mail item didn't work.	0
9	Thanks for raising these documentation issues! I just committed a fix.	1

FIGURE 5.4: Jira dataset before preprocessing

5.2.1 Expansion of Contractions

Contractions in the English language are the shortened form of word that omits certain letters. The preprocessor expands the contractions to standardize the text in the dataset.

5.2.2 Stop-word removal

In contrast to the preprocessing approaches adopted in [13, 12] wherein stop words were not removed, our study adopted a conservative stop-word removal approach. In this approach stop words that frequently appeared and did not carry any sentiment information such as "the", "on", "my", etc. were removed as shown in fig 5.5.

```
"""
Removes the stop words present in sentence
    Args:
        text (str): sentence
    Returns:
        Sentence without the specified stop words
"""
def remove_stopwords(text):
    # Custom stop word list
    stop_words_list =  ['the', 'i', 'to', 'is', 'a', 'it', 'and', 'you', 'in', 'that', 'of', 'this',
                'have', 'for', 'with', 'on', 'am',
                'are', 'if', 'my', 'an', 'as', 'would', 'your', 'there','has','then']
    for word in stop_words_list:
        pattern = r'\b'+word+r'\b'
        text = re.sub(pattern, '', text)
    return text
```

FIGURE 5.5: Code snippet for removal of stop words

5.2.3 Removal of URLs

The preprocessor removes URLs using Regex as these snippets do not convey any sentiments.

5.2.4 Removal of Punctuations

We adopted a conservative approach to punctuation removal and removed all punctuation except '.', '!', '?' as they are often used as delimiters and are also found to convey sentiments [12, 22]. Additionally, normalization of the text is done by converting it all to lowercase.

5.2.5 Tokenization

The text corpus was tokenized by using the tokenizer package of tensorflow. The text was vectorized and converted to sequences of integers and is padded. Figure 5.6, 5.7 depict the cleaned datasets in which the contractions are expanded, stop-words, punctuations,

URLs are removed and the text is converted to lowercase. Additionally, table 5.1 details the characteristics of the StackOverflow and Jira datasets after preprocessing.

	id	Label	Sentence
5	t108	positive	only need count like express original question could do hope helps!
6	t126	neutral	what will be doing images? displaying them browser?
7	t128	negative	really painfull get all code form tfs whenever make build using nant. any settings cnet.config which will only take files which changed form last time which can speed up process.
8	t183	neutral	added bounty see can get full answer ideally code snippet when added above answers question.

FIGURE 5.6: StackOverflow dataset after preprocessing

	Sentence	Label
6	sorry think missed some discussion mailing list. implementation test breakiterator attached.	-1
7	sorry rushing commit forgot add author during commit.	-1
8	direct link mail item did not work.	0
9	thanks raising these documentation issues! just committed fix.	1

FIGURE 5.7: Jira dataset after preprocessing

Dataset	No. of classes	Average Length	Maximum Length	Dataset size
StackOverflow	3	135	838	4423
Jira	3	95	3206	2571

TABLE 5.1: Characteristics of datasets after preprocessing

Neural networks accept only vectors as inputs and so the samples in the dataset are converted to vectors by performing an embedding-lookup in to find the embedding weight vector in the loaded word embedding models. The creation of the word embedding models are detailed in the following section.

5.3 Word-Embedding

Our study compares the effect on the performance of the deep learning classifiers when SE-specific and generic word embedding are used in the embedding layer of the classifiers. These embedding models are detailed in the following sections.

5.3.1 Google News word embedding

The pre-trained Google News word embedding is trained on 100 billion google news headlines. The model was developed by following a configuration of 300 vector dimension resulting in a vocabulary size of 3 million words [49]. The figure 5.8 depicts the word cloud of the top 700 words in the Google word embedding model.

FIGURE 5.8: Word Cloud of Google News word embedding model

5.3.2 Software-specific word embedding

To create the software-specific (SO) word embedding, two million StackOverflow posts with answers were mined. The Gensim library was used to create the Word2Vec model, and Continuous Bag of words (CBOW) architecture was followed. The word embedding were created by following the class diagram in fig 5.9. The posts were preprocessed as specified in section 5.2, and the tokenized sentences obtained were passed as a list and was input to the Word2Vec model to build the vocabulary. The model was then trained for 3 epochs with a vector dimension of 100 and window size of 5 (words that occurred less than 5 times were ignored) as shown in fig 5.10. The vector dimension of 100 was chosen, since the dataset is small and contains limited vocabulary. Hence the lower dimension is adequate for capturing the important features. The resulting Word2Vec model comprises of a vocabulary of 220,232 keywords. The figure 5.11 depicts the word cloud of the top 700 words in the SO word embedding model.

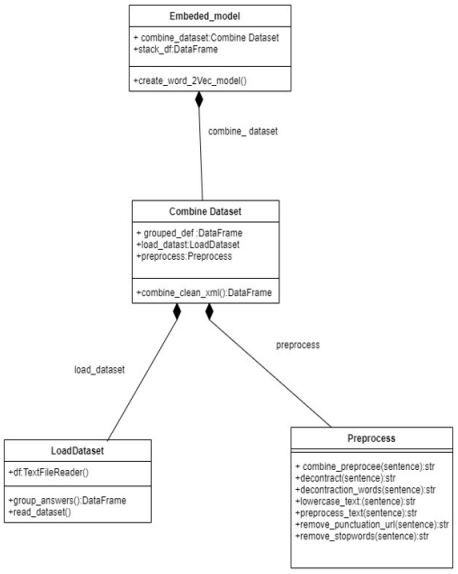


FIGURE 5.9: Class diagram for creating software-specific embeds

```
#word2vec parameters
W2V_SIZE = 100
W2V_WINDOW = 5
W2V_EPOCH = 3
W2V_MIN_COUNT = 5

# Collect the corpus for training word embeddings
#used stackOverflow posts that contains questions and answers

corpus = []
for text in np.array(stack_df.post_corpus):
    split_text = text.split()
    corpus.append(split_text)

#initializing the model
w2v_model = gensim.models.word2vec.Word2Vec(vector_size=W2V_SIZE,
                                            window=W2V_WINDOW,
                                            min_count=W2V_MIN_COUNT,
                                            workers=5)

#builing a vocabulary from the specified corpus

w2v_model.build_vocab(documents)

#training the word2vec model

w2v_model.train(documents, total_examples=len(documents), epochs=W2V_EPOCH)

#saving the word2vec model

w2v_model.wv.save_word2vec_format('w2v_SO_100.bin', binary=True)
```

FIGURE 5.10: Code snippet of Word2Vec parameters

FIGURE 5.11: Word Cloud of SO word embedding model

5.4 Deep Learning Classification Model

This section details the set of transformations applied on the input sentence in order for the sentiment classifier to identify the sentiment.

1. First, we preprocess the samples as described in 5.2 and then proceed to calculate the average sentence length of the samples in the datasets as a reference to the maximum number of inputs to the network [79]. To ensure the classification network receives equal length inputs, for the samples whose length is lesser than the average sentence length, the sample is padded until it reaches the desired length. Whereas for the samples that exceed the average sentence length, the input of the exceeding words are truncated. Thus, the length of the samples is normalized. For example, the sentence "I have committed this. Thanks Aaron!" is preprocessed and transformed to "committed. thanks aaron! ". This sample belongs to the Jira dataset and is padded to average length of sentences which is 95.

2. Since the deep-learning classifier only accepts vectors as input, we use the embedding layer to convert the input text to dense vectors. The embedding layer contains the semantic representation of text. We load the pre-trained word embedding models that contains the list of all words present in its vocabulary followed by the vector values for each of the words.

3. The input sentence is then integer encoded and padded or truncated to a uniform length using the Tokenizer package of Tensorflow as shown in fig 5.12, 5.13. Additionally, we create embedding matrices for the words present in the input sentence, by using the integer representation of unique words present in the sample and by performing embedding look-up to find the embedding weight vector from the loaded pre-trained word embedding [79, 12]. Moreover, if there exists a word that has no corresponding vector in the word embedding model, a zero vector is allotted with the dimension of the word embedding model. The embedding matrices are in dimensions (1x95x100) and (1x95x300) for SO and GN embeddings respectively. Figures 5.14, 5.15 illustrate the embedding matrices obtained for the above specified input sentence.

```
[[7, 1, 799]]
```

FIGURE 5.12: Integer encoded values for input sample

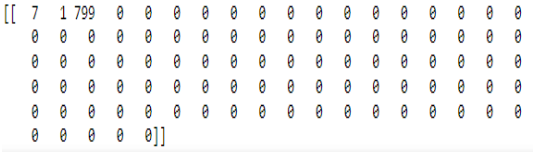


FIGURE 5.13: Padded input sample

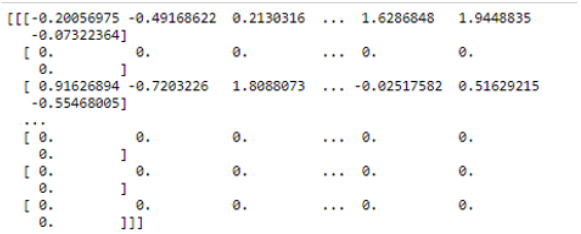


FIGURE 5.14: Embedding matrix from SO embeddings (1x95x100)

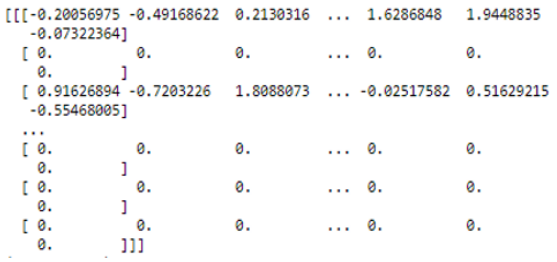


FIGURE 5.15: Embedding matrix from SO embeddings (1x95x300)

4. The sentiment classifiers are developed using CNN, LSTM and BiLSTM algorithms. In accordance with the previous studies in [79, 59, 77, 51, 7, 35] the authors observed significant higher performance when stacked layers of deep learning algorithms such as CNN, LSTM and BiLSTM were used to build their sentiment classifiers. This motivates us to build a sentiment classifier by employing more than 1-layer of the deep-learning algorithm. Thus, we use a 2-layer stacked CNN/LSTM/BILSTM to extract rich contextual features of text. These layers are stacked sequentially such that the embedding matrix of input sequence is passed onto the stacked layers of CNN/LSTM/BiLSTM and it outputs hidden vectors.

These hidden vectors passed onto a hidden dense layer in the case of LSTM/BILSTM classifiers and max-pooling, flatten layers for CNN classifier to further deepen the network. The final dense layer uses softmax activation function and outputs the sentiment of the sentence in form of probability distribution. The figure 5.16 depicts the sentiment of the input sentence, the neutral sentiment has the highest probability. Thus, sentiment of the given input sentence is neutral.

```
[[0.14477354 0.6900443  0.16518226]]
```

FIGURE 5.16: Output from dense layer

The architecture of the developed deep-learning classifiers are further explained in the following sections

5.4.1 Stacked CNN model

Convolution Neural Networks work by recognizing patterns, thereby identifying the correlated words present in the text. In this model, 2 layers of 1 dimensional CNN is used. Figure 5.17 depicts the configuration of the model, in which the output from the embedding layer is convolved with a convolution layer of 16 filters and kernel of size 3 followed by a second CNN layer of 16 filters and 3 kernels, a max-pooling layer, flatten layer and dense layer of 3 neurons to output the sentiment. Additionally, L2-regularization also known as weight decay of value 0.01 is added in the CNN layers to reduce overfitting [64]. The output from the second CNN layer is fed into the max-pooling layer to capture the fine-grained features from different parts of the text [41]. Moreover, dropouts of 0.4 and 0.2 are used after the max-pooling layer. Flatten layer is employed to flatten the dimensions as shown in figure 5.18. Categorical Cross Entropy is used as the loss function to find the optimal weights of the neural network [31, 27]. The model is trained by choosing a batch size of 64 to reduce the computational cost. Furthermore, the model employs SGD optimizer to manipulate the weights and learning rate to minimize the losses [27].

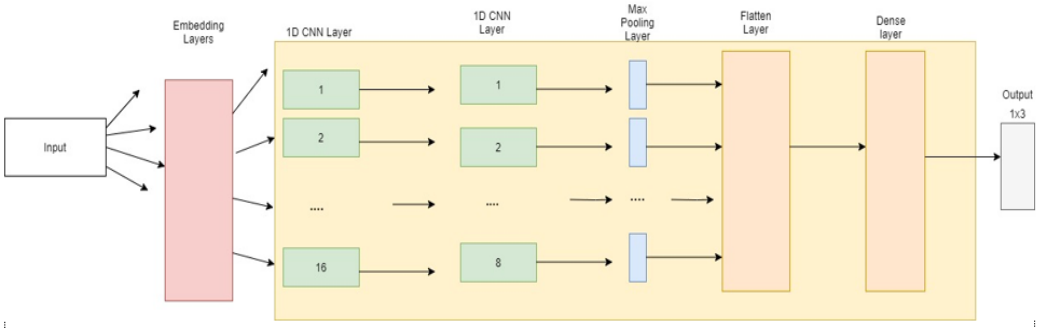


FIGURE 5.17: Structure of stacked layer CNN classifier

```
"""Creates a CNN model
   Args:
       vocab_length (int): size of the embedding model vocabulary
    Returns:
        CNN model of specified parameters
"""


def create_model(vocab_length):

    model = Sequential()
    embedding_layer = Embedding(vocab_length, embedding_dim, embeddings_initializer = Constant(embedding_matrix),
                                input_length=max_length, trainable = False)
    model.add(embedding_layer)
    model.add(Conv1D(16, 3, padding='same', activation='relu', kernel_regularizer=regularizers.l2(l=0.01)))
    model.add(Conv1D(8, 3, padding='same', activation='relu', kernel_regularizer=regularizers.l2(l=0.01)))
    model.add(MaxPooling1D())
    model.add(Dropout(0.4))
    model.add(Flatten())
    model.add(Dropout(0.2))
    model.add(Dense(3, activation='softmax'))
    opt = opt = tf.keras.optimizers.SGD(learning_rate=0.01)
    model.compile(loss='categorical_crossentropy',optimizer= opt ,metrics= ['accuracy'])
    model.summary()
    return model
```

FIGURE 5.18: Code Snippet of stacked CNN classifier

5.4.2 Stacked LSTM model

This model comprises of an embedding layer, two consecutive LSTM layers by a dense layer of 8 neurons and a dense layer of 3 neurons to output the classified sentiment label as depicted in the figure 5.19. The embedding layer is initialized with random weights and learnt the embedding for all words present in the training set. In this configuration, double LSTM layers are built comprising of 16 neurons each and is found that the model performed well without overfitting and underfitting. Additionally, the model employs a variant of SGD optimizer known as root mean squared propagation (RMSprop). RMSprop uses the exponentially weighted averages of the gradients to normalize the gradient itself and hence is considered as a prominent option for RNN-based deep learning models [71, 31]. Moreover, the model employs Categorical Cross entropy as the loss function since this is a multi-class classification task. Batch size of 64 was chosen as per the recommendation of Hameed et al. in [30] to build a robust model.

Overfitting of the neural network model is prevented by incorporating a dropout and recurrent dropout of 0.3 in the LSTM layer of the Model. Furthermore, L2-regularization of 0.001 is added in LSTM layer to prevent overfitting [64]. Figure 5.20 details the code snippet for the stacked 2-layer LSTM classifier.

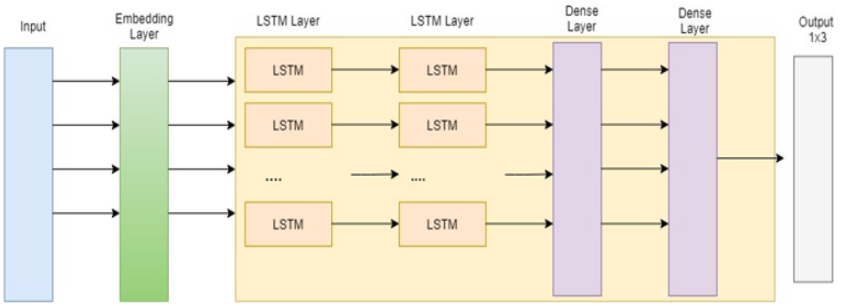


FIGURE 5.19: Structure of stacked layer LSTM classifier

```
#method creates the stacked 2-layer LSTM model

def create_model(vocab_length):

    model = Sequential()
    embedding_layer = Embedding(vocab_length, embedding_dim, embeddings_initializer = Constant(embedding_matrix),
                                input_length=max_length, trainable = False)
    model.add(embedding_layer)
    model.add(LSTM(16, dropout=0.3, recurrent_dropout = 0.3,
                   kernel_regularizer=regularizers.l2(0.001), return_sequences = True))
    model.add(LSTM(16, dropout=0.3, recurrent_dropout = 0.3,
                   kernel_regularizer=regularizers.l2(0.001)))
    model.add(Dense(8, activation = 'relu'))

    model.add(Dense(3, activation='softmax'))
    opt = keras.optimizers.RMSprop(learning_rate=0.0001)
    model.compile(loss='categorical_crossentropy',optimizer= opt ,metrics= ['accuracy'])
    model.summary()
    return model
```

FIGURE 5.20: Code Snippet of stacked LSTM classifier

5.4.3 Stacked BiLSTM model

The stacked BiLSTM model comprises of an embedding layer, two consecutive BiLSTM layers of 16 neurons, followed by a dense layer of 8 neurons and a final dense layer comprising of 3 neurons to output the sentiment label of the passed input sentence as shown in the figure 5.21. Dropout and a recurrent dropout of 0.1 is incorporated in both the BiLSTM layers to prevent overfitting. Additionally, L2-regularizer of value 0.001 is also used to prevent overfitting [30]. Furthermore, RMSprop and Categorical

cross entropy is used as the optimizer and loss function respectively. Figure 5.22 depicts the code snippet that is used to build the stacked BiLSTM classifier.

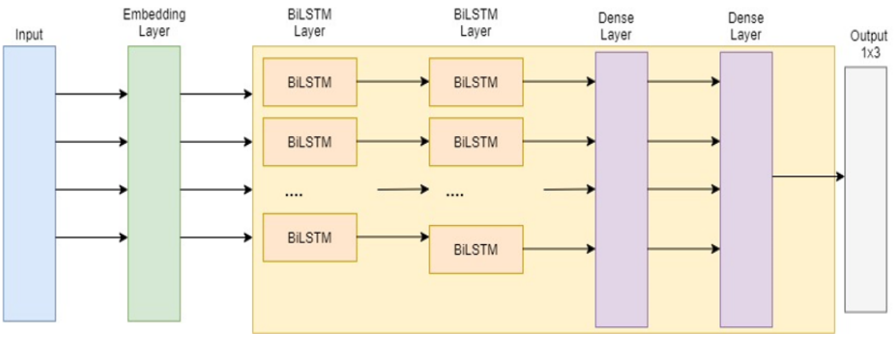


FIGURE 5.21: Structure of stacked layer BiLSTM classifier

```
"""Creates a BiLSTM model
   Args:
       vocab_length (int): size of the embedding model vocabulary
    Returns:
        BiLSTM model of specified parameters
"""

def create_model(num_word):
    model2 = Sequential()
    embedding_layer = Embedding(num_word, embedding_dim, embeddings_initializer = Constant(embedding_matrix), input_length=max_le
    model2.add(embedding_layer)
    model2.add(Bidirectional(LSTM(16,dropout=0.3, recurrent_dropout = 0.1, kernel_regularizer=regularizers.l2(0.001), return_sequ
    model2.add(Bidirectional(LSTM(16,dropout=0.3, recurrent_dropout = 0.1, kernel_regularizer=regularizers.l2(0.001))))

    model2.add(Dense(8, activation = 'relu'))
    model2.add(Dense(3, activation='softmax'))
    opt = keras.optimizers.RMSprop(learning_rate=0.0001)
    model2.compile(loss='categorical_crossentropy',optimizer= opt ,metrics= ['accuracy'])
    model2.summary()
    return model2
```

FIGURE 5.22: Code Snippet of stacked BiLSTM classifier

5.5 Results

All experiments are run with stratified 5-fold stratified cross validation technique, to obtain subpar results. The performance of various models are reported in terms of precision, recall and F-measure. The overall performance of the model is calculated by employing macro-averaging as the aggregated technique. In macro-averaging the precision and recall calculated independently for each class and is then averaged. Hence all classes are weighted equally [11].

5.5.1 StackOverflow Dataset

The results for the classifiers when trained on the StackOverflow dataset using stratified 5-fold are detailed below.

5.5.1.1 Stacked CNN classifier

The performance of the stacked CNN classifier over 5 folds are tabulated in the tables 5.2, 5.3 when Google News and StackOverflow word embedding are employed. Figures 5.23, 5.24 depict the structure of the CNN. The SO and Google news embedding have an output dimension of 100 and 300 respectively and the maximum length is set as the average length of the sentences for the StackOverflow dataset. The results of the embedding layer is in a matrix of dimension 1 x 135 x 100 and 1 x 135 x 300 for SO and Google news embedding respectively. These matrices are then passed to the CNN layer, max-pooling, flatten and then the dense layer to output the sentiment. The results are obtained by training the SO embedding based classifier for 25 epochs and the Google news embedding based classifier for 70 epochs. The classifier trained using SO embedding obtained an overall f-measure of 0.72 and the classifier trained using Google news embedding obtained an overall f-score of 0.75.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 135, 100)          977400
_________________________________________________________________
conv1d (Conv1D)              (None, 135, 16)           4816
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 135, 8)            392
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 67, 8)             0
_________________________________________________________________
dropout (Dropout)            (None, 67, 8)             0
_________________________________________________________________
flatten (Flatten)            (None, 536)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 536)               0
_________________________________________________________________
dense (Dense)                (None, 3)                 1611
=================================================================
Total params: 984,219
Trainable params: 6,819
Non-trainable params: 977,400
_________________________________________________________________
```

FIGURE 5.23: Structure of double layered CNN with SO embedding trained on StackOverflow Dataset

	Overall			Precision			Recall			F1-score		
Fold	P	R	F	Neg	Neu	Pos	Neg	Neu	Pos	Neg	Neu	Pos
0	0.75	0.71	0.71	0.74	0.65	0.87	0.43	0.85	0.86	0.54	0.73	0.86
1	0.72	0.70	0.71	0.64	0.64	0.87	0.52	0.76	0.83	0.57	0.70	0.85
2	0.71	0.71	0.71	0.64	0.68	0.82	0.54	0.71	0.87	0.59	0.69	0.84
3	0.74	0.74	0.74	0.65	0.72	0.86	0.65	0.73	0.85	0.65	0.73	0.85
4	0.71	0.71	0.71	0.58	0.67	0.89	0.61	0.72	0.79	0.59	0.69	0.84
AVG	0.73	0.71	0.72	0.65	0.67	0.86	0.55	0.75	0.84	0.59	0.71	0.85
STDEV				0.06	0.03	0.02	0.08	0.06	0.03	0.04	0.02	0.008

TABLE 5.2: Results employing SO embedding for StackOverflow Dataset

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 135, 300)          2933100
_________________________________________________________________
conv1d (Conv1D)              (None, 135, 16)           14416
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 135, 8)            392
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 67, 8)             0
_________________________________________________________________
dropout (Dropout)            (None, 67, 8)             0
_________________________________________________________________
flatten (Flatten)            (None, 536)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 536)               0
_________________________________________________________________
dense (Dense)                (None, 3)                 1611
=================================================================
Total params: 2,949,519
Trainable params: 16,419
Non-trainable params: 2,933,100
_________________________________________________________________
```

FIGURE 5.24: Structure of double layered CNN with Google News embedding trained on StackOverflow Dataset

	Overall			Precision			Recall			F1-score		
Fold	P	R	F	Neg	Neu	Pos	Neg	Neu	Pos	Neg	Neu	Pos
0	0.76	0.76	0.76	0.67	0.73	0.89	0.74	0.73	0.82	0.70	0.73	0.85
1	0.73	0.72	0.73	0.66	0.66	0.88	0.56	0.56	0.84	0.61	0.71	0.86
2	0.77	0.73	0.74	0.80	0.65	0.86	0.53	0.83	0.82	0.64	0.73	0.84
3	0.75	0.73	0.74	0.70	0.68	0.86	0.57	0.78	0.85	0.63	0.73	0.86
4	0.81	0.79	0.79	0.83	0.73	0.87	0.62	0.83	0.91	0.71	0.78	0.89
AVG	0.76	0.75	0.75	0.73	0.69	0.87	0.60	0.79	0.83	0.66	0.74	0.86
STD.DEV				0.07	0.04	0.01	0.08	0.04	0.03	0.04	0.03	0.02

TABLE 5.3: CNN Results employing Google news embedding for StackOverflow Dataset

5.5.1.2 Stacked LSTM classifier

The tables 5.4, 5.5 tabulates the performance of the stacked LSTM classifier for SO embedding and Google news embedding. The structure of the classifier is depicted in figure 5.25, 5.26. The output from the embedding layer is fed into two consecutive LSTM layers, followed by a dense layer of 8 neurons and the final dense layer to output the resulting sentiment. The results are obtained by training the classifiers for 80 epochs. The classifier trained using SO embedding obtained an overall f-score of 0.74 and the classifier trained using Google news embedding obtained an overall f-score of 0.81.

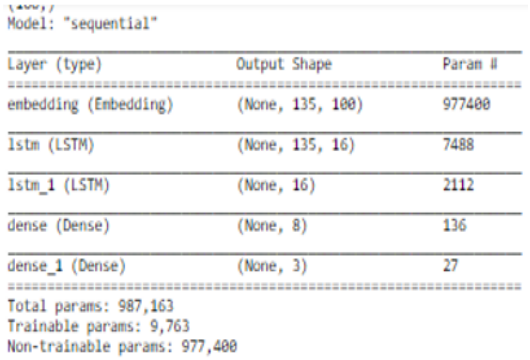


FIGURE 5.25: Structure of LSTM with SO embedding trained on StackOverflow Dataset

	Overall			Precision			Recall			F1-score		
Fold	P	R	F	Neg	Neu	Pos	Neg	Neu	Pos	Neg	Neu	Pos
0	0.76	0.75	0.75	0.69	0.71	0.87	0.67	0.77	0.81	0.68	0.74	0.84
1	0.76	0.75	0.76	0.67	0.72	0.89	0.63	0.79	0.84	0.65	0.75	0.87
2	0.73	0.73	0.73	0.62	0.69	0.89	0.63	0.74	0.81	0.62	0.72	0.85
3	0.75	0.73	0.74	0.67	0.68	0.90	0.63	0.78	0.78	0.65	0.73	0.83
4	0.75	0.74	0.74	0.65	0.71	0.89	0.65	0.77	0.80	0.65	0.74	0.84
AVG	0.75	0.74	0.74	0.66	0.70	0.89	0.64	0.77	0.81	0.65	0.74	0.85
STD.DEV				0.03	0.02	0.01	0.02	0.02	0.02	0.02	0.01	0.01

TABLE 5.4: LSTM Results employing SO embedding for StackOverflow Dataset

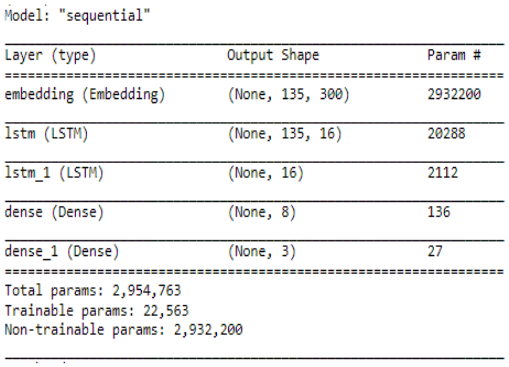


FIGURE 5.26: Structure of LSTM with Google News embedding trained on StackOverflow Dataset

	Overall			Precision			Recall			F1-score		
Fold	P	R	F	Neg	Neu	Pos	Neg	Neu	Pos	Neg	Neu	Pos
0	0.82	0.81	0.81	0.76	0.77	0.92	0.70	0.82	0.91	0.73	0.79	0.91
1	0.84	0.83	0.84	0.77	0.78	0.96	0.74	0.85	0.89	0.76	0.81	0.92
2	0.81	0.79	0.80	0.81	0.73	0.89	0.69	0.83	0.86	0.74	0.78	0.87
3	0.81	0.81	0.82	0.74	0.78	0.92	0.77	0.79	0.88	0.76	0.79	0.90
4	0.79	0.79	0.79	0.72	0.73	0.91	0.74	0.77	0.85	0.74	0.75	0.88
AVG	0.81	0.81	0.81	0.76	0.76	0.92	0.73	0.81	0.87	0.75	0.78	0.89
STD.DEV				0.03	0.02	0.02	0.03	0.03	0.02	0.01	0.02	0.02

TABLE 5.5: LSTM Results employing Google News embeddings for StackOverflow Dataset

5.5.1.3 Stacked BiLSTM classifier

The tables 5.6, 5.7 tabulates the results of the stacked BiLSTM classifier trained on the Stackoverflow Dataset using the SO and Google news word embeddings over 5 folds. The results are obtained by training the model for SO-based classifier for 100 epochs and the Google news-based classifier for 140 and on a batch size of 64. As observed from the table, the SO-embedding based classifier obtains an overall f-score of 0.78 and the Google news embedding based classifier obtains an overall f-score of 0.84. Figure 5.27, 5.28 depict the structure of BiLSTM classifiers employing SO and Google News word embeddings.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 135, 100)          977700
_________________________________________________________________
bidirectional_2 (Bidirection (None, 135, 32)           14976
_________________________________________________________________
bidirectional_3 (Bidirection (None, 32)                6272
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 264
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 27
=================================================================
Total params: 999,239
Trainable params: 21,539
Non-trainable params: 977,700
_________________________________________________________________
```

FIGURE 5.27: Structure of BiLSTM with SO embedding trained on StackOverflow Dataset

	Overall			Precision			Recall			F1-score		
Fold	P	R	F	Neg	Neu	Pos	Neg	Neu	Pos	Neg	Neu	Pos
0	0.80	0.80	0.80	0.72	0.75	0.93	0.74	0.80	0.85	0.73	0.78	0.89
1	0.76	0.75	0.75	0.69	0.71	0.87	0.66	0.76	0.84	0.67	0.73	0.85
2	0.79	0.78	0.78	0.68	0.77	0.91	0.74	0.77	0.83	0.71	0.77	0.87
3	0.77	0.79	0.76	0.67	0.73	0.90	0.71	0.86	0.81	0.69	0.74	0.85
4	0.80	0.80	0.80	0.73	0.77	0.91	0.70	0.80	0.90	0.71	0.78	0.91
AVG	0.78	0.78	0.78	0.71	0.74	0.90	0.73	0.80	0.85	0.72	0.76	0.87
STD.DEV				0.020	0.03	0.02	0.02	0.04	0.03	0.006	0.02	0.03

TABLE 5.6: BiLSTM Results employing SO embeddings for StackOverflow Dataset

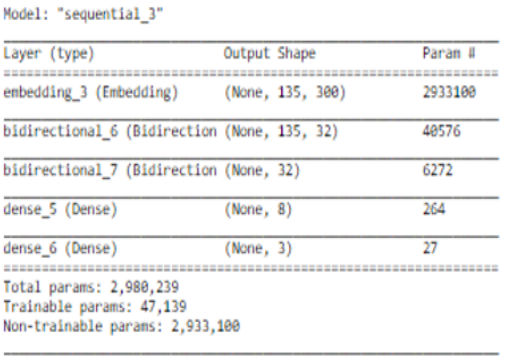


FIGURE 5.28: Structure of BiLSTM with Google News embedding trained on StackOverflow Dataset

	Overall			Precision			Recall			F1-score		
Fold	P	R	F	Neg	Neu	Pos	Neg	Neu	Pos	Neg	Neu	Pos
0	0.84	0.81	0.82	0.81	0.79	0.92	0.79	0.84	0.81	0.80	0.81	0.85
1	0.84	0.84	0.84	0.80	0.82	0.91	0.81	0.83	0.87	0.80	0.83	0.89
2	0.85	0.85	0.84	0.81	0.82	0.92	0.82	0.83	0.90	0.81	0.83	0.90
3	0.85	0.85	0.85	0.83	0.82	0.90	0.82	0.85	0.88	0.83	0.83	0.90
4	0.84	0.84	0.84	0.81	0.78	0.93	0.76	0.88	0.88	0.79	0.83	0.91
AVG	0.84	0.84	0.84	0.81	0.81	0.90	0.80	0.84	0.87	0.81	0.83	0.88
STD.DEV				0.01	0.02	0.01	0.03	0.01	0.03	0.02	0.01	0.02

TABLE 5.7: BiLSTM Results employing Google News embeddings for StackOverflow Dataset

5.5.2 Jira Dataset

The results for the classifiers when trained on the Jira dataset using stratified 5-fold are detailed below.

5.5.2.1 Stacked CNN classifier

The results in the tables 5.8, 5.9 tabulates the performance of the stacked CNN classifier on the Jira dataset, when SO and GN embeddings are employed. The structure of the classifiers is shown in figure 5.29, 5.30 The results are obtained by training the model for 80 epochs for the GN embedding based classifier and 25 epochs for the SO embeddings. It is observed that the GN embeddings-based classifier obtains an overall f-score of 0.82 and the SO embedding based classifier obtains an overall f-score of 0.78.

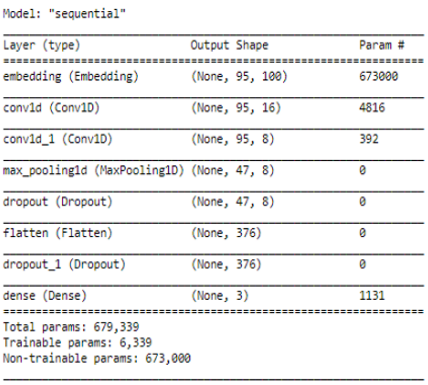


FIGURE 5.29: Structure of stacked CNN with SO embedding trained on Jira Dataset

	Overall			Precision			Recall			F1-score		
Fold	P	R	F	Neg	Neu	Pos	Neg	Neu	Pos	Neg	Neu	Pos
0	0.84	0.81	0.81	0.80	0.81	0.91	0.82	0.74	0.87	0.81	0.73	0.89
1	0.83	0.80	0.80	0.80	0.74	0.94	0.78	0.76	0.85	0.77	0.73	0.88
2	0.79	0.76	0.76	0.75	0.73	0.89	0.68	0.71	0.87	0.71	0.68	0.88
3	0.78	0.76	0.76	0.75	0.71	0.89	0.79	0.58	0.87	0.75	0.63	0.88
4	0.82	0.73	0.80	0.77	0.81	0.89	0.73	0.75	0.70	0.77	0.72	0.78
AVG	0.81	0.76	0.78	0.77	0.76	0.90	0.75	0.71	0.83	0.76	0.73	0.86
STD.DEV				0.02	0.04	0.02	0.05	0.07	0.08	0.03	0.04	0.005

TABLE 5.8: Results of CNN employing SO embeddings for Jira Dataset

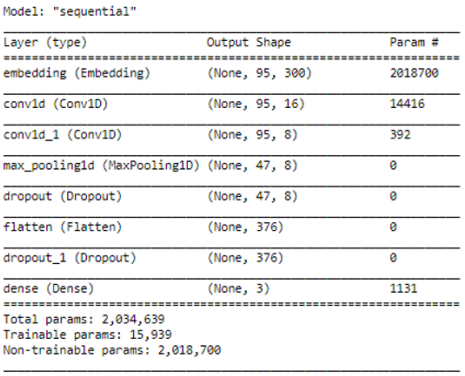


FIGURE 5.30: Structure of stacked CNN with Google news embedding trained on Jira Dataset

	Overall			Precision			Recall			F1-score		
Fold	P	R	F	Neg	Neu	Pos	Neg	Neu	Pos	Neg	Neu	Pos
0	0.84	0.84	0.84	0.80	0.79	0.93	0.85	0.79	0.89	0.82	0.79	0.91
1	0.81	0.81	0.81	0.81	0.72	0.90	0.84	0.73	0.87	0.82	0.72	0.88
2	0.82	0.82	0.82	0.82	0.76	0.89	0.82	0.75	0.88	0.82	0.76	0.89
3	0.82	0.83	0.84	0.88	0.75	0.82	0.81	0.79	0.9	0.84	0.77	0.91
4	0.80	0.81	0.80	0.74	0.72	0.94	0.84	0.68	0.9	0.79	0.70	0.92
AVG	0.82	0.82	0.82	0.81	0.75	0.90	0.83	0.75	0.89	0.82	0.75	0.90
STDEV				0.05	0.03	0.05	0.02	0.05	0.01	0.02	0.04	0.02

TABLE 5.9: Results of CNN employing Google News embeddings for Jira Dataset

5.5.2.2 Stacked LSTM classifier

The results in table 5.10, 5.11 tabulates the performance of the stacked LSTM classifier on the Jira dataset, when SO and GN word embeddings are employed. The structure of the classifiers is shown in figure 5.31, 5.32. The results are obtained by training the SO embedding based model for 60 epochs and the GN embedding based model was trained for 100 epochs. It is observed that the google news embeddings-based classifier obtains an overall f-score of 0.82 and the SO embedding based classifier obtains an overall f-score of 0.78.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 95, 100)           672900
_________________________________________________________________
lstm (LSTM)                  (None, 95, 16)            7488
_________________________________________________________________
lstm_1 (LSTM)                (None, 16)                2112
_________________________________________________________________
dense (Dense)                (None, 8)                 136
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 27
=================================================================
Total params: 682,663
Trainable params: 9,763
Non-trainable params: 672,900
```

FIGURE 5.31: Structure of LSTM with SO embedding trained on Jira Dataset

	Overall			Precision			Recall			F1-score		
Fold	P	R	F	Neg	Neu	Pos	Neg	Neu	Pos	Neg	Neu	Pos
0	0.76	0.76	0.76	0.71	0.65	0.92	0.69	0.74	0.85	0.70	0.69	0.89
1	0.78	0.79	0.78	0.72	0.72	0.91	0.73	0.76	0.88	0.72	0.74	0.89
2	0.77	0.78	0.78	0.76	0.66	0.90	0.73	0.74	0.86	0.75	0.70	0.88
3	0.76	0.76	0.76	0.70	0.66	0.92	0.71	0.69	0.88	0.70	0.68	0.91
4	0.79	0.80	0.80	0.76	0.69	0.94	0.76	0.78	0.86	0.76	0.73	0.90
AVG	0.78	0.78	0.78	0.73	0.68	0.92	0.72	0.75	0.87	0.73	0.71	0.89
STD.DEV				0.03	0.02	0.01	0.02	0.03	0.01	0.02	0.02	0.01

TABLE 5.10: Results of LSTM employing SO embeddings for Jira Dataset

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 95, 300)           2019000
_________________________________________________________________
lstm (LSTM)                  (None, 95, 16)            20288
_________________________________________________________________
lstm_1 (LSTM)                (None, 16)                2112
_________________________________________________________________
dense (Dense)                (None, 8)                 136
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 27
=================================================================
Total params: 2,041,563
Trainable params: 22,563
Non-trainable params: 2,019,000
_________________________________________________________________
```

FIGURE 5.32: Structure of LSTM with Google News embedding trained on Jira Dataset

	Overall			Precision			Recall			F1-score		
Fold	P	R	F	Neg	Neu	Pos	Neg	Neu	Pos	Neg	Neu	Pos
0	0.82	0.82	0.82	0.77	0.76	0.93	0.79	0.78	0.89	0.78	0.77	0.91
1	0.85	0.85	0.85	0.84	0.79	0.92	0.84	0.81	0.91	0.84	0.80	0.91
2	0.82	0.81	0.81	0.82	0.74	0.89	0.76	0.74	0.93	0.79	0.74	0.91
3	0.80	0.80	0.80	0.80	0.70	0.89	0.78	0.71	0.90	0.79	0.70	0.90
4	0.80	0.80	0.80	0.81	0.71	0.88	0.80	0.70	0.9	0.80	0.71	0.89
AVG	0.82	0.82	0.82	0.81	0.74	0.90	0.79	0.75	0.91	0.80	0.74	0.90
STD.DEV				0.02	0.04	0.02	0.03	0.05	0.01	0.02	0.04	0.01

TABLE 5.11: Results of LSTM employing Google News embeddings for Jira Dataset

5.5.2.3 Stacked BiLSTM classifier

The results from the tables 5.12, 5.13 tabulate the performance of the stacked BiLSTM classifier on the Jira dataset when SO and GN embeddings are used. The structure of the BiLSTM classifier is as shown in figures 5.33, 5.34. The results are obtained by training the SO embedding classifier for 60 epochs and the GN embedding based classifier for 90 epochs. It is observed that the google news embeddings-based classifier obtains an overall f-measure 0.86 and the SO embedding based classifier obtains an overall f-measure of 0.84.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 95, 100)           673000
_________________________________________________________________
bidirectional (Bidirectional (None, 95, 32)            14976
_________________________________________________________________
bidirectional_1 (Bidirection (None, 32)                6272
_________________________________________________________________
dense (Dense)                (None, 8)                 264
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 27
=================================================================
Total params: 694,539
Trainable params: 21,539
Non-trainable params: 673,000
_________________________________________________________________
```

FIGURE 5.33: Structure of BiLSTM with SO embedding trained on Jira Dataset

	Overall			Precision			Recall			F1-score		
Fold	P	R	F	Neg	Neu	Pos	Neg	Neu	Pos	Neg	Neu	Pos
0	0.84	0.84	0.84	0.83	0.76	0.94	0.78	0.82	0.92	0.80	0.79	0.93
1	0.86	0.86	0.86	0.84	0.78	0.95	0.84	0.84	0.90	0.84	0.81	0.92
2	0.79	0.81	0.79	0.75	0.70	0.92	0.76	0.79	0.85	0.75	0.75	0.88
3	0.85	0.85	0.85	0.85	0.79	0.92	0.84	0.80	0.91	0.85	0.79	0.92
4	0.83	0.83	0.83	0.81	0.75	0.93	0.80	0.82	0.88	0.80	0.78	0.9
AVG	0.84	0.84	0.84	0.82	0.76	0.93	0.80	0.81	0.89	0.81	0.78	0.91
STD.DEV				0.04	0.03	0.01	0.03	0.02	0.03	0.04	0.02	0.02

TABLE 5.12: Results of BiLSTM employing SO embeddings for Jira Dataset

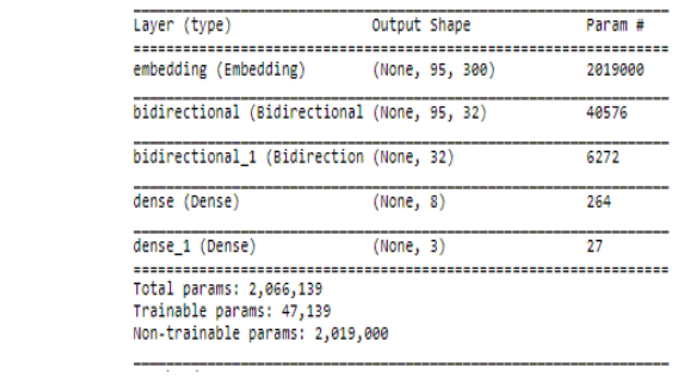


FIGURE 5.34: Structure of BiLSTM with GN embedding trained on Jira Dataset

	Overall			Precision			Recall			F1-score		
Fold	P	R	F	Neg	Neu	Pos	Neg	Neu	Pos	Neg	Neu	Pos
0	0.85	0.87	0.86	0.88	0.76	0.92	0.84	0.88	0.90	0.86	0.81	0.91
1	0.87	0.87	0.87	0.88	0.79	0.94	0.86	0.84	0.92	0.87	0.81	0.93
2	0.86	0.86	0.86	0.86	0.78	0.94	0.84	0.84	0.91	0.85	0.81	0.92
3	0.86	0.87	0.86	0.85	0.80	0.94	0.86	0.84	0.91	0.85	0.82	0.92
4	0.84	0.84	0.84	0.81	0.80	0.91	0.84	0.76	0.92	0.82	0.78	0.91
AVG	0.86	0.86	0.86	0.86	0.79	0.93	0.85	0.83	0.91	0.85	0.81	0.92
STD.DEV				0.03	0.02	0.01	0.01	0.04	0.01	0.02	0.02	0.01

TABLE 5.13: Results of BiLSTM employing Google news embeddings for Jira Dataset

5.6 CLI Application

Command line interface application as shown in fig 5.35 is developed to enable the user to train the deep-learning classifiers. The user inputs the filename containing the software sentiment dataset, word embedding model, deep-learning classifier, and number of epochs to train the classifier. The classification report for every fold is output, in addition to the overall performance of the classifier at the end of all 5 folds. The class diagram is depicted in figure 5.36.

FIGURE 5.35: CLI application used for training the models

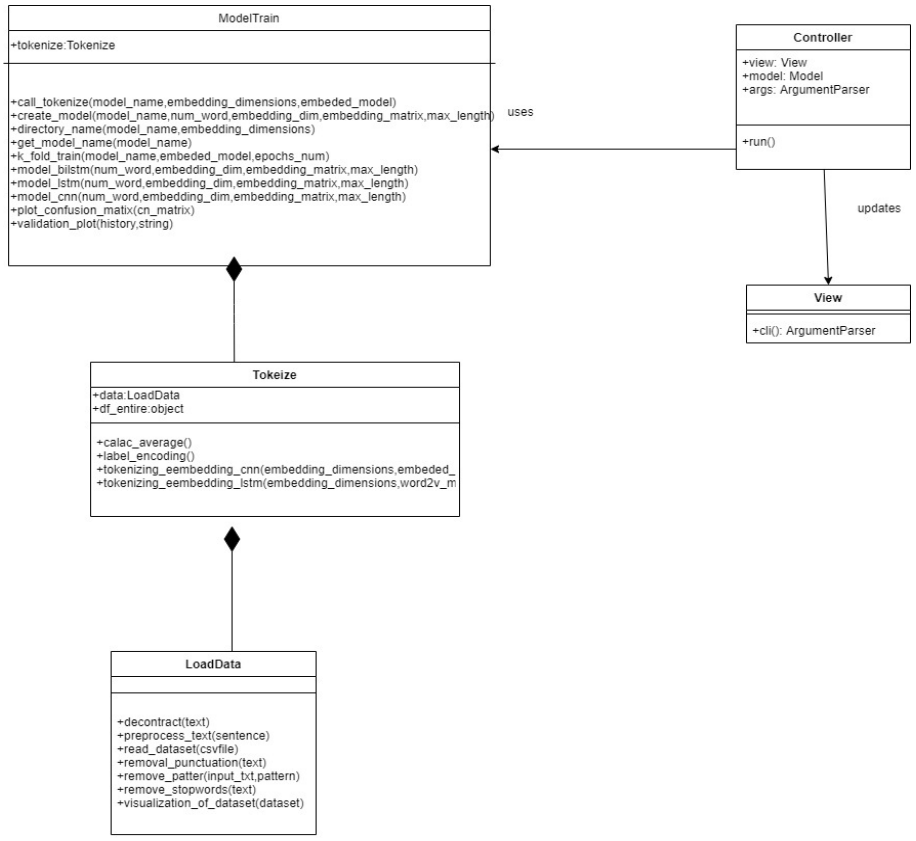


FIGURE 5.36: MVC Class diagram for the CLI application

5.7 GUI Application

A simple GUI application is developed following the class diagram depicted in fig 5.37, in which the user can input the sentence and sentiment of the sentence is identified accordingly. The Tkinter library of python is used to develop the GUI and it consists of a text box to input the sentence, radio buttons to select the word embedding models and the classifier models. The input sentence is preprocessed as specified in section 5.2 and then passed onto the classifier to predict the sentiment. The figure 5.38 depicts the class diagram.

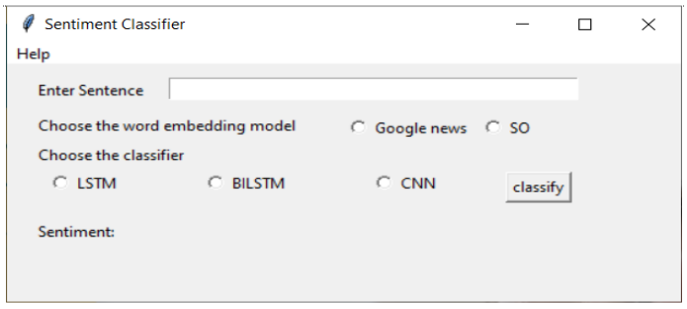


FIGURE 5.37: GUI of the Sentiment Classifier

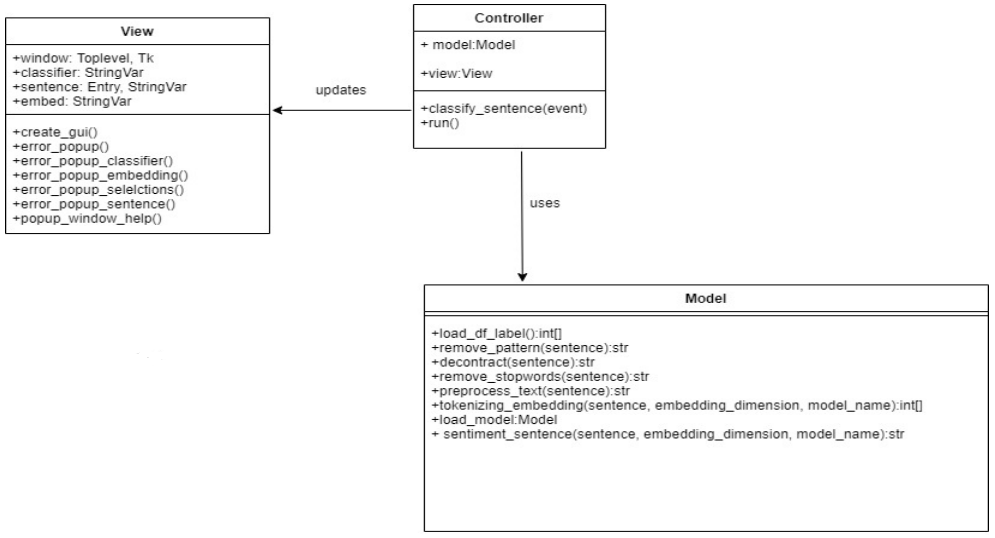


FIGURE 5.38: MVC Class diagram for the GUI application

5.8 Implementation challenges

Although the project plan specified in appendix 8.2 was followed closely, there were few challenges that was faced during implementation.

1. Dealing with large dataset

   To create the SO word-embeddings, the StackOverflow datadump containing 2 million posts were used. Since the neural network model of Word2Vec extracts features automatically, time taken was quite high to create the SO word embeddings.

2. Dealing with Many Models

   Another challenge faced was training the many models, and finding the best among them. It took time training each model and comparing them. Furthermore, stratified 10-fold cross validation did not provide the best results and so 5-fold stratified cross validation was used. Overall, 12 models were trained and tested, some of which were time-consuming and took more than 2 hours to train and test.

5.9 Baseline Methods

This section details the baseline methods that are used to compare with the developed classifiers in this study in 6.5.

5.9.1 SentiStrength

SentiStrength [70] is a state-of-art lexicon-based sentiment classifier that classifies the sentiment of short informal text. SentiStrength employs the use of a lexicon dictionary that comprises of positive and negative list of words, sentiment word strength list and a booster word list. To identify the sentiment of the text, SentiStrength computes the algebraic sum and reports the overall trinary score, such that the positive sentiment sentence is denoted as (+1), negative sentiment sentence is denoted as (-1) and a neutral sentiment sentence is represented by (0) [69, 57].

5.9.2 SentiStrength-SE

SentiStrength-SE [35] is an extension of SentiStrength adapted to the SE domain. The lexicon dictionary of SentiStrength-SE was built on top of SentiStrength by manually adjusting the sentiment scores of the words in order to capture the semantics of SE-domain [37, 57]. These adjustments were done by running SentiStrength on the Group-1 subset of the Jira dataset to analyze the reasons for the misclassification and thereby customize the lexicon dictionary accordingly. To ensure a fair comparison, we did not include Group-1 issue comments in the Jira dataset.

5.9.3 Senti4SD

Senti4SD developed by Novielli et al. in [13] is a machine learning based sentiment analyser that leverages lexicons features based on the sentiment word list of, keyword features such as unigrams and bi-grams and semantic features. The classifier employs SVM algorithm to classify the sentences.

Chapter 6

Evaluation

This chapter presents and discusses the findings of the experiments conducted on the deep learning classifiers using different datasets and when SE-specific word-embeddings and generic word-embeddings are employed.

6.1 Comparison of deep-learning models

The deep-learning models are trained and simultaneously tested by employing stratified k-fold cross validation technique on the StackOverflow and the Jira datasets. The values for the evaluation metrics are calculated by taking the mean over 5 runs. The overall performance is calculated by using the macro-averaging technique. As mentioned earlier, macro-averaging was selected since all the sentiment classes will be weighted equally. Additionally, the best value for each of the metric is highlighted.

6.1.1 StackOverflow dataset

F-measure is the harmonic average of precision and recall. High f-measure indicates high precision and recall values. This helps us to compare the performance of different deep-learning classifiers developed, in the case of uneven class distribution in datasets. From figure 6.1, we observe that stacked-CNN has the lowest overall f-measure leading to its substandard performance. Additionally, on closer examination of results from table 6.1, it is observed, the stacked CNN secures the lowest recall for the negative and

lowest precision for the neutral classes. Moreover, the performance of stacked LSTM is comparable to that of stacked BiLSTM, however the latter outperforms the former in terms of overall precision, recall and f-measure. Furthermore, the stacked BiLSTM obtains higher f-measure values for negative and neutral classes.

Class	Metric	Stacked- LSTM	Stacked -BiLSTM	Stacked -CNN
Positive	Precision	0.89	0.90	0.86
	Recall	0.81	0.85	0.84
	F-score	0.85	0.87	0.85
Negative	Precision	0.66	0.71	0.65
	Recall	0.64	0.73	0.55
	F-score	0.65	0.72	0.59
Neutral	Precision	0.70	0.74	0.67
	Recall	0.77	0.80	0.74
	F-score	0.74	0.76	0.71
Overall	Precision	0.75	0.78	0.73
	Recall	0.74	0.78	0.71
	F-score	0.74	0.78	0.72

TABLE 6.1: Performance of SO-embeddings based classifiers for StackOverflow dataset

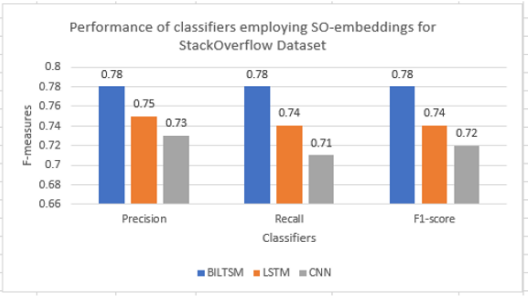


FIGURE 6.1: Performance of classifiers using SO word embedding

To illustrate the impact of domain customization on the performance of classifiers, this study records precision, recall and f-measure values when SO and GN embeddings are used. 6.1, 6.2 reports the results achieved using SO and GN embeddings respectively. Comparing the results from tables 6.1, 6.2, it is observed that all the classifiers employing GN word embeddings have significantly higher overall f-measures when compared to the classifiers using SO word embeddings.

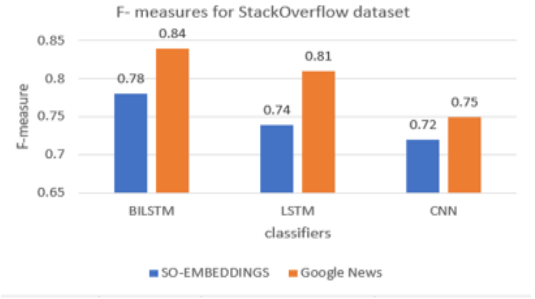


FIGURE 6.2: Comparisons of f-measures depending on embedding models

Class	Metric	Stacked- LSTM	Stacked -BiLSTM	Stacked -CNN
Positive	Precision	0.92	0.90	0.87
	Recall	0.87	0.87	0.83
	F-score	0.89	0.88	0.86
Negative	Precision	0.76	0.81	0.73
	Recall	0.73	0.80	0.60
	F-score	0.75	0.81	0.66
Neutral	Precision	0.76	0.81	0.69
	Recall	0.81	0.84	0.79
	F-score	0.78	0.83	0.74
Overall	Precision	0.81	0.84	0.76
	Recall	0.81	0.84	0.75
	F-score	0.81	0.84	0.75

TABLE 6.2: GN-embeddings based classifiers for StackOverflow dataset

Based on the precision, recall and f-measure values for individual classes and overall, we conclude the most balanced classifier for this dataset is the stacked BiLSTM classifier using GN embeddings which obtains an overall f-measure of 0.84.

6.2 Jira dataset

From figure 6.3 we observe that the stacked BiLSTM classifier obtains the highest overall f-measure. Additionally, from table 6.3, it is observed the stacked BiLSTM classifier obtains the highest precision, recall and f-measure for positive, negative, and neutral classes. Moreover, we see that the stacked LSTM and stacked CNN obtain equal overall f-measure values.

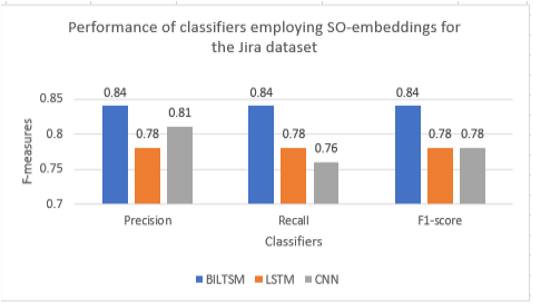


FIGURE 6.3: Performance of classifiers using SO-embeddings for jira dataset

Class	Metric	Stacked- LSTM	Stacked -BiLSTM	Stacked -CNN
Positive	Precision	0.92	0.93	0.90
	Recall	0.87	0.89	0.83
	F-score	0.89	0.91	0.86
Negative	Precision	0.73	0.82	0.77
	Recall	0.72	0.80	0.75
	F-score	0.73	0.81	0.76
Neutral	Precision	0.68	0.76	0.76
	Recall	0.75	0.81	0.71
	F-score	0.71	0.78	0.73
Overall	Precision	0.78	0.84	0.81
	Recall	0.78	0.84	0.76
	F-score	0.78	0.84	0.78

TABLE 6.3: Performance of SO-embeddings based classifiers for Jira dataset

Furthermore, when comparing the performance of classifiers employing SO and GN embeddings, it is observed that all classifiers employing GN embeddings outperform the SO embedding based classifiers for the Jira dataset as seen from tables 6.3, 6.4 and fig 6.4. Thus, the best performing classifier for the Jira dataset is the stacked BiLSTM employing GN embeddings, which secures an overall f-measure of 0.86.

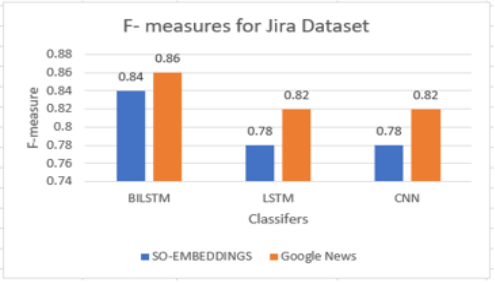


FIGURE 6.4: Comparisons of f-measures for jira dataset

Class	Metric	Stacked- LSTM	Stacked -BiLSTM	Stacked -CNN
Positive	Precision	0.90	0.93	0.90
	Recall	0.91	0.91	0.89
	F-score	0.90	0.92	0.90
Negative	Precision	0.81	0.86	0.81
	Recall	0.79	0.85	0.83
	F-score	0.80	0.85	0.82
Neutral	Precision	0.74	0.79	0.75
	Recall	0.75	0.83	0.75
	F-score	0.74	0.81	0.75
Overall	Precision	0.82	0.86	0.82
	Recall	0.82	0.86	0.82
	F-score	0.82	0.86	0.82

TABLE 6.4: Performance of GN-embeddings based classifiers for Jira dataset

6.3 Summary

When comparing the performance of deep-learning classifiers on StackOverflow and Jira datasets, it is observed that the stacked BiLSTM model obtains the highest f-measure values when compared to stacked CNN and stacked LSTM, irrespective of the word embedding model used. The subpar results of the stacked BiLSTM classifier is due to the additional LSTM layer in the backward direction, thus stacking more than one BiLSTM layer enables the developed classifier to extract and learn more features when compared to the other two models. This is in line with the previous study conducted in [79], hence suggesting that the stacked BiLSTM classifiers are robust across both the datasets.

Additionally, the next best-performing model is observed to be the stacked LSTM classifier, that outperforms the stacked CNN in terms of precision, recall and f-measure for StackOverflow dataset and in terms of recall for the Jira dataset, this is because of the sequential structure of LSTM that can capture the semantic dependencies of text, in contrast to the hierarchical structure of CNN, thus leading to better performance of stacked LSTM in comparison to stacked CNN.

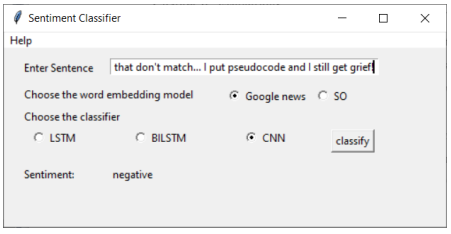
Consequently, when comparing the impact of domain customization on the performance of the classifiers, it is observed that for the StackOverflow and Jira datasets, all the developed classifiers employing generic word embeddings performed significantly better than the classifiers using software-specific SO embeddings. This is in accordance with the previous study in [12], that observed higher performance of classifiers using generic word-embedding. The difference in performance is attributed to the differing Word2Vec model parameters that were used in the creation of the SO and Google news embeddings. The Google news embeddings were created by training on 100 billion Google News posts, thereby obtaining a vocabulary of size 3 million words, whereas the SO embeddings were trained on 2 million posts obtaining a relatively smaller vocabulary of size 220,232 words.

Additionally, the StackOverflow and Jira datasets contains a greater number of samples that are broad and generic in context, leading to robust performance of the Google news embedding based classifiers. Thus, off-the-shelf GN word-embeddings are proven to be more generalizable due to the large corpus of text they are trained upon and can be used in the task of sentiment analysis to achieve maximal classification performance. Hence, the developed stacked BiLSTM classifier using GN embeddings is considered the best model when compared to stacked LSTM and stacked CNN models.

6.4 GUI application results

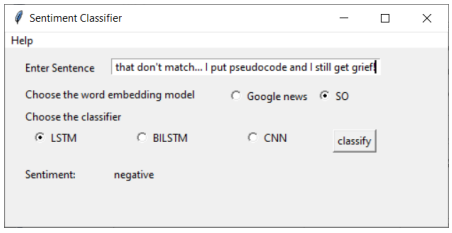
This section specifies the outputs obtained on the GUI application for sentiment classification. For the input sentence "I swear - I don't put pseudo code I get told off for having bad variable names and things that don't match... I put pseudocode and I still get grief!" is transorfmed as "swear do not put pseudo code get told off having bad variable names things do not match... put pseudocode still get grief!" after preprocessing. Subsequently the preprocessed sample is padded and embedding matrices are created and used as inputs to the deep-learning classifiers as specified in section 5.4. As observed from figures 6.5, 6.6, 6.7 all the classifiers incorporating SO and GN embeddings accurately identify the sentiment of the sentence as negative.

(A) SO embedding



(B) GN embedding

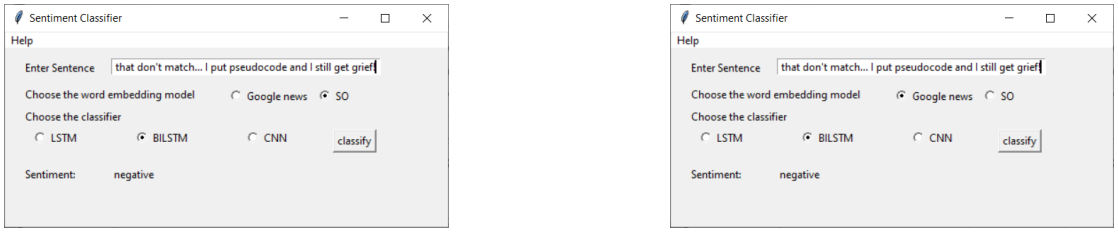
FIGURE 6.5: CNN classifiers



(A) SO embedding

(B) GN embedding

FIGURE 6.6: LSTM classifiers



(A) SO embedding (B) GN embedding

FIGURE 6.7: BiLSTM classifiers

6.5 Comparison of previous work

In this section, we compare the performance of the stacked BiLSTM classifier employing GN embeddings (stacked-BiLSTM-GN) to the existing baseline SA tools (SentiStrength [1], SentiStrength-SE[2] and Senti4SD[3]) for the StackOverflow and Jira datasets. To enable a fair comparison for each of the classifiers and the datasets, we employ the 5-fold stratified cross-validation technique and retrain the supervised classifier Senti4SD. The remaining classifiers SentiStrength and SentiStrength-SE are not retrained since they are lexicon-based classifiers and hence do not require retraining[57],[17].

6.5.1 StackOverflow dataset

On initial observation, we see that the stacked BiLSTM-GN classifier outperforms the baseline methods on the StackOverflow dataset. On evaluating the results thoroughly, we observe an improvement in the overall f-measure of the stacked-BiLSTM-GN classifier by 3% and 5% when compared to the lexicon SA tools, SentiStrength and SentiStrength-SE as seen from table 6.5, 6.6. The reason for the relatively smaller performance gap between the stacked BiLSTM-GN classifier and SentiStrength on the StackOverflow dataset is due to the usage of the SentiStrength tool in the creation of the StackOverflow dataset. Therefore, SentiStrength can accurately classify the sentiments of the sentences selected by itself. Moreover, when comparing the stacked BiLSTM-GN classifier to the ML-based classifier Senti4SD, we observed the stacked BiLSTM-GN classifier outperforms Senti4SD by only 1% in terms of f-measure as seen from table 6.7. On closer examination of the confusion matrix in 6.8, it is observed the stacked BiLSTM-GN classifer can address the challenge of negations and misclassifications of neutral samples as negative,

as the number of misclassifications of neutral samples is reduced from 14.7% to 10%for the StackOverflow dataset. As a result, we observe an increase in the f-measure for the neutral class from 0.8 to 0.83 and 0.8 to 0.81 for the negative class.

Class	Metric	SentiStrength	Stacked BiLSTM-GN	Improvement over the baseline
Positive	Precision	0.88	0.90	2%
	Recall	0.92	0.87	-5%
	F-score	0.90	0.88	-2%
Negative	Precision	0.67	0.81	14%
	Recall	0.93	0.80	-7%
	F-score	0.78	0.81	3%
Neutral	Precision	0.92	0.81	-11%
	Recall	0.63	0.84	21%
	F-score	0.75	0.83	8%
Overall	Precision	0.82	0.84	3%
	Recall	0.83	0.84	1%
	F-score	0.81	0.84	3%

Table 6.5: Comparison of stacked-BiLSTM-GN and SentiStrength

Class	Metric	SentiStrength-SE	Stacked BiLSTM-GN	Improvement over the baseline
Positive	Precision	0.90	0.90	0%
	Recall	0.82	0.87	5%
	F-score	0.86	0.88	2%
Negative	Precision	0.75	0.81	6%
	Recall	0.76	0.80	4%
	F-score	0.76	0.81	5%
Neutral	Precision	0.72	0.81	9%
	Recall	0.78	0.84	6%
	F-score	0.75	0.83	8%
Overall	Precision	0.79	0.84	6%
	Recall	0.79	0.84	5%
	F-score	0.79	0.84	5%

Table 6.6: Comparison of stacked-BiLSTM-GN and SentiStrength-SE

Class	Metric	Senti4SD	Stacked BiLSTM-GN	Improvement over the baseline
Positive	Precision	0.90	0.90	0%
	Recall	0.91	0.87	-4%
	F-score	0.90	0.88	-2%
Negative	Precision	0.78	0.81	3%
	Recall	0.84	0.80	-4%
	F-score	0.80	0.81	1%
Neutral	Precision	0.83	0.81	-2%
	Recall	0.77	0.84	7%
	F-score	0.80	0.83	3%
Overall	Precision	0.83	0.84	1%
	Recall	0.84	0.84	0%
	F-score	0.83	0.84	1%

Table 6.7: Comparison of stacked-BiLSTM-GN and Senti4SD

		Prediction					
		Senti4SD			BILSTM -GN		
		Negative	Neutral	Positive	Negative	Neutral	Positive
A	Negative	202(84.1%)	34(14.1%)	4(1.6%)	192(80%)	37(15.4%)	11(4.25%)
C	Neutral	50(14.7%)	261(77%)	28(8.3%)	35(10%)	286(84.4%)	19(5.6%)
T	Positive	7(2.2%)	20(6.5%)	279(91%)	10(3.2%)	30(9.8%)	266(87%)
U							
A							
L							

FIGURE 6.8: Confusion matrices of Senti4SD and stacked-BiLSTM-GN

6.5.2 Jira dataset

From table 6.8, 6.9 it is observed that the stacked BiLSTM-GN classifier outperforms lexicon classifiers SentiStrength and SentiStrength-SE in terms of overall f-measure by 12% and 4% respectively. Similarly, from table 6.10 the stacked BILSTM-GN classifier outperforms Senti4SD by 4% on the Jira dataset. Additionally, on examination of the confusion matrices from figure 6.9, it is observed the stacked BiLSTM-GN classifier successfully reduces the misclassifications of neutral samples as negative from 12.9,% to 7.8,%, thereby increasing the f-measure of the neutral class from 0.73 to 0.81 and from 0.81 to 0.86 for the negative class. The above-mentioned results demonstrate the superiority of the developed stacked BiLSTM-GN classifier in this thesis that successfully reduces the misclassifications of neutral samples as negative, when compared to the existing SA tools for the SE domain.

Class	Metric	SentiStrength	Stacked BiLSTM-GN	Improvement over the baseline
Positive	Precision	0.84	0.93	9%
	Recall	0.88	0.91	3%
	F-score	0.86	0.92	6%
Negative	Precision	0.77	0.86	9%
	Recall	0.70	0.85	15%
	F-score	0.73	0.86	13%
Neutral	Precision	0.61	0.79	18%
	Recall	0.63	0.83	21%
	F-score	0.62	0.81	19%
Overall	Precision	0.74	0.86	12%
	Recall	0.74	0.86	12%
	F-score	0.74	0.86	12%

TABLE 6.8: Comparison of stacked-BiLSTM-GN and SentiStrength for jira dataset

Class	Metric	SentiStrength-SE	Stacked BiLSTM-GN	Improvement over the baseline
Positive	Precision	0.93	0.93	0%
	Recall	0.91	0.91	0%
	F-score	0.91	0.92	1%
Negative	Precision	0.87	0.86	-1%
	Recall	0.73	0.85	8%
	F-score	0.79	0.86	7%
Neutral	Precision	0.71	0.79	8%
	Recall	0.84	0.83	0%
	F-score	0.77	0.81	4%
Overall	Precision	0.84	0.86	2%
	Recall	0.83	0.86	3%
	F-score	0.82	0.86	4%

TABLE 6.9: Comparison of stacked-BiLSTM-GN and SentiStrength SE for jira dataset

Class	Metric	Senti4SD	Stacked BiLSTM-GN	Improvement over the baseline
Positive	Precision	0.88	0.93	5%
	Recall	0.92	0.91	-1%
	F-score	0.90	0.92	2%
Negative	Precision	0.83	0.86	3%
	Recall	0.79	0.85	6%
	F-score	0.81	0.86	5%
Neutral	Precision	0.74	0.79	5%
	Recall	0.73	0.83	10%
	F-score	0.73	0.81	8%
Overall	Precision	0.81	0.86	5%
	Recall	0.82	0.86	4%
	F-score	0.81	0.86	5%

TABLE 6.10: Comparison of stacked-BiLSTM-GN and Senti4SD for jira dataset

		Prediction					
		Senti4SD			BILSTM -GN		
		Negative	Neutral	Positive	Negative	Neutral	Positive
ACTUAL	Negative	121(79.1%)	25(16.3%)	6(3.9%)	129(85%)	20(13.1%)	3(1.9%)
	Neutral	18(12.9%)	102(72.8%)	20(14.3%)	11(7.8%)	116(83%)	13(9.2%)
	Positive	7(3.2%)	11(5.9%)	203(91.8)	10(4.2%)	11(4.9%)	201(90.9%)

FIGURE 6.9: Confusion matrices of Senti4SD and stacked-BiLSTM-GN

Chapter 7

Conclusions

7.1 Conclusion

In this thesis, we present a deep learning model for SA of SE text. This work aims at finding the best deep-learning algorithm to identify the sentiment of SE text. We built sentiment classifiers by employing stacked layers of CNN, LSTM and BiLSTM algorithms. The results from this study, showed that the BiLSTM classifier produces better results when compared to other classifiers. Additionally, we studied the impact on the performance of classifiers when generic and software specific word embedding were used and found that the developed classifiers incorporating the general GN word embedding model performed better in identifying sentiments when compared to SO-embedding based classifiers. By incorporating GN word embedding in the stacked BiLSTM model, we found the above classifier achieve improved results and also address the challenge of misclassification of neutral sentiments as negative. Moreover, adopting the MVC design architecture for the development of GUI and CLI applications ensured quicker development process and enabled easier debugging, as this design patten allows us to clearly structure the project between different elements. Finally, all the requirements specified in chapter 3 in addition to the optional requirement have been achieved.

7.1.1 Limitations

In this thesis, we identify the limitations of the developed classifiers by analysing the misclassified samples and designating them to specific error classes.

1. Presence of Sentimental lexicons in neutral sentences

   The deep learning classifiers that incorporate Google News and StackOverflow word embedding have erroneously classified the sentence "HXT, a library which is used for parsing XML, is a very good example for the usage of arrows (have a look how often the word occurs in the module names of this package!). You shall have a look on the great tutorial: http://adit.io/posts/2012-04-14-working_with_HTML_in_haskell. htmlBut it is also good to have the arrow concept for functions. For example the following code just works, because is an instance of the arrow class (The operator is defined in Control.Arrow).Thanks to the arrow syntax you have also a great tool to write complex computations in Haskell (it works as well for functions, monads and XML filters in HXT)." as positive sentiment. The above-mentioned sample is of neutral sentiment, however due to the presence of the words ("good", "great") leads to its misclassification.

2. Implicit Sentiment present in sentences

   The classifiers fail to capture the implicit sentiment present in the sentence. For example, in the sentence "Lastly, I don't see any styling data. Does your project have some CSS defined somewhere" expresses a negative sentiment as the user is complaining about the absence of the styling data. However, the sentiment in the sentence is vague, such that the classifiers is unable to accurately identify it.

3. Presence of opposite sentiments in a sentence

   Among the misclassified samples, we observe that in the cases of sentences that contains both positive and negative sentiment words, for example in the sentence "I'm working on Java web application and using from and as a web server. When i build the project, build complete successfully but when i start , the files that created during build in folder, getting to deleted!!" conveys a positive sentiment at the beginning but later on ends up in a negative note. Subsequently, the classifiers misclassify this negative sample as a neutral one.

7.2 Future Work

As for future work, we intend on fine-tuning the developed classifiers by adding more layers and residual connections to enhance the classification performance. Additionally,

we plan to explore various transformer language models such as BERT, which can effectively interpret context and polysemous words, in contrast to word-embedding models such as Word2vec. Thus, achieves ground-breaking results in many of the NLP tasks. Additionally, we intend on conducting a comparison study to compare different deep-learning architectures and transformer models to enhance the classification performance for SA in the SE domain.

Chapter 8

Appendix A

8.1 Discussion on Professional, Legal, Ethical and Social Issues

This section details the professional, legal, ethical and social issues that are present in the implementation of the project and how these issues are addressed.

8.1.1 Professional Issues

The code written is in high standard and follows the British Computing Society Code of Conduct and practices[1]. The code is well commented to provide the reader with accurate information. Any algorithms taken from any prior work will be cited accordingly. The implementation of the project will be divided into three milestones- developing the deep-learning classifiers, evaluations and dissertation report writing. Additionally, OneDrive is used to store the source code after the evaluation of the thesis.

8.1.2 Legal Issues

The Data Protection Law and Intellectual Property Law is closely followed and applied.

[1] https://www.bcs.org/membership/become-a-member/bcs-code-of-conduct

8.1.3 Data Protection Law

The dataset used in this project is publicly available and was developed by Novielli et al. in [13] and Ortu et al. in [58]. The gold-standard dataset comprises of posts from StackOverflow that are manually annotated with emotion labels. The dataset is used for academic purposes and is deleted once the dissertation is complete. This project follows the technical measures of the organization to prevent data loss and unauthorized used.

8.1.4 Intellectual Property law

"Intellectual property law deals with the rules for securing and enforcing legal rights to inventions, designs, and artistic works" [28]. The study takes the consideration of copyrights, patents and design rights. Moreover, any information used in this study has been cited and referenced appropriately.

8.1.5 Ethical and Social Issues

This project does not have any ethical and social issues as it does not involve the use of any sensitive data.

8.2 Project Plan

8.3 Project Stakeholders

The following stakeholders and their requirements have been identified and tabulated 8.1 in the development of this project

Stakeholder	Requirement
Project Supervisor	Working Software
Myself	High-quality implementation, extensive research and thorough documentation

TABLE 8.1: Project Stakeholders

8.4 Project Tasks and Deliverables

The below table 8.2 tabulates the key tasks and its deliverables.

Task	Deliverable
Literature Review	Analysing previous work
Methodology adopted	Deep learning algorithm-based classifier to be built. The algorithms to be used is CNN, LSTM and BiLSTM
Develop a Standalone	Create a GUI and a executable version of the developed software
Documentation	Thorough documentation of the project
Submission	Submit the project well-before the deadline

TABLE 8.2: Project tasks and deliverables

8.5 Risk Analysis and Mitigations

Due to the nature of the thesis and schedule, quite a few risks that may occur during the course of project implementation are identified and tabulated as shown in table 2.

Risk	Likelihood	Impact	Mitigation
Loss of Source Code	High	High- Need to start from scratch	Push the code to Git repository regularly
Dataset insufficient for training	Medium	High- This can produce incorrect readings	Use another publicly available dataset
Insuffiecient computing resource	Medium	Medium- Likely to cause a delay in testing and development	Use the university resource
Student falling ill	High	High- Project plan might get delayed by a week.	Add flexibility to the schedule

Supervisor goes on leave	Medium	High- Delayed feedbacks	Use Microsoft Teams and emails
Implementation proves difficult	Medium	High- project will not be completed on time.	Arrange project hierarchically, such that the deliverables can be built on top of each other. Start with the easier ones first and difficult parts later. This will ensure that the initial aspects can be submitted. Develop backup plan for every deliverable. The back-up plan may include steps to direct the deliverable to be a research-based deliverable than an implementation deliverable.

TABLE 8.3: Risk and mitigations

8.6 Project plan

The Figures 8.1, 8.2 and illustrates the project plan for this project. The duration of the research report is from the beginning of January to the beginning of April. The implementation of the project started from the beginning of May. The project includes preprocessing the dataset, developing word-embeddings, building deep-learning sentiment classifiers and evaluation of the models developed with accuracy, precision, F1-score and recall. The development of the project is also documented and written in the report.

Title	Start date	End Date	Duration
Literature review	06/01/2021	06/03/2021	59
Research Reprot	07/03/2021	08/04/2021	32
Exam Period	10/04/2021	27/04/2021	17
Development	01/05/2021	15/08/2021	106
Processing the dataset	01/05/2021	11/05/2021	10
Generating SE specific and generic word embeddings	11/05/2021	26/05/2021	15
Building CNN model	11/05/2021	17/05/2021	6
Evaluating CNN model	18/05/2021	26/05/2021	8
Building LSTM model	11/06/2021	17/06/2021	6
Evaluating LSTM model	18/06/2021	25/06/2021	7
Building BiLSTM model	26/06/2021	03/07/2021	7
Evaluating BiLSTM model	04/07/2021	11/07/2021	7
Comaprison of models	12/07/2021	15/07/2021	3
Dissertation writing	15/07/2021	08/08/2021	24

FIGURE 8.1: Project plan Table

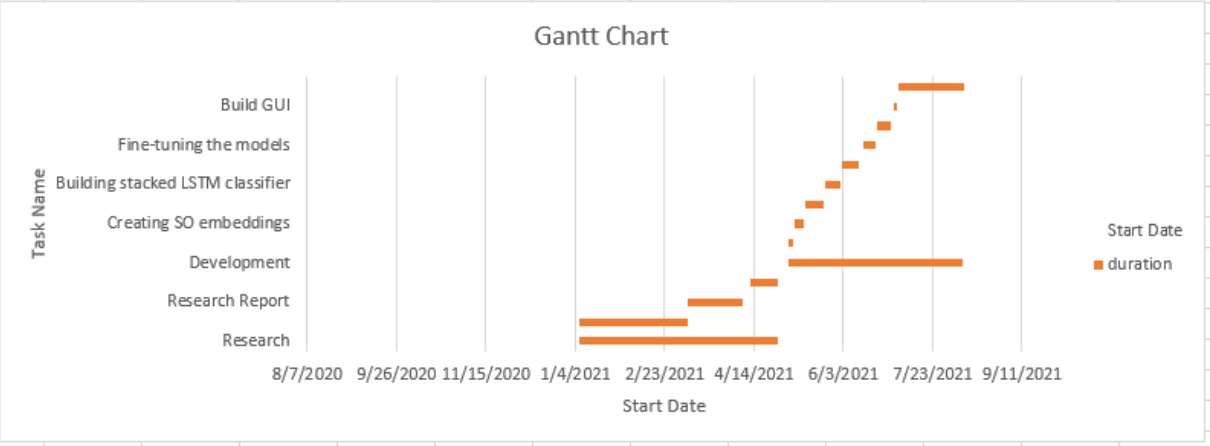


FIGURE 8.2: Gantt Chart

Bibliography

[1] (2010). SentiStrength. http://sentistrength.wlv.ac.uk/. [Online; accessed July 2021].

[2] (2017). SentiStrength-SE. http://laser.cs.uno.edu/Projects/Projects.html. [Online; accessed July 2021].

[3] (2018). Senti4SD. https://github.com/collab-uniba/Senti4SD. [Online; accessed July 2021].

[4] Ahmed, T., Bosu, A., Iqbal, A., and Rahimi, S. (2017). Senticr: A customized sentiment analysis tool for code review interactions. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 106–111.

[5] Ali, M., Son, D.-H., Kang, S.-H., and Nam, S.-R. (2017). An accurate ct saturation classification using a deep learning approach based on unsupervised feature extraction and supervised fine-tuning strategy. *Energies*, 10:1830.

[6] Alom, M. Z., Taha, T., Yakopcic, C., Westberg, S., Hasan, M., Esesn, B., Awwal, A., and Asari, V. (2018). The history began from alexnet: A comprehensive survey on deep learning approaches.

[7] Aslam, A., Qamar, U., Saqib, P., Ayesha, R., and Qadeer, A. (2020). A novel framework for sentiment analysis using deep learning. In *2020 22nd International Conference on Advanced Communication Technology (ICACT)*, pages 525–529.

[8] Baktha, K. and Tripathy, B. K. (2017). Investigation of recurrent neural networks in the field of sentiment analysis. In *2017 International Conference on Communication and Signal Processing (ICCSP)*, pages 2047–2050.

[9] Barnes, J., Klinger, R., and Schulte im Walde, S. (2017). Assessing state-of-the-art sentiment models on state-of-the-art sentiment datasets. In *Proceedings of the*

8th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis, pages 2–12, Copenhagen, Denmark. Association for Computational Linguistics.

[10] Bird, S., Klein, E., and Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st edition.

[11] Biswas, E., Karabulut, M. E., Pollock, L., and Vijay-Shanker, K. (2020). Achieving reliable sentiment analysis in the software engineering domain using bert. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 162–173.

[12] Biswas, E., Vijay-Shanker, K., and Pollock, L. (2019). Exploring word embedding techniques to improve sentiment analysis of software engineering texts. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 68–78.

[13] Calefato, F., Lanubile, F., Maiorano, F., and Novielli, N. (2018). Sentiment polarity detection for software development. pages 128–128.

[14] Calefato, F., Lanubile, F., and Novielli, N. (2017). Emotxt: A toolkit for emotion recognition from text. pages 79–80.

[15] Cambria, E., Das, D., Bandyopadhyay, S., and Feraco, A. (2017). *A Practical Guide to Sentiment Analysis*, volume 5.

[16] Chachra, A., Mehndiratta, P., and Gupta, M. (2017). Sentiment analysis of text using deep convolution neural networks. In *2017 Tenth International Conference on Contemporary Computing (IC3)*, pages 1–6.

[17] Chen, Z., Cao, Y., Lu, X., Mei, Q., and Liu, X. (2019). Sentimoji: An emoji-powered learning approach for sentiment analysis in software engineering. In *Proceedings of the 2019 ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE'19*.

[18] Dang, N. C., Moreno-García, M. N., and De la Prieta, F. (2020). Sentiment analysis based on deep learning: A comparative study. *Electronics*, 9(3).

[19] Dashtipour, K., Gogate, M., Adeel, A., Larijani, H., and Hussain, A. (2021). Sentiment analysis of persian movie reviews using deep learning. *Entropy*, 23.

[20] Deng, Z.-H., Luo, K.-H., and Yu, H.-L. (2014). A study of supervised term weighting scheme for sentiment analysis. *Expert Systems with Applications*, 41(7):3506–3513.

[21] Ding, J., Sun, H., Wang, X., and Liu, X. (2018). Entity-level sentiment analysis of issue comments.

[22] Efstathiou, V., Chatzilenas, C., and Spinellis, D. (2018). Word embeddings for the software engineering domain. In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, pages 38–41.

[23] Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2):179–211.

[24] Filho, P., Avanço, L., Pardo, T., and Nunes, M. (2014). $\text{Nilc}_{u}sp$: $Animprovedhybridsystemforsentimentanalysisintwittermessages.$ $pages428$ $--432.$

[25] Garcia, D., Zanetti, M. S., and Schweitzer, F. (2013). The role of emotions in contributors activity: A case study on the gentoo community. In *2013 International Conference on Cloud and Green Computing*, pages 410–417.

[26] Goldberg, Y. (2017). Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10:1–309.

[27] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[28] Gron, A. (2017). *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 1st edition.

[29] Guzman, E., Azócar, D., and Li, Y. (2014). Sentiment analysis of commit comments in github: An empirical study. *11th Working Conference on Mining Software Repositories, MSR 2014 - Proceedings*.

[30] Hameed, Z. and Garcia-Zapirain, B. (2020). Sentiment classification using a single-layered bilstm model. *IEEE Access*, 8:73992–74001.

[31] Hameed, Z. and Zapirain, B. (2020). Sentiment classification using a single-layered bilstm model. *IEEE Access*, 8:73992–74001.

[32] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9:1735–80.

[33] Islam, M. and Zibran, M. (2018). Deva: sensing emotions in the valence arousal space in software engineering text. pages 1536–1543.

[34] Islam, M. R. and Zibran, M. F. (2017). A comparison of dictionary building methods for sentiment analysis in software engineering text. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 478–479.

[35] Islam, M. R. and Zibran, M. F. (2017). Leveraging automated sentiment analysis in software engineering. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 203–214.

[36] Islam, M. R. and Zibran, M. F. (2018). A comparison of software engineering domain specific sentiment analysis tools. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 487–491.

[37] Islam, M. R. and Zibran, M. F. (2018). Sentistrength-se: Exploiting domain specificity for improved sentiment analysis in software engineering text. *Journal of Systems and Software*, 145:125–146.

[38] Jongeling, R., Datta, S., and Serebrenik, A. (2015). Choosing your weapons: On sentiment analysis tools for software engineering research. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 531–535.

[39] Kalaivani, K. S., Uma, S., and Kanimozhiselvi, C. S. (2021). Comparison of deep learning approaches for sentiment classification. In *2021 6th International Conference on Inventive Computation Technologies (ICICT)*, pages 1043–1047.

[40] Kansara, D. and Sawant, V. (2020). *Comparison of Traditional Machine Learning and Deep Learning Approaches for Sentiment Analysis*, pages 365–377.

[41] Kim, Y. (2014). Convolutional neural networks for sentence classification. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*.

[42] Kohavi, R. (2001). A study of cross-validation and bootstrap for accuracy estimation and model selection. 14.

[43] Kurniasari, L. and Setyanto, A. (2020). Sentiment analysis using recurrent neural network. *Journal of Physics: Conference Series*, 1471:012018.

[44] Lin, B., Zampetti, F., Bavota, G., Di Penta, M., Lanza, M., and Oliveto, R. (2018). Sentiment analysis for software engineering: How far can we go? In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 94–104.

[45] Liu, B. (2012). *Sentiment Analysis and Opinion Mining*. Morgan amp; Claypool Publishers.

[46] Liu, B. and Zhang, L. (2012). *A Survey of Opinion Mining and Sentiment Analysis*, pages 415–463. Springer US, Boston, MA.

[47] Mabrouk, A., Redondo, R. P. D., and Kayed, M. (2020). Deep learning-based sentiment classification: A comparative survey. *IEEE Access*, 8:85616–85638.

[48] Medhat, W., Hassan, A., and Korashy, H. (2014). Sentiment analysis algorithms and applications: A survey. *Ain Shams Engineering Journal*, 5(4):1093–1113.

[49] Mikolov, T., Corrado, G., Chen, K., and Dean, J. (2013). Efficient estimation of word representations in vector space. pages 1–12.

[50] Monika, R., Deivalakshmi, S., and Janet, B. (2019). Sentiment analysis of us airlines tweets using lstm/rnn. In *2019 IEEE 9th International Conference on Advanced Computing (IACC)*, pages 92–95.

[51] Monika, R., Deivalakshmi, S., and Janet, B. (2019). Sentiment analysis of us airlines tweets using lstm/rnn. In *2019 IEEE 9th International Conference on Advanced Computing (IACC)*, pages 92–95.

[52] Mukwazvure, A. and Supreethi, K. P. (2015). A hybrid approach to sentiment analysis of news comments. In *2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions)*, pages 1–6.

[53] Murgia, A., Ortu, M., Tourani, P., Adams, B., and Demeyer, S. (2018). An exploratory qualitative and quantitative analysis of emotions in issue report comments of open source systems. *Empirical Software Engineering*, 44:1.

[54] Murgia, A., Tourani, P., Adams, B., and Ortu, M. (2014). Do developers feel emotions? an exploratory analysis of emotions in software artifacts. *11th Working Conference on Mining Software Repositories, MSR 2014 - Proceedings*.

[55] Narayanan, V., Arora, I., and Bhatia, A. (2013). Fast and accurate sentiment classification using an enhanced naive bayes model.

[56] Novielli, N., Calefato, F., and Lanubile, F. (2018a). A gold standard for emotion annotation in stack overflow. pages 14–17.

[57] Novielli, N., Girardi, D., and Lanubile, F. (2018b). A benchmark study on sentiment analysis for software engineering research.

[58] Ortu, M., Murgia, A., Destefanis, G., Tourani, P., Tonelli, R., Marchesi, M., and Adams, B. (2016). The emotional side of software developers in jira.

[59] Ouyang, X., Zhou, P., Li, C. H., and Liu, L. (2015). Sentiment analysis using convolutional neural network. In *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, pages 2359–2364.

[60] Pennington, J., Socher, R., and Manning, C. (2014). GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

[61] Qaisar, S. M. (2020). Sentiment analysis of imdb movie reviews using long short-term memory. In *2020 2nd International Conference on Computer and Information Sciences (ICCIS)*, pages 1–4.

[62] Rahman, M. M., Roy, C., and Kievanloo, I. (2015). Recommending insightful comments for source code using crowdsourced knowledge.

[63] Rao, G., Huang, W., Feng, Z., and Cong, Q. (2018). Lstm with sentence representations for document-level sentiment classification. *Neurocomputing*, 308:49–57.

[64] Reed, R. D. and Marks, R. J. (1998). *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. MIT Press, Cambridge, MA, USA.

[65] Rumelhart, D., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.

[66] Shen, J., Baysal, O., and Shafiq, M. O. (2019). Evaluating the performance of machine learning sentiment analysis algorithms in software engineering. In *2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pages 1023–1030.

[67] Shen, J., Baysal, O., and Shafiq, M. O. (2019). Evaluating the performance of machine learning sentiment analysis algorithms in software engineering. In *2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pages 1023–1030.

[68] Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C., Ng, A., and Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. *EMNLP*, 1631:1631–1642.

[69] Thelwall, M., Buckley, K., and Paltoglou, G. (2012). Sentiment strength detection for the social web. *Journal of the American Society for Information Science and Technology*, 63:163–173.

[70] Thelwall, M., Buckley, K., Paltoglou, G., Cai, D., and Kappas, A. (2010). Sentiment strength detection in short informal text. *Journal of the American Society for Information Science and Technology*, 61:2544–2558.

[71] Tieleman, T. and Hinston, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. pages 26–31.

[72] Ting, K. M. (2010). *Confusion Matrix*, pages 209–209. Springer US, Boston, MA.

[73] Wang, Q., Sun, L., and Chen, Z. (2019). Sentiment analysis of reviews based on deep learning model. In *2019 IEEE/ACIS 18th International Conference on Computer and Information Science (ICIS)*, pages 258–261.

[74] Witten, I. H. and Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[75] Zhang, K., Song, W., Liu, L., Zhao, X., and Du, C. (2019). Bidirectional long short-term memory for sentiment analysis of chinese product reviews. In *2019 IEEE 9th International Conference on Electronics Information and Emergency Communication (ICEIEC)*, pages 1–4.

[76] Zhang, L. and Liu, B. (2017). *Sentiment Analysis and Opinion Mining*, pages 1152–1161. Springer US, Boston, MA.

[77] Zhang, X., Zhao, J., and Lecun, Y. (2015). Character-level convolutional networks for text classification. In *NIPS'15: Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, pages 649–657.

[78] Zhang, Y. and Hou, D. (2013). Extracting problematic api features from forum discussions. In *2013 21st International Conference on Program Comprehension (ICPC)*, pages 142–151.

[79] Zhou, J., Lu, Y., Dai, H.-N., Wang, H., and Xiao, H. (2019). Sentiment analysis of chinese microblog based on stacked bidirectional lstm. *IEEE Access*, 7:38856–38866.