Learning-based Approaches for Addressing Challenges in Sentiment Analysis

THESIS

Submitted in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

RUPAL BHARGAVA

Under the Supervision of

Dr. Yashvardhan Sharma

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE
PILANI (RAJASTHAN) INDIA

February 2019

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
PILANI, RAJASTHAN

CERTIFICATE

This is to certify that the thesis entitled "**Learning-based Approaches for Addressing Challenges in Sentiment Analysis**", submitted by **Rupal Bhargava** ID No. **2013PHXF0009P** for the award of Ph.D degree of the Institute and embodies original work done by her under my supervision.

Signature of the supervisor	:	____________________
Name	:	**DR. YASHVARDHAN SHARMA**
Designation	:	Associate Professor, Department of Computer Science BITS Pilani, Pilani Campus Pilani
Date	:	__________

Dedicated To
My Parents

Acknowledgements

I would like to express my sincere gratitude to my advisor Dr. Yashvardhan Sharma for the continuous support during my PhD course and related research assistance. His patience, motivation, and immense knowledge helped me to overcome the difficulties during the course. His guidance helped me every time during the research and drafting this thesis.
Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Poonam Goyal, Prof. Aruna M for their insightful comments and encouragement, and also for their deep-diving queries which incented me to widen my research from various perspectives. My sincere thanks also goes to Prof. J.P Mishra and Prof. Sudeept Mohan for their continuous support and encouragement. Without their precious support it would not have been possible to conduct this research. I would like to express my gratitude to Prof. Navneet Goyal and Prof. Rahul Banerjee for their constant guidance, help, support and well-wishes. I thank my fellow research scholars for the stimulating discussions and for all the fun we have had in the past four years.
Last but not the least, I would like to thank my parents, my brother, my husband,inlaws and other family members for supporting me throughout the thesis drafting and my life in general.

Abstract

Sentiments and their implications play a vital role in our life. People appraise the opinions of others before reaching a conclusion. With the advent of forums, blogs and social networking sites, there has been a considerable increase in sharing of opinions and sentiments on social media. Netizens exhibit divergent views on a particular subject and many times end up agreeing to the disagreement. A substantial amount of research has been done in automated text analysis, viewpoint analysis and opinion extraction. The increase in usage of social media in multilingual countries, such as India, has posed challenges of accommodating multilingual text in addition to monolingual text and its processing. The immense increase in information overload of viewpoints on the internet, calls for efficient methods to extract the useful information with the view to facilitate decision making. Opinion mining or viewpoint excavation has been treated as a classification problem which stratifies documents or products as good/bad or positive/negative. People may have ambivalent opinions about the topic or product.

Information overload has lead to increased focus on the need for creating an alternative approach for representation and selection, of text and multimedia contents. An efficient framework is needed for representing the essential parts of the text so that user can decide whether he/she should read the whole text or not. For such cases, text summarization is a better suited for representing an opinion.

Although the automated text summarization is focused on text inputs, but multimedia information, video, images, recorded tracks, online information or hypertexts can also be considered as inputs.

The thesis focuses on the problem of text summarization which considers vital

parts of the document, extracts utilitarian information, and provides a broad overview of opinions. This saves the user from going through many documents to reach a conclusion. Work done also focuses on aspects/features present in the document and associates opinions with each of them.
Social media content has been used for extracting sentiment, but all the content cannot be trusted because it can be faked. Distinguishing a fake post from a genuine post can be challenging at times. The thesis attempts to automate the problem of spam detection using hybrid sequential models.
Sarcasm inverts the sentiments being expressed and social media being an informal platform, increases the chances of sarcasm in the text. Detecting sarcasm is a challenging task even for humans. The thesis aims to automate the task of sarcasm detection to facilitate identification of true sentiments from the text.
Due to unofficial nature of social media, there has been an increase in its usage. This has resulted in the use of regional languages. People usually mix their regional language with English or any other language known to them. To identify the actual sentiment, it is essential to deal with the code mixed text. Automated processing of code mixed text can be very challenging due to lack of readily available tools such as named entity recognizer and part of speech tagger. The thesis attempts to develop necessary tools for processing code mixed text and further uses these tools for the applications related to code mixed text. Due to increasing text on social media, it is becoming increasingly important to focus on finding solutions for above mentioned problems and challenges.

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations/Symbols

Term	Definition
ATSSI	Abstractive Text Summarization using Sentiment Infusion
ANN	Artificial Neural Network
BRNN	Bidirectional Long Short Term Memory Recurrent Neural Network
CRF	Conditional Random Field
CNN	Convolutional Neural Network
DPIL	Detecting Paraphrases in Indian Language
DUC	Document Understanding Conference
FIRE	Forum of Information Retrieval Evaluation
GRU	Gated Recurrent Units
GAN	Generative Adversial Networks
HAPD	Hybrid Approach for Paraphrase Detection
HMM	Hidden Markov Model
HMMW	Hybrid Multi Model Weighting
LR	Logistic Regression
LSTM	Long Short Term Memory
LCS	Longest Common Subsequence
MT	Machine Translation
MSIR	Mixed Script Information Retrieval
MLP	Multilayer Perceptron
MSS	Multilingual Single Document Summarization
NB	Naive Bayes
NE	Named Entities
NER	Named Entity Recognition
NLP	Natural Language Processing
NASA	Neural Network Based Architecture for Sentiment Analysis
POS	Part of Speech
ROUGE	Recall Oriented Understudy of Gisting Evaluation
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SWN	SentiWordNet
SDDL	Spam Detection based on Deep Learning
SDML	Spam Detection based on Machine Learning
SVM	Support Vector Machine
TGGAN	Text Generation using Generative Adversial Networks
WGAN	Wasserstein Generative Adversial Networks

Chapter 1

Introduction

With the rapid growth in usage of social media, users are experiencing lesser communication barriers worldwide. This has not only widened the scope of communication but has also provided users with a chance of expressing their opinions or views publicly. Opinions or sentiments expressed on social media influences people in a positive or negative manner. Opinions play a fundamental role in the decision-making process of both individual and organizations as it impacts people's attitudes and beliefs [Liu 2012]. Opinions make people more aware of their surroundings and views of their fellow beings. This has led to increased interest in mining and analyzing sentiments.
Sentiment analysis is a computational study of people's opinions, attitudes and emotions [Medhat *et al.* 2014]. Sentiment analysis and opinion mining are often referred interchangeably by researchers with few exceptions and hence, both the terms have been referred interchangably in the thesis. There are three primary classification levels in sentiment analysis, i.e. document level, sentence level and aspect/feature level. Document-level sentiment analysis aims to classify an opinionated document expressing a positive or negative opinion or sentiment. It considers the whole document as basic information unit about the specific topic of interest. Sentence level sentiment analysis is similar to that of document-level sentiment analysis where sentences are considered as short documents [Liu 2012]. Sentence level sentiment analysis identifies subjectivity of a sentence followed by the classification of a sentence expressing positive, negative or neutral sentiment. Users do not necessarily express a single opinion in a sentence or document. For example, consider the sentence "iPhone 6 has great features, but it is costly". In the above example, the

user has expressed both sentiments (negative and positive) in the single sentence. Hence, the assumption made by the sentence level sentiment analysis is not always necessarily correct, i.e. a sentence may contain more than one type of opinion for different aspects of a topic or product. Similarly, a document may contain mixed opinions. Classifying text at document level or sentence level does not provide all the necessary details that may be important for the end user. Aspect level sentiment analysis identifies different aspects/features and associates the corresponding opinion with it. Figure 1.1 shows a generic block diagram of sentiment analysis. A opinionated document is used for object/feature extraction, opinion holder extraction and subjectivity classification. Information extracted from all the phases is passed to sentiment classification phase, where sentiments are identified. This flow of the process is generic and varies upon the level of sentiment analysis.

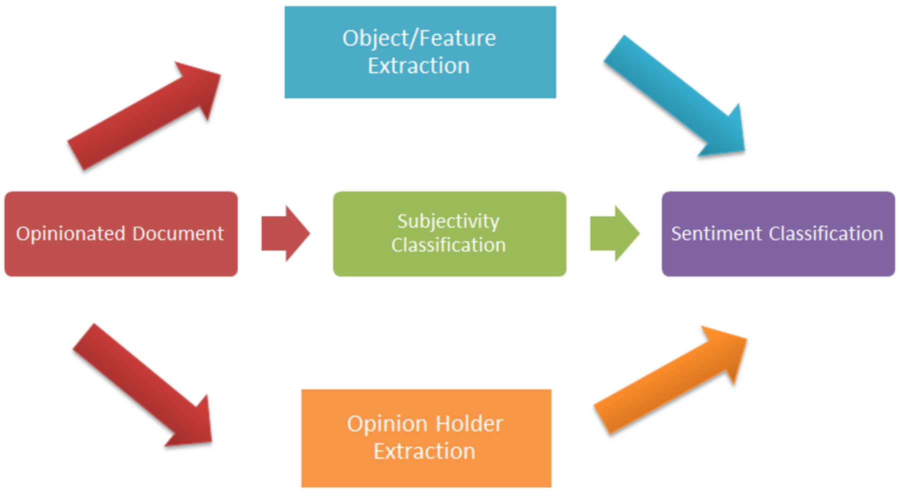

Object/Feature Extraction
Opinionated Document
Subjectivity Classification
Sentiment Classification
Opinion Holder Extraction

Figure 1.1: Block Diagram for Sentiment Analysis

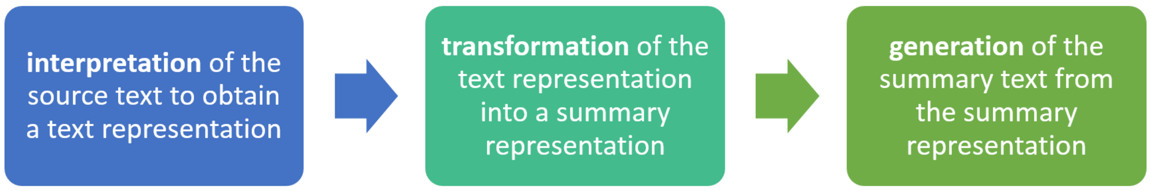
1.1 Motivation

Sentiment analysis finds its roots in various areas of text processing and extraction. It has enabled people in business to get insights of opinions of customers about their products without surveys [Hu & Liu 2004]. It has also assisted people in politics [Tumasjan *et al.* 2010], stocks [Zhang & Skiena 2010] and public policies [Velásquez & Gonzalez 2010]. Over the time, people change their own opinion as well, and different aspects evolve about a topic. Importance of aspect and its opinion keep changing with time, and hence, temporal opinion mining has become an essential challenge for sentiment analysis and opinion mining.

With multiple different data sources on the internet, users have a tremendous amount of text to review and cull out information. The manual process involving a significant amount of text processing to get to a right set of information is quite laborious. Due to this, there is a need for processing and representation of the text such that it can present right set of information. A procedure is needed to identify relevant parts of the text and remove the redundant information present in it. This can be achieved through text summarization.

According to [Mani *et al.* 1999], text summarization is the process of distilling the most important information from a sources to produce an abridged version of the text for a particular set of users and tasks. Figure 1.2 represents the process of text summarization. Text summarization can be classified based on various parameters such as text input, language and purpose. Traditionally it focuses on text input, which can range from simple text to multimedia content. Further, it can also be categorized based on single document or multiple documents. Apart from these, text summarization has also been classified in the literature on the basis of language (monolingual, transiting or multilingual), output (abstract or extract) and purpose (generic or user-focused) [Lloret 2008]. [Jones *et al.* 1999] distinguishes three classes of context factors used for effective text summarization: input (text form, subject type and unit), purpose (situation, audience and use) and output (material, format). The main challenges faced in text summarization are of identifying the similar set of information with little or no difference. As both the challenges are significant, a decision to choose the correct objective needs to be taken. Text Summarization algorithms must have capabilities to preserve the information and concisely represent it. Moreover, existing techniques and algorithms should be able solve the task of summarization for Indian Languages as well. This motivated our work on the problem of text summarization.

Reviews and opinions expressed on social media have influenced people across the globe. Opinions or views expressed on these platforms are generated for various reasons. Companies conduct surveys to enhance and improve the image and quality of their product.



interpretation of the source text to obtain a text representation
transformation of the text representation into a summary representation
generation of the summary text from the summary representation

Figure 1.2: Basic procedure of Text Summarization

People participating in these surveys express their views/opinions which can be used for decision making. While these reviews may be helpful, they cannot be trusted entirely, because reviews may be faked/biased. According to [Lau *et al.* 2011], the BBC and New York Times have reported that fake reviews are becoming a common problem on the Web, and a photography company has recently been subjected to hundreds of defamatory consumer reviews. In 2014, the Canadian government issued a warning, encouraging consumers to be wary of fake online endorsements that give the impression that they have been made by ordinary consumers and estimated that one-third of all online reviews are fake[1]. Over the period, few resources such as Consumerist[2] and MoneyTalksNews[3] have offered tips to help consumers spot fake reviews. Sufficient amount of work has been done in email spam detection and web spam, but the area of spam combating in social media is still not mature enough [Chakraborty *et al.* 2016a]. This is mainly due to the fact, that social media network is ever increasing and has provided marketers with a cheap and new mode of communication. A survey[4] conducted by brand protector found that rate of increase in spam during a period from January 2013 to July 2013 has gone up by 355%. Another study[5] also proved that spam proliferation of is rampant. As spam hinders the utility and performance of social media text [Chakraborty *et al.* 2016b], it is essential to identify spam opinions to provide the user with the correct opinion about a topic/person/product. [Dixit & Agrawal 2013] has categorized spam reviews into untruthful reviews, reviews for brands and non-reviews. The unthruthful reviews are the ones which intentionally donot express geniune opinion for the product. Review for a

[1]http://www.competitionbureau.gc.ca/eic/site/cb-bc.nsf/eng/03782.html
[2]https://consumerist.com/2010/04/14/how-you-spot-fake-online-reviews/
[3]http://www.moneytalksnews.com/2011/07/25/3-tips-for-spotting-fake-product-reviews-%E2%80%93-from-someone-who-wrote-them/
[4]http://www.moneytalksnews.com/2011/07/25/3-tips-for-spotting-fake-product-reviews-%E2%80%93-from-someone-who-wrote-them/
[5]http://www.baselinemag.com/security/malware-attacks-and-phishing-scams-increase.html

brand, product or specific seller are ones that talk about the company of the product rather than talikng about the product. The non reviews are the ones pertaining to advertisement or unrelated text. Identifying fake reviews has been considered as the most challenging task at hand. [Chakraborty *et al.* 2016b] have classified social spam into four significant categories: malicious links (link that damage, deceive or otherwise harm a user or computer), fake profiles, bulk summaries (known as spam bombs, these are comments posted multiple times or comments with similar text) and fraudulent reviews (the reviews written with an intent to falsify the image of product/brand or create a wrong impression on users). Above mentioned problems and challenges motivated us to work on spam detection.

Social media provides an informal communication platform for users. This allows users to express their emotions freely and in an informal way. Sarcasm is a verbal form of irony that is intended to express contempt and ridicule[1]. Sarcasm has become an integral part of the social media text as it is commonly used by people to express their opinion. Sarcasm is characterized as the ironic and satirical wit that is intended to insult, mock or amuse [Riloff *et al.* 2013]. Sarcasm transforms the polarity of an apparently positive or negative utterance into its opposite [González-Ibánez *et al.* 2011]. It is usually difficult to detect sarcasm as it is considered as a graceful and smart way of saying something implicitly, whether good or bad. With advancements in automatic processing of social media text, sarcasm has been a challenge for many Natural Language Processing (NLP) based systems. Informal nature of social media text has resulted in an unstructured ambiguous text which contains URL, username, user-defined labels. [Tsytsarau & Palpanas 2012] has listed following four challenges in sarcasm identification:

1. In spoken statement, sarcasm can be identified with particular tone or facial expression but for written text, no such clues are available. This makes it difficult to detect sarcasm in written text.

2. In a sarcastic statement, positive words are used to convey negative opinion.

3. Domain knowledge is required in many cases of sarcasm detection. For example, “Yet, another mind-blowing performance of Indian team in Sri Lanka."

[1]www.freedictionary.com

4. Sometimes sarcasm uses the hyperbole. Hyperbole is the use of exaggeration, i.e. use of words belonging to a superlative degree. For example "Extraordinary performance in exam!!"

Above stated problems and challenges motivated us to take a step towards the problem of sarcasm detection. Detecting sarcasm can further help in improving sentiment analysis of the text. Over the period of the time, evolution of social media has provided new opportunities for information access and language technology, but it has also thrown new challenges. Few of the challenges being faced commonly are those of understanding and processing spelling errors, creative spellings (e.g. 'gr8' for 'great'), phonetic typing, word play (e.g. "2 looooooooong" for "too long"), abbreviations (e.g. 'OMG' for "Oh My God!") and code mixing (e.g. "but Jadavpur University te physics nya porte chai" for "but I wanted to study physics at Jadavpur University" [Das & Gambäck 2015]). It is evident from the above examples that there is a need to develop tools and applications for the other languages as well. Non-English speakers do not always use Unicode to write in their native language. They use phonetic typing, frequently insert English elements (through code mixing and Anglicism), often mix multiple languages to express their thoughts, making automatic language detection in social media texts very challenging [Das & Gambäck 2015]. Code-switching can be defined as the alternation between two or more languages, language varieties, or language registers in discourse between people who have more than one language in common. Code mixing is defined as the alternation of two or more languages within a sentence [Moradi 2014]. One of the two languages is commanding language; the primary language is often called the matrix language, while the minor language is the embedded language [Moradi 2014]. [Ling *et al.* 2013] found that people often switch between two or more languages on social media, both at conversational level and at the message level. Extracting knowledge from such a text is a difficult task as code mixing occurs at different levels, resulting in the following challenges:

1. Code mixed social media text is multilingual. Due to this, semantics is spread across languages [San 2009].

2. Social media data does not have specific terminology [Das & Gambäck 2014].

There is a growing need to develop automated processing tools and applications that can analyze code mixed text. This requirement motivated us to develop tools for code mix social media text and develop applications based on code mix text.

1.2 Research Gaps

Research gaps in the field of sentiment analysis are as follows:

1. India is one of the most diversified countries in terms of languages and population. A considerable population uses social media for a variety of purposes. In depth analysis of text containing different Indian languages for the sentiment identification need to be carried out.

2. With advancements and ever growing changes, it is important to identify different aspects and their respective opinions being talked about on social media text. It is essential to identify variation in the sentiments over the period.

3. Extracting opinions from different social media sources and regions can lead to the higher availability of information. It also brings various challenges with it such as uniformity, redundancy and language switching. Text summarization may help in solving few of these problems by summarizing the critical content and removing the redundancy present in the text.

4. People expressing an opinion may not be genuine all the time. Moreover, advertisements are also included in the text which may not be relevant while evaluating the sentiment. It has become essential to identify spam in social media text.

5. Social media being an informal platform for communication consists of large amount of text containing sarcasm. Sarcasm detection has been considered as a challenge in text processing and sentiment analysis.

6. With a vast number of users spread across the globe, there has been an increase in usage of code mixing in social media text. There is a lack of efficient tools for processing code mix text.

1.3 Thesis Goals/Objectives

This thesis addresses the following challenges:

1. Identifying sentiments present in social media text containing monolingual Indian languages. With changing aspects and their respective sentiments in a social media text, thesis proposes a solution to the problem of temporal sentiment analysis.

2. Providing precise, unambiguous and vital information is need of the hour and hence, the problem of text summarization has been targeted.

3. Spam in social media text may hinder the accurate evaluation of sentiments. Moreover, due to informal nature of the social media platform, it is usual for users to post sarcastic comments or messages. This may reduce effectivness of the automatic processing of text for sentiment analysis. The thesis aims at identifying spam and sarcasm in social media text.

4. Code Mixing has been a significant challenge in automated processing of social media text. The thesis focuses on devlopment of tools and applications for code-mixing. Part of Speech (POS) taggers are mostly available for all the monolingual languages, but POS tagging of code mixing text has been a challenge for quite some time now. Similar to POS tagging, Named Entity Recognition (NER) for code mixed text is another challenging task. The thesis targets at building POS taggers and named entity recogniser for code mixed text. Question classification is one of the significant challenges of question answering system. This thesis aims at resolving the problem of question classification in code mixed social media text. To understand the text, one has to interpret the meaning of each word to make some sense with the sentence. Hence, a word level language identifier has been proposed in the thesis. Research has been going on in the field of sentiment analysis for past decade but the introduction of code mix text has added many new challenges. The problem of sentiment analysis in code mix text has been targeted in this thesis.

For achieving these objectives, the thesis focuses on various machine learning classifiers including deep learning techniques for solving the natural language processing task.

1.4 Scope of Thesis

Sentiment analysis has been a challenge for researchers for a decade and good amount of research is being carried out addressing various involved issues. The thesis focuses on sentiment analysis and its challenges such as text summarization, spam detection, sarcasm detection and code mixing. Learning based approaches have been proposed for sentiment analysis of monolingual Indian language. With time aspects and opinions change about a topic or a product, hence it becomes essential to analyze these changes. The thesis focuses on temporal sentiment analysis as well. However, the thesis has limited its scope to problems mentioned above. Challenges such as implicit opinion mining and identification of provoking and controversial statements have not been targeted in the thesis. The process of manual annotation is complicated in case of spam and sarcasm due to uncertainty and no clear indication in the text, which is a major reason for lack of annotated datasets. In case of code mixing, due to lack of sufficient amount of data available to train the models correctly, code mixed text generation is essential but it is not included in the thesis.

1.5 Thesis Organization/ Outline

Thesis is organized as follow:
In Chapter 2, background information and related work are detailed. This chapter discusses the state of art and literature review.
Chapter 3 discusses different proposed solutions for sentiment analysis in Indian languages and temporal sentiment analysis. Different deep learning based approaches have been proposed to solve these problems. Approach for sentiment analysis in Indian language uses different layers of the neural network to identify the sentiment or opinion expressed in the text. The approach proposed for temporal sentiment analysis identifies ever changing aspects and their varying sentiments over the time using clustering and Convolutional Neural Network (CNN) respectively.
Chapter 4 describes the proposed algorithms for paraphrasing, extractive text summarization and abstractive text summarization in English and Indian Languages. Two algo-

rithms have been proposed for the problems. The first algorithm for paraphrase detection in Indian languages relies on different machine learning classifiers whereas the second one relies on CNN(s) and Reccurent Neural Networks. In extractive Text Summarization, three different approaches have been proposed considering three different scenarios. The first proposed approach uses the scoring technique, the second approach uses machine learning technique and the third approach uses fully connected CNN for generating extractive text summaries of English and Indian Languages. For abstractive summarization of English text, the sentiment infusion based technique has been proposed in this chapter which infuses the sentiment to reduce the size of the text and maximize the information while generating the summaries. This chapter also proposes a generic algorithm for generating text summaries in English and Indian languages using deep nets.

Chapter 5 describes the proposed algorithm for spam detection in social media text using deep learning techniques. Different unified architectures of deep learning have been proposed in this chapter followed by their comparison with each other and machine learning based approach.

Chapter 6 explains proposed solutions to sarcasm detection in English, Indian languages and code mixed scenario. Proposed algorithms rely on different machine learning and deep learning techniques. Experiments and comparisons among different proposed algorithm have been discussed in the chapter.

Chapter 7 and 8 highlight the problem and various challenges associated with code mixing. Tools and applications have been built for handling code mixed social media text such as POS Tagger, named entity recognizer, language identification, question classification and sentiment analysis.

Chapter 9 discusses the overall results, contributions and further improvisations are proposed.

Chapter 2

Related Work

2.1 Sentiment Analysis

Opinions, sentiments and emotions are a significant part of a human's life choices and behaviour. Sentiment Analysis (SA) is a fascinating problem since it deals with the study of one's emotions expressed via text. Sentiment analysis (also known as opinion mining) refers to the use of natural language processing, text analysis and computational linguistics to identify and extract subjective information in source materials [1]. User-generated content has been a valuable source of information for a variety of sources. Sentiment analysis and opinion mining are two emerging fields and are used interchangeably. However, few pieces of research consider them different. According to [Tsytsarau & Palpanas 2012], opinion mining is about determining whether a piece of text contains opinion, a problem that is also known as subjectivity analysis, whereas the focus of SA is the sentiment polarity detection by which the opinion of the examined text is assigned a positive or negative sentiment. [Pang *et al.* 2008] presented an extensive survey about challenges tasks in the field of sentiment analysis. [Tang *et al.* 2009] focused on four problems of opinion mining, i.e. subjectivity classification, word sentiment classification, document sentiment classification and opinion extraction. Another comprehensive survey on sentiment analysis was done by [Liu & Zhang 2012] where the author discussed all the essential concepts and topics related to SA. In 2012, [Tsytsarau & Palpanas 2012] presented another survey with definitions, problem discussion and various approaches. In 2016 [Giachanou &

[1]https://en.wikipedia.org/wiki/Sentiment_analysis

Crestani 2016], discussed the challenges(Text length, Topic relevance, Incorrect English, Data sparsity, Negation, Stopwords, Tokenization, Multilingual content, Multimodal content) associated with SA in Twitter. [Giachanou & Crestani 2016] also categorized Twitter Sentiment Analysis (TSA) approaches into four different classes, machine learning, lexicon based, graph-based and hybrid of machine learning and lexicon based.
One of the earliest work done in sentiment analysis is by [Pang *et al.* 2002]. They evaluated several supervised machine learning classifiers like Naive Bayes (NB), Support Vector Machine (SVM) and maximum entropy classifier on movie review dataset and achieved an accuracy of 82.9% with SVM and 81.0% using Naive Bayes. Later majority of approaches dealing with the problem of sentiment analysis has been developed to detect the overall sentiment polarity of a sentence. In 2009, [Go *et al.* 2009] employed a distant supervision technique to perform SA. They used technique demonstrated by [Read 2005] to collect the data and used emoticons to differentiate between the sentiments. A similar approach was adopted by [Pak & Paroubek 2010], but they performed multi-class classification instead of binary classification. [Davidov *et al.* 2010] leveraged hashtags and emoticons in tweets for collecting training data and presented a supervised algorithm similar to *K*-Nearest Neighbors (KNN). [Jiang *et al.* 2011] used combined target dependent and independent features and defined manual rules for detecting syntactic patterns for identifying if the term was related to the specific object. [Asiaee T *et al.* 2012] proposed a three-step cascade classifier framework for SA where they identified the topic in the first step, in the second step they identified the sentiment of the tweet and in the last step, they decided the polarity of the tweet. [Hamdan *et al.* 2013] examined features including concept from DBpedia, verb groups and adjectives from WordNet [Pedersen *et al.* 2004] and senti-features from SentiWordNet (SWN) [Baccianella *et al.* 2010a]. Authors also employed a dictionary for emoticons, abbreviations and slang words. [Aston *et al.* 2014] represented tweets using character *n*-gram and then selected top *n* features of a gram using different evaluation algorithms such as chi-squared, gain ratio and info gain.
[Lin & Kolcz 2012] used hashed byte 4-grams and applied linearly combined Logistic Regression (LR) classifiers with different size of ensembles. Another ensemble approach was proposed by [Da Silva *et al.* 2014] with more number of classifier (Random Forest (RF), SVM, Multinomial Naive Bayes (MNB), LR) being involved. In 2013, [Hassan *et al.* 2013]

used most common features including uni-grams, bi-grams, POS and semantic features and proposed a bootstrapping ensemble framework. They also claimed that their framework could be used to build time series as well. In 2016, [Poria *et al.* 2016b] proposed an approach for multimodal sentiment analysis where authors used both feature and decision level features to merge affective information from different sources. [Zimbra *et al.* 2016] proposed an approach based on Dynamic Architecture for Artificial Neural Networks (DAN2). Unique characteristics of Twitter and sentiment expression were used for feature engineering. [Märkle-Huß *et al.* 2017] used rhetoric structure theory to represent text at document level in a hierarchical manner. Authors proposed a combination of weights and grid search over the RST tree generated to predict SA.
With the increasing trend of deep learning, [Jiang *et al.* 2011] proposed adaptive recursive neural network for SA which used dependency tree to find syntactically related words and propagate sentiment from sentiment words. [Tang *et al.* 2014] learned sentiment specific word embeddings using indirect supervision whereas citetang2015learning used word embeddings from a large amounts data to represent the semantic representation of users and products. [Vo & Zhang 2015] split tweets into two parts and build the model such that word embeddings of two context were used to identify SA. [Lopez & Kalita 2017] have explained how CNN can be applied to NLP for sentiment analysis. Later [Shirani-Mehr 2014] analyzed different deep learning methods for sentiment classification of movie reviews with CNN model using word2vec word embedding performing 88.2% accuracy. [Araque *et al.* 2017] proposed deep learning using word embedding and linear machine learning algorithm. Authors aggregated the proposed approaches and performed experiments on different variants of both the approaches. [Hassan & Mahmood 2017] proposed ConvLstm, neural network architecture that employs CNN and Long Short Term Memory (LSTM) on top of pre-trained word vectors.

2.1.1 Sentiment Analysis in Indian Languages

In recent years, there has been much flow of information in Indian languages on World Wide Web. It is easy to deduce the sentiment from the spoken text by the tone of the speaker; whereas, in case of written text, the context becomes the tool for determining the polarity. Being low on the resource, SA in Indian languages has been difficult. Sentiment analysis is widely applied to reviews and social media for a variety of applications, ranging from marketing to customer service. In table 2.1 existing literature survey has been presented in a tabular form where the references are cited in the second column, with the language and year in third and first column respectively. The polarity of classification used is mentioned in the fourth column. The sixth column elaborates on the techniques and approach used in the respective papers, which can be focused upon by the reader according to his field of interest. These include machine learning and lexicon based techniques, and further elaborate on the methodology or approach used. The scope of the data used for evaluation of the article's algorithm is mentioned in the fifth column. It could be reviews collected from the newspapers websites, BBC Hindi, blogs, Twitter posts, movie songs and text paragraph.

Year	Author	Language	Data Scope/ Text Domain	Techniques/ Algo Used
2014	[Hasan *et al.* 2014]	Bengali	Text Paragraph	Lexicon-Based Approach • WordNet to get the senses of each word according to its parts of speech and SentiWordNet to get the prior valence (i.e. polarity) of each word • Features used: POS tags
2014	[Chowdhury & Chowdhury 2014]	Bengali	Tweets	Machine Learning • Semi-supervised bootstrapping approach for the development of the training corpus which avoids the need for labor intensive manual annotation • SVM and Maximum Entropy (MaxEnt) for classification • Construction of a Twitter-specific Bangla sentiment lexicon • Features used: Word n grams (uni and bi gram), emoticons, lexicon(bangla), POS tags, Negation. Combination of features is used

Year	Author	Language	Data Scope/ Text Domain	Techniques/ Algo Used
2016	[Muhammad *et al.* 2016]	Bengali	1500 short Bangla comment from various social sites	Machine Learning • TF-IDF (Term Frequency-Inverse Document Frequency) • Features used: Most Frequent Words, Starting and Ending Letters,Sentence Length and Pattern Analysis
2010	[Das & Bandyopadhyay 2010a]	Bengali	Dataset consisted of the following: • news reports that aim to objectively present factual information • opinionated articles that clearly present author's and reader's views, evaluation or judgment about some specific events or persons	Machine Learning • SVM classifier • Features used: SentiWordNet(Bengali), Negative Word, Stemming cluster, Functional Word, Chunk, POS, Dependency tree feature.

Year	Author	Language	Data Scope/ Text Domain	Techniques/ Algo Used
2013	[Mittal *et al.* 2013]	Hindi	Hindi review websites	Lexicon Based • Rules are devised for handling negation and discourse relation • Hindi SentiWordNet (HSWN) is used for polarity values of words • Improved HSWN is created using assumption, all synonyms are of same polarity and all antonyms are of reverse polarity • Features used: HSWN, Improved HSWN, Negation, Discourse relations(Conj_after, Conclusive or Inferential conjunction)
2014	[Sharma *et al.* 2014]	Hindi	Movie reviews were collected from the Hindi newspapers website	Lexicon Based • Unsupervised dictionary approach • The polarity of the reviews is determined on the basis of majority of opinion words • Features used: POS Tags, Opinion words and seed list , Negation

Year	Author	Language	Data Scope/ Text Domain	Techniques/ Algo Used
2014	[Ghosh & Dutta 2014]	Hindi	Twitter posts in Hindi	Machine Learning and Lexicon Based • Resource-based approach • Naive Bayes classifier • Features considered: HSWN, Negation
2015	[Sharma *et al.* 2015c]	Hindi	Hindi tweets	Lexicon Based • Subjective lexicon method. • SentiWordNet creation which contains adjectives and adverbs • Features considered: POS Tags, Negation
2015	[Kumar *et al.* 2015b]	Kannada	"182 positive Kannada reviews and 105 negative Kannada reviews reviews were mainly collected for broad domains consisting of commercial products"	Machine Learning • Classifiers used: J48, Random Tree, ADT Tree, Breadth First, Naive Bayes and SVM • Features used: Dictionary of positive and negative words, Negation, POS Tags, Turney's method [Turney 2002], Sentence based approach [Khan & Baharum 2011].

Year	Author	Language	Data Scope/ Text Domain	Techniques/ Algo Used
2015	[Anil *et al.* 2015]	Kannada	Same as [Kumar *et al.* 2015b]	Lexicon based • Pattern based approach • Features used: Kannada dictionary (self generated), Turney's algorithm [Turney 2002], Negation, Significant Sentence algorithm [Khan & Baharum 2011]
2015	[Anagha *et al.* 2016]	Malayalam	different movie reviews from various web sites	Machine Learning and Lexicon-based • Maximum Entropy Model is used for tagging • Maximum Entropy Classification ïňĄnds out in which class the review must belong, given a context so that it maximizes the entropy of the classiïňĄcation system.
2016	[Beegum & V.A 2016]	Malayalam	Data sets of movie reviews	Lexicon-based • Training phase: Creation of phase dictionary and a polarity dictionary • In phrase dictionary store the score value and domain of each single word and its combination. Polarity dictionary contain the words, combination words and its polarity.

Year	Author	Language	Data Scope/ Text Domain	Techniques/ Algo Used
2012	[Balamurali 2012]	Marathi	Travel Reviews from various blogs and Sunday travel editorials	Machine Learning • SVM • Feature used: WordNet senses
2013	[Deepali & Garg 2013]	Punjabi	Punjabi newspaper sites and Punjabi blogs	Machine Learning and Lexicon-based • Feature used: N-Gram approach (Frequency) • Naive Bayes
2014	[Sharma 2014]	Punjabi	different websites, newspapers, blogs	Machine Learning and Lexicon-based • Feature used: N-Gram approach • Naive Bayes

Year	Author	Language	Data Scope/ Text Domain	Techniques/ Algo Used
2014	[Kaur & Gupta 2014]	Punjabi	Text Paragraph	Lexicon-based • Using subjective lexicon created by using Hindi Subjective Lexicon. • three popular methods are used for the generation of subjective lexicon- – Use of Bi-Lingual Dictionary – Machine Translation (MT) – Use of Word net • Negation Handling
2016	[Se *et al.* 2016]	Tamil	Tamil movie reviews	Machine Learning • Classifiers used: SVM, MaxEnt, Decision Tree, Naive Bayes • Features used: Tamil SentiWordNet, Punctuations and Apostrophe

Year	Author	Language	Data Scope/ Text Domain	Techniques/ Algo Used
2016	[Kausikaa & V 2016]	Tamil	Tweets	Machine Learning • Translation of Tamil Tweets into English Tweets. • Finding out the semantic similarity using path-length similarity. • Classification of Sentiments using SVM.
2016	[Phani *et al.* 2016]	Tamil, Hindi, Bengali	Tweets	Machine Learning • Features used: Word n gram, Character n gram, Surface features, SentiWordNet features • Classifier used: Multinomial Naive Bayes, Logistic Regression (LR), decision tree, Random Forest (RF), SVM, SVC, and SVM Linear SVC (LS) used

Year	Author	Language	Data Scope/ Text Domain	Techniques/ Algo Used
2012	[Manchala *et al.* 2012]	Telugu	Telugu blog texts and News headlines of English SemEval 2007	Machine Learning • Conditional Random Field (CRF) based classifier has been applied for recognizing six basic emotion tags • A score based technique has been adopted to calculate and assign tag weights to each of the six emotion tags • A sense based scoring strategy has been applied to identify sentence level emotion scores for the six emotion tags based on the acquired word level emotion tags. • Feature used: POS, First sentence in a topic, SentiWordNet, Reduplication, Question Words, Special Punctuations, Quoted sentences, Emoticons, Uni gram and Bi-gram
2016	[Mukku *et al.* 2016]	Telugu	A corpus consisting of 7,21,785 raw Telugu sentences was provided by Indian Languages Corpora Initiative (ILCI)2 e Telugu Newspapers	Machine Learning • Classifier used: Naive Bayes, Logistic Regression, SVM, Decision tree, Random forest, Multi Layer Perceptron(MLP) Neural Network, Adaboost ensemble • Feature used: Doc2Vec [1]

[1]https://radimrehurek.com/genism/index.html

Year	Author	Language	Data Scope/ Text Domain	Techniques/ Algo Used
2016	[Abburi *et al.* 2016]	Telugu	300 Telugu movie songs and lyrics corresponding to each song are taken : YouTube	Machine Learning • Feature used: Lyric Feature, Audio Feature • SVM, NB and a combination of both these classifiers are developed to classify the sentiment using the textual lyric features. • Gaussian Mixture Models (GMM), SVM and a combination of both these classifiers are developed to classify the sentiment using audio features.

2.1.2 Temporal Sentiment Analysis

As for temporal analysis, being a relatively new field, so only few research have been done in it. [An *et al.* 2014] presented a study to yield insights on climate change sentiment using social media text. They used existing sentiment analysis algorithms, data-mining techniques, and time series methods with the aim to detect and track sentiment regarding climate change from Twitter feeds. [Bollen *et al.* 2011] mapped everyday tweets to a six-dimensional mood vector (tension, depression, anger, vigor, fatigue, confusion). They compared the results to the timeline of cultural, social, economic, and political events that occurred during the same period. After analyzing the impact of world global events on the mood of microblog posts, they found that the mood level in posts was correlated with cultural, political, and other events.

In another study, [O'Connor *et al.* 2010] tried to identify the relationship between opinion expressed in tweets and the public opinion obtained by polls. Authors retrieved relevant tweets to some specific topics and then estimated the sentiment score of every day. They used a simple lexicon-based approach and the Multi-Perspective Question Answering (MPQA) sentiment lexicon [Wiebe *et al.* 2005] for identifying the sentiment score. Sentiment time series were then produced with an average moving window of past k days. It was found that there was a strong correlation between the time series and the polling data on customer confidence and political opinion.

[Bifet & Frank 2010] proposed a real-time sentiment analysis model using a data-stream mining approach. The proposed approach could monitor the evolution of the impact of words on class predictions. The linear classifier was used to learn the stochastic gradient descent (SGD) method that had a similar performance with multinomial naive bayes. [Hao *et al.* 2011] focused on visual sentiment analysis and explored three different approaches on a large volume of tweets. They proposed a topic-based text-stream analysis method to determine the topic of discussion based on a number of opinionated attributes. [Das *et al.* 2011] have developed a system called TempEval, tool for visualizing the change of opinion with time using machine learning algorithm CRF using sentiment as a feature of the event. [Bjørkelund & Burnett 2012] has developed a temporal analysis framework for hotel reviews which used naive bayes and SentiWordNet for sentiment classification.

[Xia & Zhiwen 2016] also used supervised machine learning algorithms such as naive bayes and decision tree for temporal sentiment analysis of Twitter data.

2.2 Paraphrase Detection

Paraphrase detection has been a major area of research in the recent times because of its significance in many areas of natural language processing. Extensive work has been done on developing machine learning techniques. Most of the previous works in paraphrase detection carefully engineer syntactic or semantic features or use heuristics. [Islam & Inkpen 2008] proposed a modified form of Longest Common Sub-sequence (LCS) algorithm whereas [Mihalcea *et al.* 2006] used corpus-based similarity measures to detect paraphrases. These techniques fail to take into account the fact that in a sentence, many short sequences add to the structure and importance of the long sequence regardless of where they occur in the sentence. Hence, a deep learning technique that identifies a feature paying little attention to the position of occurrence should be able to help overcome these shortcomings. [Vo *et al.* 2015] proposed simple features like n-grams, edit distance scores, METEOR (Metric for Evaluation of Translation with Explicit Ordering) word alignment, BLEU (Bilingual Evaluation Understudy) for detecting paraphrases and semantic similarity tasks on Twitter data. Similarly, analysis of various similarity measures like sentence-level edit distance measure, simple n-gram overlap measure, exclusive longest common prefix (LCP) n-gram measure, BLEU measure and sumo measure along with paraphrase detection based on abductive machine learning has been proposed in [El-Alfy *et al.* 2015]. [Malakasiotis 2009] proposed three methods for paraphrase detection using string similarity measures.

One of earliest work for paraphrase detection in deep learning was done by [Sundaram *et al.* 2009] where authors proposed an unsupervised feature learning technique with Recursive Auto-encoders (RAE) for detecting paraphrases on Twitter. In their proposed technique they first converted data to parse trees using the phrase-structure parser and then passed it to the RAE for training. The vector generated from the RAE is converted to form a similarity matrix and thus paraphrase detection is done using this matrix. [Huang 2011a] has proposed an unsupervised recursive auto-encoder architecture for paraphrase detection. The recursive auto-encoder uses *tanh* as the sigmoid-like activation function and gives the representation of sentences along with their sub-phrases. These representations are then used for paraphrase detection. Two approaches are used to extract the same number of features for different sentence pairs, aggregating representations to form a single feature and using a similarity matrix approach. With the first approach, they achieved 66.49% accuracy while with the second method accuracy of 68.06% was achieved. In 2011, [Socher *et al.* 2011] proposed an approach where dynamic pooling and unfolding Recursive Auto Encoders (RAE) were used for the task of paraphrase detection. The RAE(s) are unsupervised and learn feature vectors for phrases using syntactic trees. A dynamic pooling layer was used in their approach which computed fixed-sized vectors from variable input matrices. This fixed size vector was used as an input to the classifier. Authors achieved an accuracy of 76.8% and F-Measure of 83.6% on the Microsoft Research Paraphrase Corpus(MSRPC).

In 2014, [Kalchbrenner *et al.* 2014] proposed a CNN architecture that extracted *k* top values from the convolutional filter to get a fixed length output. To achieve multi-granularity, they stacked several levels of convolutional filters. [Yin & Schütze 2015] proposed architecture using double CNN with multi-granular interaction features for paraphrase identification. The multi-granular features were learned using the CNN. A logistic regression classifier was used for classifying these features for paraphrase identification. Separate input matrices that converged at a later step were used for two sentences. In contrast, the approach defined in this thesis uses a single input matrix for both the sentences.

[He *et al.* 2015] used a CNN that extracted features at multiple levels of granularity and used multiple types of pooling. Their model had two layers; layer one used CNN architecture for capturing different levels of granularity and layer two which compared local regions in a sentence for similarity measurements. Their approach achieved a 78.60% accuracy on MSRPC. WordNet-based lexical similarity approach was used by[Fernando & Stevenson 2008]. Each sentence was represented as a one-hot vector, and then a similarity matrix was built. The matrix was built taking into account all word pairs. The authors achieved an accuracy of 74.1% with an F-measure of 82.4% on MSRPC. In approach proposed in this thesis, a similar WordNet similarity matrix has been to train the CNN.

When considering Indian languages, paraphrase detection has been done using machine learning techniques that use heuristics more than the semantic features. Before [Anand Kumar *et al.* 2016], no work was done on paraphrases for Indian languages because of a lack of dataset. [Anand Kumar *et al.* 2016] opened doors for developing tools to detect paraphrases in Indian languages. At task of Detecting Paraphrases in Indian Languages(DPIL) at Forum of Information Retrieval Evaluation (FIRE) 2016 [Anand Kumar *et al.* 2016], the techniques usually used features such as Jaccard similarity and METEOR metrics which are all heuristic based. A few approaches that use semantic features like Soundex codes, POS taggers, do not perform as well as the heuristic-based approaches because of the error introduced in these steps. [Kong *et al.* 2016] use gradient boosting algorithm for classification. The features used include Jaccard coefficient, cosine similarity, dice coefficient and other METEOR based metrics. Their approach performed the best on Punjabi(0.932), Malayalam(0.785) and Tamil(0.776) languages. [Bhargava *et al.* 2016a] implemented a supervised classification model for detecting paraphrases. POS tags, stems of words and Soundex codes corresponding to the words in sentences were used as features. They achieved an accuracy of 90.5% and f-measure of 87.6% on the Hindi dataset for DPIL task at FIRE 2016. [Saini 2016a] use various machine learning approaches including random forest and SVM(s). Features used include common tokens in two sentences, common IDF scores and sentence length. Their approach performed best on the Hindi dataset with a F-measure of 0.907.

2.3 Text Summarization

With the rapid increase in large internet users and the content being generated online, automatic text summarization has been evolving as a field of research. [Luhn 1958] work was one of the pioneer work in the area, since then remarkable progress has been made with the popularity of deep learning approaches ([Rush *et al.* 2015], [Chopra *et al.* 2016]). A summary is a brief text that is generated from one or more text. The summary contains the main points from the said text, and the length is usually not more than half of the original text. Text Summarization is the process which gathers the most information from the text(s)to generate the abridged version. Traditionally, summarization focuses on text input, but in the past, there have been systems that accept multimedia input (video or audio). From time to time different categories have been proposed for text summarization on the basis of input, output, purpose and kind of information. Major of them being single and multi-document, extractive and abstractive, generic and query focused ([Dunlavy *et al.* 2007],[Gong & Liu 2001], [Ouyang *et al.* 2011], [Wan 2008]), supervised and unsupervised ([Riedhammer *et al.* 2010], [Mani & Maybury 1999]), monolingual, multilingual and cross-lingual. The summarization systems can be either multi-document [Barzilay *et al.* 1999] or single document [Litvak & Last 2008]. A multi-document can be monolingual or multilingual [Radev *et al.* 2004]. Moreover, the summaries can either be generic or query based [Park *et al.* 2006]. Summarization is traditionally a three-step process:

- Generating a text representation of the source.
- Transforming the representation to a summary representation.
- Generating a summary of the representation.

Three factors determine the quality of the summarization system: input, output and purpose. The input can be either a generic text or topic based, speech transcripts or news documents, single document or multiple documents. The purpose of the summary is to determine the target audience for the summary and if the summary is query based or not. Output factors define the content of the summary.
The NLP community has been working on the task of text summarization since last sixty years. [Radev *et al.* 2002] describe a summary as "a text that is produced from one or more texts, which conveys important information in the original text(s), and that is no longer than half of the original text(s) and usually significantly less than that". This definition throws light on the three essential characteristics of summarization:

- The summary can be produced from one or more than one documents.
- Relevant information should be preserved in summarization.
- The content should be short enough for a quick perusal.

The two most common terms that come up while discussing summarization are extraction and abstraction. Extraction involves identifying the most important sentences from the text and extracting them without any modification. Abstraction aims to present information in a new form with modifications done like compressing sentences, fusing sentences([Bhargava *et al.* 2016c]). Abstractive text summarization aims to produce a more human-like summary.

In the first instances of summarization, extracting features like the position of the sentence in the text ([Baxendale 1958]), word and phrase frequency ([Luhn 1958]) and key phrases ([Edmundson 1969]). Extractive summarization focuses on summary content whereas abstractive summarization puts focus on the grammatical correctness of the sentences, the form of the sentences. One of the issues that summarization systems face is the evaluation. During the last two decades, various competitions such as DUC[1], TREC[2] and TAC[3] have established baseline systems and created training data for evaluating such systems. However, a universal strategy for evaluating summaries is still not in place. More so in Indian languages, there is no data, and no unanimously agreed upon measure for evaluating such summaries.

2.3.1 Extractive Text Summarization

Many approaches have been proposed for Extractive Text Summarization. Few of them being, statistical approaches ([Ko & Seo 2008]), topic based ([Harabagiu & Lacatusu 2005]), graph-based ([Radev *et al.* 2002]), discourse-based ([Chan 2006]) and machine learning based. [Yeh *et al.* 2005] proposed a trained summarizer and latent semantic analysis for summarization of text. They proposed two techniques: Modified corpus-based approach(MCBA) and Latent Semantic Analysis based Text Relation Map (TRM) technique. MCBA depends on score function and analyzes important features for generating summaries. Genetic Algorithm trains the score function for obtaining an appropriate combination of feature weights. It also uses LSA to obtain a document's semantic matrix and uses sentence's semantic representation to build relationship map. [Chan 2006] uses shallow linguistics based techniques. Discourse network is considered where interrelated parts are represented as a single unit, and textual continuity is used to combine different phrases using this discourse. [Ko & Seo 2008] uses contextual information and statistical

[1]http://duc.nist.gov/
[2]http://trec.nist.gov/
[3]https://tac.nist.gov//

approaches. It combines two consecutive sentences into a bi-gram pseudo sentence so that contextual information is applied to statistical sentence-extraction techniques. The statistical sentence extraction techniques first select salient bigram pseudo sentences, and then each selected bi-gram pseudo sentence is separated into two single sentences. The second sentence-extraction task for the separated single sentences is performed to make a final text summary. Few works proposed unsupervised approaches ([Lee *et al.* 2009]) as well for summarization task. Another unsupervised approach was proposed by [Alguliev *et al.* 2011] which uses integer linear programming for identifying relevant text. The approach was named as Maximum Coverage and Minimum Redundancy (MCMR). This approach tries to optimize relevance, redundancy and length of the text. A query based multi-document summarization was proposed by [Ouyang *et al.* 2011] where seven features are used to decide the importance of sentence, in which three features are query dependent whereas four of them are independent of the query. On the basis of *n*-gram pseudo human summary training data and set of documents are compared for relevance score. [Ouyang *et al.* 2013] suggested another approach where all words are organized as Directed Acyclic Graph (DAG). This approach selects new and relevant sentences in two ways. First, it uncovered concepts are taken into consideration only during estimation of the relevance of the sentences to ensure novelty of among sentences. Simultaneously, the relationship between sentences is utilized to enhance saliency measure. [Ferreira *et al.* 2013] used fifteen scoring methods including different sentence scoring, word scoring and graph-based scoring approaches. [Lloret & Palomar 2013] used lexical, syntactic and semantic levels of language analysis to find relevant sentences. [Fattah 2014] used statistical features and trained machine learning techniques, naive bayes classifier, maximum entropy and SVM. [Yang *et al.* 2014] proposed a theme based summarization where sentences are clustered based on the theme. Each theme cluster is based on a generative model. [Fang *et al.* 2015] proposed topic aspect-oriented summarization which is based on topic factors. Various features are used to represent multiple aspects and preferences of the topic. Approach targeted text as well as image summarization. [Heu *et al.* 2015] proposed FoDoSu, which exploits Flicker tag clusters for selecting important sentences. Documents are preprocessed, after which the words obtained are used by word analysis module during which a Word Frequency Table (WFT), where the frequency is computed for each word in the documents and words having high semantic relationships are discovered with the help of tag clusters from Flicker. The WFT gets updated when words having high semantic relationships are discovered. This is calculated using HITS algorithm. After each word's relevance and contribution is analyzed, score of each sentence is computed using rel-gram, and each sentence is ranked with word cluster. The system finally selects highly scored sentences to generate summaries of multiple documents.

2.3.2 Abstractive Text Summarization

Abstractive techniques in text summarization include rule-based approach [Genest & Lapalme 2012], sentence compression ([Clarke & Lapata 2006], [Knight & Marcu 2002], [Zajic *et al.* 2007]), merging sentence based on their semantics ([Liu *et al.* 2015], [Wang *et al.* 2008]). Graph-based algorithms, in particular has been proven to work well on both summarizing texts containing lots of redundant data ([Ganesan *et al.* 2010a] and [Lloret & Palomar 2011]). [Sankarasubramaniam *et al.* 2014] leverage Wikipedia in addition to graph-based algorithms to generate extractive summaries. They first map all the sentences to corresponding Wikipedia topic and thus a bipartite graph is obtained where one of vertices represent the Wikipedia topics, and the other set represent the sentences in the document. [Sankarasubramaniam *et al.* 2014] then uses an iterative ranking algorithm to find the best candidate sentences in the document. [Sankarasubramaniam *et al.* 2014] also introduces incremental summarization wherein longer summaries are generated in real-time by simply adding sentences to shorter summaries. Since the summaries generated are extractive, the precision is less when compared to the results of techniques that generate abstractive summaries.

[Liu *et al.* 2015] use the advances in the semantic representation of the text in the form of Abstract Meaning Representation (AMR) graphs to form summaries. The summarization framework consists of parsing input sentences to form individual AMR graphs, combining the individual AMR graphs to form a summary AMR graph and then generating text from the summary graph. The individual graphs are converted to summary graph using a perceptron model prediction algorithm which predicts with a high accuracy the subgraph that has to be selected for summary generation. [Bhargava *et al.* 2016c] described an approach that used directed graphs. The graphs use the original sentence word order to generate abstractive summaries. Their technique leverages the graphical form of the input text to reduce redundancy. If their algorithm finds two sentences that are collapsible, they use the connectors based on the sentiments of the adjoining sentences. [Ganesan *et al.* 2010a] describe an approach that used directed graphs that use the original sentence word order to generate abstractive summaries. Their technique leverages the graphical form of the input text to reduce redundancy. If their algorithm finds two sentences that are collapsible, they use the connectors already present in one of the sentences to be used as the connector for the collapsed sentence. While this technique is effective, it still has a drawback since there might be two sentences which are capable of being fused together, but can't be fused because of the absence of a pre-existing connector.

[Lloret & Palomar 2011] describes a technique in which they have built a directed weighted word graph where each word text represents a node in the graph and the edge contains the adjacency relation between the words. The weight of the edge is determined by using a combination of their page-rank value and the frequency of the words. To determine important sentences, the first node consists of the first ten words with highest TF-IDF score. Sentence correctness are ensured using the basic rules of grammar like the length a sentence should be greater than three words, a sentence must contain a verb and should not end in an article or conjunction. A huge flaw with this methodology is that a lot of important information is lost because of the impositions of grammar on the sentences and the policy of selecting the ten words with highest TF-IDF scores. Furthermore, a lot of redundant sentences will still be present in the summary because the TF-IDF scores will give more importance to them.

[Moawad & Aref 2012] proposed a new method for generating single document abstractive summaries by reduction using semantic graph. Rich semantic graphs using this approach which in turns generate abstractive summaries. [Lloret *et al.* 2013] proposed two types of COMPENDIUM (abstractive and extractive) in which after choosing important sentences, information compression and fusion stage is implemented. Recall Oriented Understudy of Gisting Evaluation (ROUGE)-1 score obtained for COMPENDIUM$_E$ and COMPENDIUM$_A$ are 44.02% and 38.66% respectively. [Khan *et al.* 2015] proposed an abstractive approach in which summary is generated by semantic representation of

the source documents. Semantic Role Labeling (SRL) was used to represent documents through predicate-argument structures. Similar structures are clustered on basis of semantic similarity. These structures are then ranked on basis of features weighted and optimized by genetic algorithm. [Banerjee *et al.* 2015] proposed an approach wherein most important document is selected from multiple documents. Then each sentence of that important document becomes an individual clusters. Sentences of other documents are then assigned to these clusters based on highest similarities. Using the graph-based approach, *k*-shortest path were generated. Sentences belonging to set of shortest path are then selected using integer linear programming to maximize the content. [Bing *et al.* 2015] extracts concepts and facts represented by noun and verb phrases. Phrases are selected and combined for creation of new sentence which is validated using integer linear optimization. Score for each phrase is calculated on basis of redundancy in document. [Rush *et al.* 2015] proposed abstractive summarization with a data-driven approach. Neural attention model in combination with contextual input encoder is created such that it generates each summary word based on input sentence.
In the recent past, deep-learning based sequence-to-sequence models, have been successful in many problems such as machine translation ([Bahdanau *et al.* 2014]), speech recognition ([Bahdanau *et al.* 2016]) and video captioning ([Venugopalan *et al.* 2015]). Much work has also been done using deep learning for sentence summarization. [Rush *et al.* 2015] proposed a method that utilized a local attention-based model that generated each word of the summary conditioned on the input sentence. While the model was structurally simple, it could easily be trained end-to-end and scaled to a significant amount of training data. [Lopyrev 2015] proposed an application of an encoder-decoder recurrent neural network with LSTM units and brought attention to generate headlines from the text of news articles. The authors use an encoder-decoder model where the encoder was fed as input to the text of a news article, one word at the time. Each word was first passed through an embedding layer that transforms the word into a distributed representation. The decoder generates, using a softmax layer and the attention mechanism, each of the words of the headline, ending with an end-of-sequence symbol. After generating each word, that same word is fed in as input when generating the next word. [Nallapati *et al.* 2016] described an abstractive text summarization model using attentional encoder-decoder re-

current neural networks. The authors capture keywords using feature-rich encoder, to identify a fundamental idea for the summarization and model rare/unseen words using switching generator-pointer. The authors capture hierarchical document structure with hierarchical attention to generate a summary.

2.3.3 Text Summarization in Indian Languages

The world wide web has provided us with large data. In the current world, the attention span of people is continuously decreasing. Information for quick perusal is becoming a necessity. The efforts to summarize documents in Indian languages have been relatively recent, with [Patel *et al.* 2007] proposing a language agnostic approach to summarization. All the advancement related to Indian language document summarization has been made in the past *10* years.

2.3.3.1 Methodology

Various methodologies have been introduced in the past decade for automatic text summarization in Indian languages. Section 2.3.3.1.1 presents a brief overview of the features that have been repetitively used by the techniques proposed by various authors for Indian languages. Section 2.3.3.1.2 further elaborates the techniques used for Indian language text summarization. These techniques can be broadly divided into score based, machine Learning based or graph based. Scoring based approaches assign scores to the sentences based on features such as sentence length and presence of cue words. Machine learning based approaches decide the importance of sentences based on features such as TF-IDF scores and keywords identification, and determine if the sentence should be present in summary or not. Graph-based approaches convert the source text into a graph and then generate a summary from the graph.

2.3.3.1.1 Features

1. Relative length

   Sentence length determines the importance of the sentences. Sentences of short length are given lower importance as they contain lesser information.

2. Keywords/Title Keywords identification

   Keywords are thematic words containing relevant information. Keywords are identified by calculating TF-ISF (Term Frequency-Inverse Sentence Frequency) score. Keywords are also the words that are present in the headline/title of the document to be summarized. Keywords are topical words containing vital information. Keywords are distinguished by ascertaining TF-ISF score. Keywords can likewise be the words that are available in the feature/title of the document to be condensed.

3. Numeric data identification

   Sentences containing numeric data are given a higher priority since they are considered an essential part of the text.

4. Named Entity Recognition

   NER uses lists containing prefixes, suffixes and proper names. Named entity identification helps in identifying sentences that might contain information about places, or people.

5. Sentence headlines identification

   Headlines contains essential information and are always included in the summary.

6. Proper Nouns Identification

   Proper Nouns imply that a sentence might have relevant information regarding a person or a place.

7. Cue-Phrase Identification

   Words such as 'conclusion', 'finally' and 'hence' in a sentence imply that the sentence will have more weight.

8. Sentence Position

   The position of the sentences in the text might determine the importance of the sentence. Sentences that appear at the beginning or the end have more weight.

9. Presence of URLs

   Sentences with URL(s) are given higher priority since URLs contain essential information that should be present in summary.

10. Term Frequency-Inverse Document Frequency (TF-IDF) weights
    TF-IDF helps in determining the importance of words that occur the most number of times in the document. TF-IDF also helps in demarcating what words occur frequently and are not crucial to the summary.

11. Presence of inverted commas
    Inverted commas usually indicate essential words/phrases in a sentence, since inverted commas are used to place emphasis.

2.3.3.1.2 Techniques used for Summarization

Score Based

Statistical features are used by score based approaches. These features help in finding the relevant sentences and phrases in the sentence. Linguistic knowledge is not required for these techniques, and hence they can be relatively language independent. [Gupta & Kaur 2016] use Features 1-8 for ranking sentences for Punjabi text summarization. [Kallimani *et al.* 2016] use 1 and 10 to rank sentences for summary generation. [Jayashree *et al.* 2011] use 10 and GSS coefficient (add reference) for ranking sentences. [Sankar *et al.* 2011] use 8, 10 and the sentence weight is calculated by using string patterns.

Machine Learning

Machine Learning approaches can be either unsupervised or supervised or semi-supervised. The supervised approach uses annotated data to classify each sentence as to whether it belongs in the summary or not because the absence of annotated data, most approaches for Indian languages are unsupervised. [Sarkar 2012] use the feature 10 and 8, to rank sentences in the input document. Top k sentences are then selected as required. [Bhoir & Gulati 2016] use 1 - 3 and 5 - 8 as features in conjunction with a fuzzy Logic system. [Desai & Shah 2016] use 1 - 3, 8 and 11 as features to SVM model. [Thaokar & Malik 2013] used features 1 - 3 and 8 to assign scores to the sentences. Utilizing genetic algorithm, the best chromosome is chosen after the particular number of generations. At that point utilizing euclidean distance is calculated between the sentence and the fittest chromosome. Sentences are sorted, and contingent upon the compression rate sentences are chosen to create the summary. [Keyan & Srinivasagan 2012] proposed multi-document summarization using neural networks for Tamil and English. The framework includes three stages. In an initial step, the sentences of the documents are changed over into vector shape. In the first step, sentences of the documents are converted into vector form. In the second stage, based on the sentence features, weights are assigned to the vector. After calculating similarity and dissimilarity measures, summaries are produced.

Graph Based Approach

Graph based approaches represent the source text as a graph. Generating summaries involves figuring out the sub-graphs that hold more information and then traversing the graph to includes those sentences into the summary. [Subramaniam & Dalal 2015] present a graph based approach for Hindi text summarization. The authors created Rich Semantic Graph (RSG) of the original document and identified substructures of the graph (using ontological features) that can extract meaningful sentences for generating a document summary. [Ajmal & Haroon 2015] proposed a technique in which sentences in the documents are represented as nodes in an undirected graph. If the cosine similarity for two sentences is above a threshold, they are connected by an edge. Their approach could also perform query based summarization. For query based summarization, only the subgraph which contains sentences related to the query are selected. [Sankar *et al.* 2011] proposed a graph based technique for generating summaries in Tamil. [Banu *et al.* 2007] proposed a summarization approach that builds a semantic graph by discovering subject, object and predicate in the sentence.

2.3.3.2 Text Summarization Systems

2.3.3.2.1 Hindi

[Patel *et al.* 2007] proposed a language agnostic summarization approach for English, Hindi, Gujarati and Urdu documents based on statistical features. The algorithm was applied to DUC dataset for English and news articles were used for the rest of the languages. The language independence of the algorithm was tested by providing news articles for summarization, and in almost every case a degree of representativeness of more than 80% was achieved.

[Thaokar & Malik 2013] used machine learning technique in for Hindi language using a genetic algorithm. [Kumar & Yadav 2015] proposed an extractive summarization algorithm which scores sentences based on the co-occurrence of words which adhere to the theme of the document. The average accuracy of the system was evaluated with a manual summary and was calculated as 0.85.
[Subramaniam & Dalal 2015] generated a semantic graph from the input text. Heuristic rules were applied to the semantic graphs to condense them. These rules exploited the WordNet semantic relations: hypernym, holonym, and entailment. Domain ontology was used to generate summaries as information needed to generate summaries. WordNet is then used to generate multiple summaries based on synonyms. The generated multiple texts are ranked, and the best amongst them is selected for the summary.
[Bhoir & Gulati 2016] used various features. Sentences with a higher number of title words are given more weight then sentences with lesser number of title words. Only sentences with subject, object and verb triple are selected for the next step. Sentences with code mixed text are considered more important and have a higher priority. The sentences are classified using a fuzzy logic classifier and then ranked for the final summary.
[Pundlik *et al.* 2016] use the following features: sentence paragraph position, overall sentence position, numerical data in the sentence, presence of inverted commas, sentence length, keywords in the sentence. SVM is then applied to sentences in the range from 4 to 1, with '4' indicating most important sentence and '1' indicating that sentence is not important. The sentences are then extracted based on their rank. The dataset used contained text from various online news sources such as zeenews, khabarNDTV and Patrika. A total of 130 Hindi news articles from different categories of news namely Bollywood, politics and sports are present. The average number of words in an article is 400-500. The average result of experiments indicated 72% accuracy at 50% compression ratio and 60% accuracy at 25% compression ratio.

2.3.3.2.2 Kannada

[Kallimani *et al.* 2010] proposed a system called 'AutoSum'. The system makes use of key terms, like proper nouns, adjectives and adverbs to determine the importance of the sentence. Each sentence is assigned a score based on the features as mentioned earlier.

The summary is then generated where the output contains the most important sentences in the input text (determined by the score).

[Jayashree *et al.* 2011] proposed a system to generate extractive summaries. Their system extracts Kannada documents available online for keywords. To score the sentences in the input text, statistical features like TF-IDF, Galavotti Sebastiani Simi(GSS)[Galavotti *et al.* 2000] coefficient are used. The sentences once scored, were sorted in descending order by the value of the score. Based on the number of sentences specified by the user, a summary was generated. The summaries were generated based on the number of sentences determined by the user. The manual evaluation was then done to evaluate the summaries. The recall values across the categories sports, entertainment and literature, were 0.76, 0.8 and 0.7 respectively.

2.3.3.2.3 Malayalam

The approach proposed by [Ajmal & Haroon 2015] represents the source text as an undirected graph. In the undirected graph, sentences are represented by nodes. Two nodes are connected by an edge if the cosine similarity between the two sentences is above a threshold. The value of the said threshold is calculated by using the number of paragraphs and the number of sentences in the input document. This approach results in a graph, where there are sub-graphs which are segregated based on the topics that they represent. This allows in creating summaries where one can select sentences from each topic. This way the summary is representative of the whole document. This approach also eases query-based summarization as sentences from the sub-graph pertaining to the query can be selected for the summary. The maximum f-measure reported by authors was 80.18%.

2.3.3.2.4 Tamil

[Banu *et al.* 2007] proposed a summarization approach that builds a semantic graph by discovering subject, object and predicate in the sentence. The text documents are compressed by applying syntactical features. Subject, object and predicate triples are identified for each sentence and a corresponding graph is generated. Normalization is applied on the subject, object, predicate triples to reduce the frequency of the nodes in the graph, thus reducing the size of the graph. SVM algorithm is then used to classify the sentences. The algorithm identifies which sentences belong to the summary.

[Sankar *et al.* 2011] proposed an approach that scores sentences based on features like word frequency, position of terms. Sentence weights are also calculated by using string patterns. Their approach is domain independent and does not require annotated corpus. The authors used ROUGE ([Lin 2004]) scores to evaluate the generated summaries and achieved an average ROUGE score of 0.4723. [Keyan & Srinivasagan 2012] proposed multi-document summarization using neural networks for Tamil and English. The sentence weight value determines if multi-document summarization or single document summarization is to be performed. Their proposed system was able to summarize both new articles in Tamil and English.

2.3.3.2.5 Punjabi

[Vishal & Gurpreet 2012] used the following features: title/keywords (identified by using term frequency-inverse term frequency), NER, numerical data, proper nouns, and cue phrases. Scores for sentences are generated using these features. The features are assigned weights based on their importance. The final score are calculated by the following equation: $w_1.f_1 + w_2.f_2 + ... + w_n.f_n$, where $f_1...f_n$ are different features of sentences and $w_1...w_n$ are the corresponding feature weights of sentences. The sentences are then sorted to generate the final summaries. For news documents in Punjabi, f-measure values ranged from 0.97 to 0.94 depending on the compression rates. For stories, the f-measure value was 0.81 for 10% compression, 0.89 for 30% compression and 0.94 for 50% compression.

[Gupta 2013] proposed an extractive automatic text summarization system. Their approach uses a two step process for summarization. The first step involves pre-processing where stop words are removed, sentence boundaries are identified and word frequencies are calculated. The next step involves assigning weights to the sentences based on the features and then generating summaries based on these scores. The authors tested the proposed system over fifty Punjabi news documents and fifty Punjabi stories. The accuracy of their system varied from 0.81 to 0.92.

2.3.3.2.6 Bengali

[Islam & Masum 2004] proposed a summarization approach that was corpus-oriented for Bengali. Files in the corpus were scored on the premise of highest frequency words by applying vector-space-term-weighting. Their approach builds a document index and performs summarization using vector space retrieval method. Their approach built a tokenizer which tokenized the documents and then performed ranking and summarization. The tokenizer was able to handle the dangling anaphora problem which is the result of a shallow linguistic analysis.

[Das & Bandyopadhyay 2010c] proposed an opinion text summarizer. The sentiments on the topic were calculated from the input text. The sentiment information was then used in the generation of the summary. Their approach determines sentiment by implementing an aggregation model. This model helps in discerning the theme at a discourse level. The aggregation model uses a k-means clustering algorithm. Sentences for the summary were chosen based on the graph representation by applying page rank algorithm. Their approach achieved a precision of 0.72, recall value of 0.67 and f-measure score of 0.69.

[Sarkar 2012] proposed a sentence extraction technique for summarization. The technique proposed could be divided into three stages: preprocessing, sentence scoring and summary generation. In the preprocessing step, stop words were removed, stemming was done. In the next step, sentences were ranked based on the words present and position. Sentences containing words that were related to the document theme and had a TF-IDF score above a certain threshold were given a higher priority. According to the positional significance, the score of sentence is inversely proportional to its position. The sentences are ranked on the basis of their scores and the summary is generated by selecting K-top ranked sentences, where K is specified as the input to the system.

2.3.4 Applications and Challenges

Past research has established that summaries as short as 17% of the text do a good job in speeding up decision making, as much as by a factor of two ([Mani *et al.* 2002]) without any decrease in the accuracy. Summaries that are query based also help in making relevant decisions in a shorter time span. Summaries empower users to discover the

important content in the document without having to go through the entire text of the document.While summarizing scientific documents, the objective is two fold, one to identify relevant documents and second to understand the relationship between the current document and the articles that the document cites. Phone message synopses are useful for perceiving the priority of the message, the number to call back, or the caller and outlines of threads in forums are valuable in determining if the thread is relevant. Another surprising application of summarization involves automatic evaluation of GMAT essays ([Burstein *et al.* 2001]). Summarization improves the topical analysis of an essay significantly. Summarization allows to reduce redundancy in the essays written under time constraints, which in turn allows better assessment of the essay.
The first and most challenging task in automatic summarization is gathering dataset for evaluating the systems. While there has been a radical change in English text summarization in the last 10 years, no breakthrough can be seen for Indian languages. One of the challenges is the lack of efficient resources such as NER taggers and POS taggers. Another factor that is deterrent to the development of state of the art resources is that people who use the internet predominantly use English. While traditional print media might use English languages, electronic media uses English, and the media that does use Indian language has a limited demographic. Summarization for code mixed languages is a field that has not been explored yet. Since the advent of social media platforms such as Twitter and Facebook there has been no dearth of code mixed content, but since code mixing is usually used in an informal setting, summarization for such content has not been explored. The summarization systems have to compete with ever increasing vocabulary as well.

2.4 Spam Detection

Social media has seen increased spamming with the advancement in technology. Many networks have tried to keep stringent policies for the safety of their customers, but spammers adapt suitably as well. In 2008, Twitter officially announced that performance of entire system was threatened because of the Follow spam accounts [1]. [Chakraborty

[1]https://blog.twitter.com/official/en_us/a/2008/making-progress-on-spam.html

et al. 2016b] proposed nine domain-based classifications of spam techniques:

1. E-mail ([Sirivianos *et al.* 2011], [Zisiadis *et al.* 2011])
2. Blog ([Zhu *et al.* 2011], [Kantchelian *et al.* 2012], [Tan *et al.* 2013])
3. Microblog ([Ghosh *et al.* 2012], [Yang *et al.* 2012], [Chu *et al.* 2012], [Hu *et al.* 2013], [Fu *et al.* 2015], [Zheng *et al.* 2016])
4. Bookmarking ([Fakhraei *et al.* 2015], [Poorgholami *et al.* 2013], [Yang & Lee 2011])
5. Social network ([Jin *et al.* 2011], [Ahmed & Abulaish 2012], [Bosma *et al.* 2012], [Tan *et al.* 2012], [Cao *et al.* 2012], [Yang & Lee 2014])
6. Review ([Sun *et al.* 2013], [Mukherjee *et al.* 2013b], [Fei *et al.* 2013], [Sharma & Lin 2013], [Lin *et al.* 2014])
7. Location search ([Aggarwal *et al.* 2013], [Costa *et al.* 2013])
8. Comment based ([Sureka 2011], [Rădulescu *et al.* 2014])
9. Cross-media ([Lumezanu & Feamster 2012], [Wang *et al.* 2011])

Another categorization proposed was based on Twitter by [Jeong *et al.* 2016] on Twitter spam filtering, link spam filtering ([Becchetti *et al.* 2006], [Castillo *et al.* 2007], [Gyöngyi *et al.* 2004],[Krishnan & Raj 2006]), Sybil Detection ([Shi *et al.* 2013], [Gong *et al.* 2014], [Cao *et al.* 2012]) and data mining approach for spam detection ([Fakhraei *et al.* 2015], [Graham 2002], [Hovold 2005], [Kayes *et al.* 2015], [Sculley & Wachman 2007], [Zhou *et al.* 2014]), where Twitter based spam filtering was further categorized as follows:

1. Content-based spam filtering ([Ghosh *et al.* 2012], [Benevenuto *et al.* 2010], [Martinez-Romo & Araujo 2013])
2. Social network based Twitter spam filtering ([Jiang *et al.* 2014], [Stringhini *et al.* 2013])
3. Subnetwork based on spam filtering ([Wang 2010], [O'Callaghan *et al.* 2012], [Akoglu *et al.* 2010])

[Ghosh *et al.* 2012] proposed approach which used link farming property. However, this approach can be burdensome as it needs social network data for the entire network and lead to high computational cost. COMPA [Egele *et al.* 2013] used tweeting lingo, time window and Uniform Resource Locator (URL) to learn the behavioural pattern of each user for identifying spam. [Yardi *et al.* 2009] used behavioural patterns and concluded that trending topics hashtags are an effective way of spamming strategy. [Benevenuto *et al.* 2010] and [Martinez-Romo & Araujo 2013] proposed use of a number of hashtags and URL(s) or spam URL(s). [Gao *et al.* 2014] used a template-based approach for matching spam ground truth tweets. [Jiang *et al.* 2014] analyzed behavioural synchronicity for identifying fake accounts. [Stringhini *et al.* 2013] analyzed the patterns in increase and decrease of number of followers. [Viswanath *et al.* 2014] identified intentional follow and like for anomaly detection approach using Principal Component Analysis (PCA). [O'Callaghan *et al.* 2012] detected comment spammer in YouTube using video user relation network. [Akoglu *et al.* 2010] used discriminating features extracted from weighted sub-graphs from network to detect spammers. BadRank [Wu & Davison 2005] used spam labels to lower the rank of spam web pages.

Existing proposed works also employed supervised learning techniques by using features based on review text, rating and other metadata ([Jindal & Liu 2008], [Ott *et al.* 2011], [Feng *et al.* 2012a], [Mukherjee *et al.* 2013a]). [Ferrara *et al.* 2014] stated that smart spammers are hard to distinguish from legitimate users just via content-based features. [Chu *et al.* 2010] used user profile features and behaviour. A social honeypot is used by [Lee *et al.* 2011] to allure spammers for building benchmark dataset. Few of the works focused on clustering of URL(s) and network graph of spammers ([Wang *et al.* 2015a], [Wang 2010], [Yang *et al.* 2011], [Yang *et al.* 2012]). [Liu *et al.* 2016] proposed an approach based on Latent Dirichlet Allocation (LDA) using local and global information of topic distribution patterns. [Diale *et al.* 2016] optimized the kernel type and kernel parameters for improving the performance of SVM. Authors also varied a number of features for SVM, AdaBoost and random forest to check the effect of their performance. [Mi *et al.* 2015] applied stacked autoencoder for identifying spam detection.

[Jeong *et al.* 2016] deals with *follow spam* on Twitter. Authors proposed classification schemes focusing on cascaded social relations for identifying spammers and devised two schemes, TSP-Filtering and SS-Filtering, each of which utilizes Triad Significance Profile (TSP) and Social status (SS) in a two-hop sub-network centred at each other. An ensemble technique was also proposed, cascaded-filtering, which combine both TSP and SS properties. The proposed schemes are scalable because instead of analyzing the whole network, they inspect user-centered two hop social networks. [Kim *et al.* 2016] proposed two ways of spam detection, by comparing the similarity between user comments and publisher posts; and by learning single representative meta feature such as username or ID. [Shao *et al.* 2017] proposed a hybrid method based on image and text spam recognition. In case of an image, local and global image features are used whereas, in case of text semantic properties are used. [Li *et al.* 2017c] proposed neural network based model to learn representation of reviews. Authors compute sentence importance and incorporate them to represent document representation. [Song *et al.* 2017] proposed methodology exploits a probabilistic generative model for mining the latent semantics from user-generated comments and an incremental learning approach for tackling the changing feature space. [Li *et al.* 2017a] discovered that reviewer's posting rates (number of reviews written in a period of time) follow an interesting distribution pattern. Their posting rates are bi-modal. Multiple spammers also tend to collectively and actively post reviews to the same set of products within a short time frame, which they called co-bursting.

2.5 Sarcasm Detection

Earliest work on sarcasm was done by [Tepperman *et al.* 2006] which deals with sarcasm detection in speech. Since then there has been an increase in research of automatic sarcasm detection. With the expansion of communication platforms over different social media, sarcasm has been expressed in various forms (tweets, reviews and dialogues) and a variety of approaches has been explored for the same including rule-based, supervised and semi-supervised. [Riloff *et al.* 2013], [Maynard & Greenwood 2014] and [Mishra *et al.* 2017] used manual annotation for sarcastic tweets whereas some others such as [Tsur *et al.* 2010], [González-Ibánez *et al.* 2011], [Bharti *et al.* 2015] and [Bamman & Smith 2015] used hashtag based tweets for sarcasm identifications. [Wallace *et al.* 2014] and [Wallace *et al.* 2015] worked on Reddit comments. Few researchers have also focused on long text or reviews such as [Buschmeier *et al.* 2014], [Filatova 2012], [Reyes & Rosso 2014] and [Reyes & Rosso 2012]

In 2010, [Tsur *et al.* 2010] worked on hashtag based annotations. They used a semi-supervised method for sarcasm identification on a Twitter dataset (5.9 million tweets) and 66,000 product reviews from Amazon. They worked on standalone sentences and did not consider the context of data as these were standalone sentences. Pattern based features were extracted to train the model for retrieving sarcastic sentences. [Buschmeier *et al.* 2014] used amazon corpus collected by [Filatova 2012] as its dataset for training. A lot of feature engineering was performed for the machine learning techniques. The best performance of the system was with input as combination of star-rating feature, bag of words and specific features. [González-Ibánez *et al.* 2011] worked on hashtag based data which was divided into three categories (sarcastic, positive sentiment and negative sentiment), each containing 900 tweets. Various combinations of uni-grams, dictionary-based features and pragmatic factors (positive and negative emoticons and user references) features were used in conjunction with two classifiers, SVM with Sequential Minimal Optimization (SMO) and logistic regression. Work done on Amazon data collected using crowdsourcing in [Filatova 2012] was used by [Buschmeier *et al.* 2014]. They formulated the problem as a supervised classification task and evaluated different classifiers on a combination of lexical and pragmatic features. They reached an f-measure of 74 % using logistic regression. They made use of features which were proposed in previous research works and tried various permutations and combinations of features to train the classifier. [Bamman & Smith 2015] used an evenly balanced corpus of positive and negative tweets with 9,767 sarcastic and 9,767 non-sarcastic tweets respectively, totalling to corpus size of 19,534 tweets. As for the labelling of the corpus, #sarcastic or #sarcasm tagged tweets were marked for sarcasm and not otherwise. The classification is done using binary logistic regression methodology with regularization. Two categories of features were engineered, tweet based and author based features. Major tweet features were brown cluster uni-grams and bi-grams, part of speech features, pronunciation features, tweet (whole/word) sentiment (via Stanford sentiment analyzer [Socher *et al.* 2013]) whereas author features were profile information, historical topics, historical sentiment and historical salient terms. A brilliant accuracy of 85% was achieved for all the features as the input to the model.

[Cliche 2014] also worked on Twitter dataset and most important features that came as a result of his analysis were n-grams, sentiments and topics. They used the python library genism which implements topic modeling using Latent Dirichlet Allocation (LDA). Initially, topic was learned by feeding all the tweets to the topic modeller. Then they decomposed each tweet as a sum of topics, which was used as features.[1]. [Maynard & Greenwood 2014] studied sarcastic tweets and their impact to sarcasm classification. Their experiment used around 600 tweets which are marked for subjectivity, sentiment and sarcasm. They proposed that hashtag sentiment is a key indicator of sarcasm. Hashtags are often used by tweet authors to highlight sarcasm, and hence, if the sentiment expressed by a hashtag does not agree with rest of the tweet, the tweet is predicted as sarcastic.

[Bharti *et al.* 2015] present two rule-based classifiers. The first uses a parse-based lexicon generation algorithm that creates parse trees of sentences and identifies situation phrases that bear sentiment. If a negative phrase occurs in a positive sentence, it is predicted as sarcastic. The second algorithm aims to capture hyperboles by using interjection and intensifiers occur together. They describe a rule-based approach that predicts a sentence as sarcastic if a negative phrase occurs in a positive sentence. Another rule-based classifier was proposed by [Riloff *et al.* 2013] that look for a positive verb and a negative situation phrase in a sentence. The set of negative situation phrases are extracted using a well-structured, iterative algorithm that begins with a bootstrapped set of positive verbs and iteratively expands both the sets (positive verbs and negative situation phrases). They experiment with different configurations of rules such as restricting the order of the verb and situation phrase. They use a set of patterns, specifically positive verbs and negative situation phrases, as features for a classifier (in addition to a rule-based classifier). [Wang *et al.* 2015c] use SVM-HMM to incorporate sequence nature of output labels in a conversation. [Liu *et al.* 2014] compare several classification approaches including bagging and boosting, show results on five data sets. On the contrary, [Joshi *et al.* 2016b] experimentally validate that for conversational data, sequence labelling algorithms perform better than classification algorithms.

Few works have also used deep learning techniques. [Joshi *et al.* 2016c] use similarity between word embeddings as features for sarcasm detection. They augment features based

[1]https://github.com/MathieuCliche/Sarcasm_detector

on similarity of word embeddings related to most congruent and in-congruent word pairs and report an improvement in performance. The augmentation is key because they observe that using these features alone does not suffice. [Amir *et al.* 2016] present a novel CNN based that learns user embeddings in addition to utterance-based embeddings to learn user-specific context. [Ghosh & Veale 2016] use a combination of CNN, LSTM followed by a deep neural network. One of the latest research in this area, inspecting on an entirely new approach based on the psycho-linguistic side of sarcasm detection, using cognitive features extracted with the help of eye-tracking, is by [Mishra *et al.* 2017]. The basis is built on the fact that sarcasm often springs from incongruity which forces the brain to reanalyze it, as a consequence of which eye movement differs upon reading the text, i.e. distinctive eye-movement patterns may be observed in case of successful processing of sarcasm in the text in contrast to literal texts. The task requires the availability of affordable eye-trackers which is supported by developments such as Cogisen patent on "eye-tracking using inexpensive mobile web-cams". Gaze-based features were developed based on eye scan paths of participants, which were calculated using graph structure - "saliency graphs". Combined with other lexical and textual features, the model based on multi-instance logistic regression resulted in an improved average precision and accuracy of 76.5% and 75.3% respectively. An analysis over the results shows that gaze based features alone achieve ample performance superseding the combination of features in some instances. This is amongst first of works which use cognitive features for NLP task.

2.6 Code Mixing

Earlier code switching used to be considered as a substandard language even on informal platforms, but with the technological advancements, people started accepting it as a natural part of multilingual language use. Few studies have been performed for analyzing the reasons which lead to code mixing. Some studies such as [Li 2000], [San 2009] suggested that code mixing occurs due to the linguistic motivations whereas [Sotillo 2012], [Bock 2013], [Shafie & Nayan 2013] and [Goldbarg 2009], suggested that it occurred due to the social motivations to mark membership in-group in short text messages, chat messages, Facebook comments and emails respectively. Most of the text processing tools have been constrained to the English Language. With increasing trends of code mix text usage all over the globe, it has become important to develop processing tools for such text. [Fischer 2011] provided insight on language usage in different parts of the world on Twitter. A trend was observed that usually the language occurring in social media is code mixed when at small distances there are large languages present. [Dewaele 2010] claimed that "strong emotion arousal" increases the frequency of code mixing in the text.

Recent works found out a trend of code-mixed texts in Indian languages. [Gupta *et al.* 2014] formally introduced the concept of Mixed Script Information Retrieval (MSIR) and challenges associated with it. Authors also bridged the gaps by analyzing the web traffic for MSIR through Bing query logs and thereby deducing the impact of mixed scripting. Their proposed solution models the recondite representation of terms transverse different languages through deep learning architecture which ultimately results in equivalent terms to be closer to each other.

In 2015, MSIR finally attained some attention when two shared tasks were organized. In shared task organized for the Sentiment Analysis in Indian Languages (SAIL) [Patra *et al.* 2015] three languages were considered, i.e. Bengali, Hindi and Tamil but the data provided was language specific, and no code mixing was involved. Task had two scenarios in which participants had to build constrained and unconstrained systems. Constrained systems were restricted to the data provided, but in unconstrained ones, teams used various external sources for different languages. It was observed by task organizers that accuracy of unconstrained systems decreased significantly. Another shared task, MSIR [Sequiera *et al.* 2015b] was organized in which participants had to identify *9* different languages, named entities, punctuations and mix scripts in the text for which significant results were obtained. Task organizers found that most confusing language pair was that of Hindi and Gujarati. Also, based on results obtained from participating teams they concluded that performance of the system for each of the category included in the task was correlated to the number of tokens used for that category. Code mixing has now found its application in different areas such as query Labeling [Bhargava *et al.* 2015], sentiment analysis [Bhargava *et al.* 2016d], question classification and various tools are being developed for the same such as POS tagger, language identifier and name entity recognizer.

2.6.1 POS Tagging

In the past decade, there has been much increase in code mixed text. At present, there are parts of speech taggers for English with an accuracy of *97.64%* [Choi 2016] using dynamic feature induction and tested on wall street journal dataset. On the other hand, there are very few systems present for POS tagging in Indian languages which are less accurate when compared to English POS taggers. This can be attributed to the lower availability of resources available publicly for Indian languages. According to [Antony & Soman 2011] work in POS tagging for Indian languages was mainly based on rule-based approaches in the past. In Hindi, [Singh *et al.* 2006] designed a POS tagger based on exhaustive morphological analysis backed by high-coverage lexicon and a decision tree based learning algorithm. This system has achieved an average accuracy of *93.45%*. In Bengali, [Ekbal *et al.* 2009] has applied voted approach on three classifiers SVM, CRF and Maximum Entropy to gain a tag precision of *92.35%*. [RamaSree & Kusuma Kumari 2007] have designed a rule-based POS Tagger for Telugu corpus and have claimed to have achieved an accuracy of *98.016%*. There are POS taggers for Tamil, Marathi, Punjabi, Malayalam. POS tagging for code mixed data is a relatively new field of research and difficult because of the absence of tagged code mixed data.

[Solorio & Liu 2008] presented the results on the problem of POS tagging English-Spanish code-switched discourse by using preexisting taggers for both languages. They worked on different POS taggers for monolingual languages and used them to predict POS for the given the word. However, they did not face the problems of text generated through social media, and hence the accuracy achieved by them was quitesss high. The initial attempt to tag code mixed texts was made by [Jamatia & Das 2014]. An approach to annotate the dataset using CRF with *5* fold cross validation along with manual correction is discussed. The dataset is then trained using various machine learning classifiers such as SVM, Random Forests and Naive Bayes classifier. The random forest achieved the best results with *82%* in fine-grained and *83%* in coarse-grained data. With the advent of social media, people are encouraged enough to work on POS tagging of mixed languages. One such attempt was made by [Vyas *et al.* 2014]. His best effort is recognized as the introduction of transliteration problem in POS tagging of mixed languages. Their work mainly includes formalization of the problems and challenges experienced in Hindi-English POS tagging. First, POS tagging is done by assuming the knowledge of language information and normalized/transliterated form of the word. This experiment gave an idea of the possible POS tagging accuracy if all the above information can be found out correctly. The experiment uses two English taggers (Stanford tagger [Toutanova *et al.* 2003] and CMU ark tweet Tagger [Owoputi *et al.* 2013]), and both the taggers give an accuracy of *79.02%*. The next experiment assumed only the knowledge of language tag and the other modules such as back transliteration module and POS Tagger were developed resulting in an accuracy of *74.87%*. The third experiment assumed the absence of all information except the word. In this case, all the modules were developed. The third experiment gave a final accuracy of *65.39%*.

Nowadays, people are mixing heuristic operations to improve the performance of POS taggers of mixed languages further. A language identification system had been developed by [Gella *et al.* 2013] which identified the language of the words and then, used a simple heuristic to form chunks of the same language. CRF++ was then used to identify POS Tags. Most of the works done till now have proved the essence of language identification along with transliteration in the POS tagging of mixed languages. [Jamatia *et al.* 2015] collected and annotated the data for Code Mix English-Hindi Twitter and Facebook chat messages and compared the language-specific taggers to that of machine learning taggers using features based on word-level information. They have drawn a comparison between different machine learning techniques such as CRF, minimal sequential optimization, naive bayes and random forests using different features such as word context and other word-based features. The best performing system was of CRF and RF with weighted accuracies of *71.6%* and *70.6%* respectively.

[Gambäck & Das 2016] discussed the challenges posed in the evaluation of code mix data and proposed a formal measure for evaluation. [Wang *et al.* 2015b] use BLSTM-RNN for POS tagging. The Recurrent Neural Network (RNN) is word embedded. The authors achieved an accuracy of *97.40%* when tested on the Penn treebank wall street journal test set. Their approach performed comparably to Stanford POS tagger without using any language features like morphemes, root words, affixes, intonations and stresses. Given a sentence w_1, w_2, ..., w_n with tags y_1, y_2, ..., y_n, a bidirectional LSTM network was used to predict the tag probability distribution of each word. The output layer was a softmax layer whose dimension was the number of tag types. The output was the tag probability distribution of input word w_i. All weights were trained using back-propagation and gradient descent algorithm to maximize the likelihood of training data.

[Vaswani *et al.* 2016] used feed-forward neural networks with two hidden layers of rectified linear units for both POS tagging and feed-forward networks. In POS tagging, when tagging word w_i, the authors considered only features from a window of five words, with $\mathrm{w_i}$ at the centre. For each word in the window, the authors added the word lowercased and a string that encodes the basic "word shape" of the word in the window. This was computed by replacing all sequences of uppercase letters with *'A'*, all sequences of lowercase letters with *'a'*, all sequences of digits with *'9'*, and all sequences of other characters with *'*'*. Finally, two and three letter suffixes and two letter prefix were added for w_i only.

[Plank *et al.* 2016] described a basic bidirectional LSTM tagging model is a context bidirectional LSTM taking as input word embeddings. The authors incorporated sub-token information using a hierarchical bidirectional LSTM architecture. The sub-token level (either characters or uni-code byte) embeddings of words are computed using a sequence of bidirectional LSTM at the lower level. [Sequiera *et al.* 2015a] proposed an extension to the existing approaches by introducing a new feature set for transliteration. The system is tested on naive bayes and maximum entropy approaches and obtained the best accuracy of *84%*. The paper shows that integration of language detection and POS tagging systems does not improve the accuracy in overall.

Most of the research work was done on the ICON data sets 2015 onwards. [Sarkar 2016b] have designed a tri-gram based Hidden Markov Model (HMM) for POS tagging. The system first tokenized the sentences into tokens and attaches meta tags and broad POS tag information to each token. The meta tag contained the word characteristics by which it stood differently. The system is later tagged with the tri-gram tagger with the above features for three different language pairs Bengali-English, Hindi-English and Telugu-English. [Pimpale & Patel 2016] compared various machine learning algorithms such as decision tree, random forest, naive bayes and Multilayer Perceptron (MLP) for developing POS tagger. The system uses the language of the word, the language of the previous word, the language of the next word, POS tags of the previous two word, POS tags of the next two words similar words and position of the word in the sentence as features for training the system.

[Sharma *et al.* 2016] proposed a CRF based POS tagger. The system used context window words, capital letter based features, affix based features and length of the word as features to train a CRF classifier in constrained mode and used Hindi transliterator, normalizer, Twitter POS tag, English word clusters and affixes of normalized Hindi words as features in unconstrained mode. [Nelakuditi *et al.* 2016] designed a POS tagger for Telugu-English code mixed texts using two approaches. The first approach consists of machine learning approach with lexical features like prefix, suffix, infix features and presence of postpositions(PSP) word, prefix and suffix of adjacent elements and length of the word. Linear SVM(s), CRF(s) and multinomial naive bayes classifier are the machine learning models considered. The second approach consists of having two POS taggers of English and Telugu combined with a universal language tag set. A language identification system was trained on lexical features with CRF and is used to identify the words.

[Ghosh *et al.* 2016] have also designed a system on the same ICON 2015 dataset. The system uses a chunker and a lexicon dictionary for preprocessing sentences. The system then extracts features such as words in a context window of size 5, prefix, suffix and some binary features. The system uses some post-processing rules to identify emoticons, universal tags and other unidentified words.

In 2016, ICON has conducted another POS tagging sub-task and released the following dataset. [Gupta *et al.* 2017] designed a POS tagger using a hybrid approach. The system uses a rule-based tagger to identify some of the tags. The rest of the tags are identified using a CRF model. The features used include character n-gram, word normalization, prefix and suffix, word class features, word probability, stemming, phonetic normalization and some binary features. Meanwhile, [Patel *et al.* 2016] proposed deep learning approaches to solve the POS tagging problem. Authors compared simple RNN, LSTM, GRU (gated recurrent unit) a deep LSTM networks using context words embedding as the input features. The system uses ICON 2016 dataset for training. The system performance is comparable to Stanford tagger [Toutanova *et al.* 2003] and Hunpos tri-gram tagger [Halácsy *et al.* 2007]. [Sarkar 2016a] have designed a CRF model for POS tagging using the ICON 2016 dataset. The system used language features, orthographic, punctuation features, word context features and binary features for training on the CRF model. Finally, [Barman *et al.* 2016] have come up with three approaches which work on Hindi-Bengali-English code mixed texts. They have designed a pipelined system using the methods implemented by [Solorio & Liu 2008] and [Vyas *et al.* 2014]. They implemented the same systems using a stacked approach instead of a pipeline. In stacked approach the output of all the taggers, language identifiers are combined whereas in pipelined approach the POS tag of a specific language tag is done based on the output of language identifier. The results of stacked came to better than the pipelined features. The combined approach uses a SVM directly on the handcrafted features plus stacked features.

2.6.2 Named Entity Recognizer

Information extraction and natural language processing is a field that is widely researched upon from time to time in the English language as well as other native Indian languages. Mainly, named entity recognition had significant research done so far in English and multilingual corpuses. Initial attempts to identify named entities in multilingual corpuses was made in shared tasks of CoNLL 2002-03 ([Sang 2002], [Tjong Kim Sang & De Meulder 2003]). The best systems of that conference were based on machine learning techniques like hidden markov models and maximum entropy models. The CoNLL 2002 [Sang 2002] was based on Spanish and Dutch language data sets while CoNLL 2003 [Tjong Kim Sang & De Meulder 2003] was based on English and German data sets. In 2013, [Al-Rfou *et al.* 2013] have designed a system called POLYGLOT for massive multilingual NLP applications. This system can handle *40* significant languages including some Indian languages like Bengali, Hindi, Marathi, Punjabi, Tamil and Telugu for NER classification. The NER system was developed based on a massive Wikipedia database and applied semi-supervised approach for classification. In the context of Indian languages, a hybrid maximum entropy model for Hindi and Bengali languages was designed by [Saha *et al.* 2008]. Authors used gazetteer lists and language specific rules in the system. For Tamil language, NER system based on CRF was designed by [Malarkodi *et al.* 2012]. When it comes to NER systems for code-mixed texts in Indian Languages, initial attempts were made by ESM-IL sub-task of FIRE 2015 [Rao *et al.* 2015]. The task consisted of identifying named entities in Hindi and English, social media text. CRF baseline system was built, and other systems were compared with respective to that system. They observed that most of the systems designed had similar precision, but most of them have improved the recall making them better systems. [Pallavi *et al.* 2015] proposed a system by creating a set of features and training them using CRF identifying named entities. A system based on SVM classifier using brown clusters along with POS tags and clusters for identifying named entities has been developed by [Se *et al.* 2015]. [Sarkar 2015] had used POS tag as a state and developed a system using hidden markov model classifier for identifying named entities. Later on in CMEE-IL, FIRE 2016 [Rao & Devi 2016] the named entity recognition was reintroduced for Hindi-English and Tamil-English code mixed sentences. [Bhat *et al.* 2016] have introduced a neural network for identifying the entities. They have used contextual features like prefix and suffix of words in context window along with some binary features. [Gupta *et al.* 2016a] have used a CRF trained on context features like prefix, suffix and character *n*-grams of context window words along with other binary features.

2.6.3 Question Classification

The first phase of question answering system involves question classification which helps in reducing the scope of the answer. The machine learning techniques that have been proposed in the past use different semantic, lexical and syntactic features to classify the questions. Recent works on code mixed question classification include machine learning based approaches for question classification. [Bhargava *et al.* 2016b] proposed a technique that uses NER and removes the named entities in the text. They had used three different classifiers for three different runs: gaussian naive bayes classifier, logistic regression classifier, random forest classifier with the random state as *1*. The authors achieved the highest accuracy of 81.12% using gaussian naive bayes approach among all the three runs submitted.

In the past monolingual question classification has been addressed by [Huang *et al.* 2008] who proposed two approaches using SVM classifier and Maximum Entropy Model classifier. The features used by the authors were 'wh' words like 'who', 'what', 'why', head words, semantic feature of hypernyms, *n*-grams and shape of the word such as all upper case, numbers and all lower case. [Kim 2014] suggests a simple CNN model with hyperparameter tuning and static vectors. The proposed model achieves good results on multiple benchmarks. The author proposed learning of task specific vectors which led to increase in performance. CNN model effectiveness had been proven for NLP and produces good results in semantic parsing ([Yih *et al.* 2014]), search query retrieval ([Shen *et al.* 2014]) and sentence modelling ([Kalchbrenner & Blunsom 2013]). [Zhang *et al.* 2015] proposed character level convolutional networks for text classification. The authors showed that the convolutional networks did not require any knowledge of the words, the syntactic structure and semantic structure of the language.

For question analysis of such data, Question classification is done to understand the question that allows determining some constraints the question imposes on a possible answer. [Zhang & Lee 2003] used bag of words and bag of *n*-grams as features and applied *k*-NN, SVM, naive bayes to automate question classification and concluded that with surface text features the SVM outperforms the other classifiers. [Banerjee *et al.* 2016b] proposed a Question Answering (QA) system which takes cross-script (non-native) code-mixed questions and provides a list of information response to automate the question answering. Corpus acquisition was done from social media, question acquisition using a cloud based service without getting bias, corpus annotations and an evaluation scheme suitable to the corpus annotation. [Li & Roth 2002] proposed question classification using the role of semantic information developing a hierarchical classifier guided by a layered semantic hierarchy of answer types.

[M & Soman 2016] proposed two models (run-1 & 2) where, run-1 used bag-of-words (BoW) model. The run-2 was based on the recurrent neural network. The initial embedding vector was given to RNN, and the output of RNN was fed to logistic regression for training. Overall, the BoW model outperformed the RNN model by almost 7% on F-measure. [Ganesh *et al.* 2016] approached the problem using Vector Space Model (VSM). Weighted term based on the context was applied to overcome the shortcomings of VSM. The proposed approach achieved up to 80% accuracy in terms of F-measure. [Saini 2016b] used term frequency and inverse document frequency (TF-IDF) vector as a feature. A number of machine learning algorithms, namely SVM, Logistic Regression

(LR), Random Forest (RF) and gradient boosting were applied using grid Search to come up with the best parameters and model. The RF model performed the best among all the three variants. [Bhattacharjee & Bhattacharya 2016] proposed a model based on machine learning approaches. They trained three separate classifiers namely RF, one-vs-rest and *k*-NN, followed by building an ensemble classifier using these three classifiers for the classification task. The ensemble classifier took the output label from each of the individual classifiers and selected the majority label as output. In case of a tie, any one label was chosen at random as output. [Majumder & Pakray 2016] proposed three models (runs) out of which two were rule based - the first set of direct rules were applied for the run-1 while the second set of dependent rules were used for the run-3. A total of *39* rules were identified for the rule based runs. Naive bayes classifier was used in run-2 whereas naive bayes updateable classifier was used in run-3.

2.6.4 Sentiment Analysis in Code Mixed Text

In Indian languages, very less research has been conducted for code mixed sentiment analysis, until recently. For the Hindi language, [Joshi *et al.* 2010] were initial contributors. They introduced the concept of SentiWordNet to Hindi. They have proposed three approaches to solve the problem one using SentiWordNet, another using machine translation and the third one using machine learning techniques. The machine learning technique proved to be effective. In the case of Bengali, [Das & Bandyopadhyay 2009] have designed a CRF model based on SentiWordNet and other features. Similarly, there had been few research done in past few years which focused on languages such as Hindi [Sharma *et al.* 2015b], Hindi and Marathi [Balamurali 2012], Marathi [Sapkal & Shrawankar 2016], Tamil [Kumar *et al.* 2015c], Kannada [Kumar *et al.* 2015b]. Users expressing views on social media tend to introduce their native language words into English intentionally or unintentionally. This results in code mixing and makes it difficult for existing tools to analyze the text. For a system to understand the sentiment of a sentence, it must first be able to identify to which language does the utterance belong and hence, language identification becomes an integral part of code mixed text processing system. The problem of language identification is a subcategory of text classification. There has been a lot of

research done in this area. In English, *n*-grams have been proven to be a very effective method for text classification. Initial attempts to identify languages using *n*-grams was proposed by [Cavnar *et al.* 1994]. [Ceylan & Kim 2009] have also used *n*-grams in their Language identifier for search queries. The other widely used approach is the usage of the dictionary to predict the language of a sentence. This method of tagging words and sentences based on the presence of the word in the dictionary of most common words was initially proposed by [Ingle 1976]. Later on [Řehůřek & Kolkus 2009] have done some extensive work on the dictionary based approach for language identification but as pointed out by [Grefenstette 1995] the dictionary method showed the better result for sentences having more than ten words as compared to the n-gram approach. MSIR, FIRE 2015 [Sequiera *et al.* 2015b] had proposed the similar task of language identification. It dealt with word level language labelling in the code-mixed dataset comprising of words belonging to eight different Indian languages. The dataset comprised of words written in English or words transliterated from an Indian language. The task was to identify the language of each word in the dataset. Nine teams provided submissions for the sub-task 1 using supervised machine learning algorithms for language identification. Many of the submissions involved the use of character *n*-grams for feature extraction including [Kumar *et al.* 2015d] and [Bhargava *et al.* 2015]. Apart from these techniques, word2vec and clustering using k-means were also used for the task in hand [Jain 2015]. Different combinations of machine learning algorithms like SVM, naive bayes classifiers, logistic regression, CRF and random forests were used by different teams for supervised learning, out of which [Ethiraj *et al.* 2015] performed best using character *n*-grams, token features in combination with the dictionary based approach, along with using rule based and naive bayes classifier.
Sentiment analysis has attained a tremendous speed in recent times all over the world. With India being a multilingual nation, sentiment analysis also gained its speed for Indian languages. One such work is done by [Joshi *et al.* 2010]. They proposed three approaches for performing sentiment analysis in Hindi. In the first one, they are using Hindi training corpus for classifying sentiments. In the second one, they used machine translation whereas for the third one they build a Hindi SentiWordNet [Joshi *et al.* 2010] dictionary of their work. They concluded that first technique outperforms the other two. Since then a

variety of different algorithm have been proposed in different languages such as [Sharma *et al.* 2015b] in Hindi, [Balamurali 2012] in Hindi and Marathi, [Sapkal & Shrawankar 2016] in Marathi, [Kumar *et al.* 2015c] in Tamil, [Kumar *et al.* 2015b] in Kannada and many more for sentiment analysis.

[Gupta *et al.* 2014] in 2014 formally introduced the concept of MSIR and challenges associated with it. They also bridged the gaps by analyzing the web traffic for MSIR through Bing query logs and thereby deducing the impact of mixed scripting. Their proposed solution, models the recondite representation of terms transverse different languages through deep learning architecture which ultimately results equivalent terms to be closer to each other. [Gella *et al.* 2014] proposed a word level language identification technique which is a necessary and challenging step in code mixing. [Balahur *et al.* 2014] specified the fact that using multilingual data in combination of data obtained via machine translation, helps in improving sentiment classification. In a similar line of work, [Mihalcea *et al.* 2007] proposed several methods to leverage resources and tools available in English by using cross lingual projections.

[Vilares *et al.* 2015] performed sentiment analysis for English and Spanish in variant environments of monolingual, multilingual and code switching. [Das & Bandyopadhyay 2009] proposed a technique in which for language identification, words are matched directly with the dictionaries of each language transliterated into English and then for those who do not match, a set of probable words were taken from all the dictionaries taking words that are closest to the given spelling using the levenshtein algorithm. Then doublet and triplet words are considered for evaluating the probability. Final probabilities decide the relevant match.

[Das & Bandyopadhyay 2009] proposed a computational technique of generating an equivalent SentiWordNet for Bengali from publicly available English sentiment lexicons and English-Bengali bilingual dictionary. They developed SentiWordNet for a variety of Indian languages [Das & Bandyopadhyay 2010b]: Hindi, Bengali and Telugu. Further, they proposed Psycho SentiWordNet [Das & Bandyopadhyay 2011] for *56* languages. In continuation of their effort for improving SentiWordNet dictionaries, they proposed Sentimantics [Das & Gambäck 2012], where they incorporated contextual information. With this work, they enlightened the necessity of using dynamic prior polarity with context.

[Sharma *et al.* 2015a] proposed an approach comprised of two phases, one of them being language identification and the second one being sentiment analysis. They segregated Hindi and English words and calculated final sentiment score by lexicon look up in respective sentiment dictionaries. [Prabhu *et al.* 2016] learned sub word level representations in LSTM architecture instead of character level and word level representations. [Vilares *et al.* 2017] created the corpus for multilingual sentiment analysis on Twitter for English and Spanish. They proposed a multilingual model trained on fused monolingual corpuses by information fusion techniques to sentiment analysis described by [Balazs & Velásquez 2016]. They also proposed a dual monolingual model and a pipeline for determining the language of the unseen text.

Chapter 3

Sentiment Analysis

With the growing amount of data generated from the social networking sites, e-commerce services, blogs, review sites, it has become essential to gain better inisghts of the public opinion, market sentiment and consumer behaviour. This review data acts as dynamic feedback for improvement of the services/products and target potential customers. The emergence of text mining tools with growing data has played an important role in extracting valuable information. Sentiments refer to the emotions of a person. There has been a surge in the number of people expressing themselves through tweets in different languages.

Tweets have been used to give feedback in fields such as critical political issues, new movie releases and sports. These areas need public input to work on the problems deduced from reviews. It is necessary to know how many people are in favour (positive review) or against (negative review) the topic of discussion. Thus, analysis of tweets concerning the sentiments they reflect is of significant scope. According to Google in collaboration with KPMG [1], most of the people in India prefer tweeting in their regional languages, and hence, sentiment analysis of data in Indian languages has become significant. Sentiment analysis uses statistics, Natural Language Processing (NLP) and machine learning techniques to predict the polarity of a sentence and gauge the correctness of the sentiment deduced. There has been much research on English tweets but not on the tweets in Indian languages. One popular research area amongst the researchers is of extracting valuable

[1]https://assets.kpmg.com/content/dam/kpmg/in/pdf/2017/04/Indian-languages-Defining-Indias-Internet.pdf

information from highly dynamic data generated from social media. Temporal opinion mining is the computational process of collecting and analysing the reviews and opinions to determine the intensity and polarity of the expression/view/feeling/assessment concerning time. This field has received a profound interest in recent times.
Two challenges of sentiment analysis are being targeted in this chapter. Section 3.1 discusses the approach to handle sentiment analysis in Indian languages using hybrid neural network architecture. Section 3.2 presents the approach to identify changing sentiments and aspects over time using deep neural networks.

3.1 Neural Network based Architecture for Sentiment Analysis

The sentiments of monolingual tweets in one of the Indian languages (Hindi, Bengali and Tamil) is predicted in this section. The problem targeted is considered as a binary classification problem with the output being a label: *0* for negative and *1* for positive sentiment. Traditional machine learning techniques extract features from a sentence to realise its polarity, but they do not consider the meaning attached to each word of a sentence [Sun *et al.* 2016]. Word embeddings have been used to capture the semantic information of the text to overcome the drawbacks faced by traditional machine learning techniques [Xun *et al.* 2017].
Word embeddings and neural networks have been used in conjunction to identify sentiment of a tweet in the proposed work. The significant contribution of the proposed approach is the usage of all possible combinations (up to three hidden layers) of the three neural network layers (RNN [Elman 1990], CNN [Kim 2014] and LSTM [Hochreiter & Schmidhuber 1997]) and analysing the results in detail. Thirty-nine hybrid models are proposed for each of the three Indian languages, which are based on vector space. All the sequential models are tested using uniform parameters for the layers.

3.1.1 Neural Network based Architecture for Sentiment Analysis in Indian Languages

The proposed approach is a word-level binary classification approach which uses the vector-space model to predict sentiment of a tweet. The procedure is divided into three phases: pre-processing, the creation of sequential model and predictions. These stages convert the textual data to a label which determines the sentiment of a tweet.

3.1.1.1 *Phase I:* Pre-Processing

Given a set of L sentences, $S = \{S_1, S_2, ..., S_L\}$, each sentence S_i, with n words $\{w_1, w_2, .., w_n\}$, $n \in \mathbb{R}$, where $\mathbb{R}$ denotes a set of real numbers, is broken down into its constituent words, and each word is tokenised, thus creating a dictionary of words $w_t = \{`w_1', `w_2', ..., `w_n'\}$. Each word or token is assigned an index on frequency basis of its occurrence in the text file. The indexes are concatenated to return a numerical array for each sentence. The sentences in the form of numerical arrays are padded to yield sentences of similar dimensions, filling the extra columns added to equalise the dimensions with *0*, and returning a padded array, w_p.

3.1.1.2 *Phase II:* Sequential Model

The padded numerical array, w_p, of each sentence, is used as input to the Sequential model, a linear stack of layers. Each index in the array, w_p, is mapped to real-valued vectors, $\mathrm{v_i} \in \mathbb{R}^{\mathrm{d}}$, where v_i is a d-dimensional vector in high-dimensional space using the embedding layer. The Embedding layer is initialized with random weights and will learn an embedding for all of the words in the training dataset. Mapping of indexes produce a matrix of feature vectors (word embeddings) V (represented in equation 3.1) which captures the semantic and syntactic information about the words.

$$V = [v_1, v_2, .., v_{n+k}] \tag{3.1}$$

$$= \begin{bmatrix} v_1^1 & v_1^2 & \cdots & v_1^{n+k} \\ v_2^1 & v_2^2 & \cdots & v_2^{n+k} \\ \vdots & \vdots & \vdots & \vdots \\ v_L^1 & v_l^2 & \cdots & v_L^{n+k} \end{bmatrix} \tag{3.2}$$

where, each row represents vectors of a sentence

In equation 3.2, k represents the padding done in the particular instance. The output of the embedding layer, the word embeddings, is input to further layers in the model. The *39* different hybrid models contains selected combination of 1,2,3 layer network comprising of RNN, LSTM and CNN.

3.1.1.2.1 Convolutional Neural Network

CNN [Kim 2014] uses a filter $\mathbf{F} \in \mathbb{R}^n$ over a window of n words to extract overlapping features throughout the training and testing set consisting of vectors, v_i. The convolution of $\mathbf{F}$ and v_i produces a new feature, c_i:

$$c_i = f(\mathbf{F} \cdot v_i) \tag{3.3}$$

where, f is a non-linear function and '.' is element-wise multiplication.

The new features, $c = \{c_1; c_2; ...; c_i\}$ form a feature map. This feature map acts as an input to the max-pooling layer. This layer uses a non-overlapping filter of length 'a' (in case of one-dimensional pooling layer), and reduces the size of the feature matrix by keeping only the highest values in each batch [Wiatowski & Bölcskei 2017].

3.1.1.2.2 Recurrent Neural Network

RNN [Elman 1990] uses a memory feature to capture and remember the information it has learnt so far. It takes i^{th} word as input and predicts the probability of $(i+1)^{th}$ word using a non-linear function. RNN produces a new embedding matrix, including the information from the previous embedding.

3.1.1.2.3 Long-Short Term Memory

LSTM [Hochreiter & Schmidhuber 1997] is devised to overcome the problem of vanishing and exploding gradients in SimpleRNN [Elman 1990]. It uses memory cells with three gates: input, output and forget gate. LSTM decides which information to remember or forget by assigning appropriate weights to the gates. It also produces a new embedding matrix using the information from the previous one.

Each neural network works in a slightly different way from the other. When using a hybrid of these neural networks, one can increase the efficiency of the model by combining the advantages of networks. For example, using CNN along with LSTM allows CNN to learn the features using overlapping sequences resulting in a matrix of word vectors that contributes to the sentiment of the sentence. This matrix acts as an input to LSTM which can focus on more important data rather than memorising the whole sentence and hence, increasing the efficiency of the model. In few cases, combining neural networks can help form bi-directional networks which can assist in predicting future words. The stacking of layers can increase the non-linearity of the model to produce better decision boundaries and hence, more reliable results. Moreover, as linguistics of each language differs from another, the order of the techniques is also taken into consideration, so that pair of these techniques can be analyzed for different languages. However, the languages considered were based on Devanagari script and therefore no prominent difference is observed.

For each layer, parameters are selected to avoid overfitting, which remains fixed across all the models to compare the results at the end of the experiment. Table 3.1 shows the parameter settings for different layers.

In sequential model, two dense layers have been appended after the hybrid layers with different neuron units as shown in Table 3.1. These layers are fully connected layers of the model containing activation functions that connect each input neuron to each neuron of the next layer. A flatten layer is used before the dense layers to convert the dimensions of the output from previous layers to usable form in the dense layer. Two dropout layers have been applied to each of the dense layers to drop some of the connections of neurons from the dense layer to avoid overfitting (as shown in Figure 3.1). The activation functions used for dense layers are *ReLu* and *Sigmoid* because they do not suffer from the problem of saturation. The *Sigmoid* function as shown in equation 3.4,

$$S = 1/(1 + e^{-x}) \tag{3.4}$$

produces an output from *0* to *1*. It converts the embedding matrix from RNN, LSTM or output from max-pooling layer to a label *0* or *1* which represents negative and positive

Table 3.1: Parameter settings for Sentiment Analysis

Sr.no	Layers	PARAMETERS
1	Embedding	input_dim=5000, output_dim=20, input_length=30
2	SimpleRNN	Neuron units=100
3	LSTM	Neuron units=100
4	CNN	nb_filter=20, filter_length=3, activation= 'relu'
5	Max-pooling	pool_length=2
6	Dropout(Layer 1)	units: Hindi: 0.8 Bengali: 0.7 Tamil: 0.7
7	Dropout(Layer 2)	units: Hindi: 0.5 Bengali: 0.3 Tamil: 0.3
8	Dense(Layer 1)	units=20, activation= 'relu'
9	Dense(Layer 2)	units=1, activation='sigmoid'
10	model.compile()	loss= 'binary_crossentropy', optimizer= 'adam'
11	model.fit()	batch_size=32, epochs=8
12	model.predict()	batch_size=32, verbose=1
13	model.evaluate()	verbose=1

sentiment, respectively. The model is finally compiled with a loss function to yield a binary classification.

The process of sentiment analysis from a set of sentences to generate a label is shown in Algorithm 3.1, where **Y** represents the matrix of labels for each sentence in test data (**Y_ts**) and training data (**Y_tr**).

$$Y = [y_1, y_2, ..., y_L] \tag{3.5}$$

where, each column of the matrix represents a label for a given sentence.

$$y_i = \begin{cases} 0, & \text{for } negative\, S_i \\ 1, & \text{for } positive\, S_i \end{cases} \tag{3.6}$$

Algorithm 3.1: Sentiment Analysis of Indian Languages

```
   Input: x=A set of sentences, $S = \{S_1, S_2, ..., S_n\}$ with n-words, $\{w_1, w_2, ..., w_n\}$, of
          one of the Indian languages
   Output: $\mathbf{y}\epsilon\{positive : 1\ negative : 0\}$
1  Assumption: Each document is a monolingual dataset.
2  for each $S_i$ in $S$ do
3      Phase I: Pre-Processing
4          1. Tokenize
5              $w_t = f(S_i), S_i = \{w_1, w_2, ..., w_n\}\epsilon S$
6              Output: $w_t = \{‘w'_1, ‘w'_2, ..., ‘w'_n\}$
7          2. Pad
8              $w_p = g(w_t), w_t = \{‘w'_1, ‘w'_2, ..., ‘w'_n\}$
9              Output: If words in the largest sentence= $n + k$
10                     $w_p = \{‘w'_1, ‘w'_2, ..., ‘w'_n, 0, 0, 0...ktimes\}$
11     Phase II: Sequential Model
12     if $S_i$ belongs to training data then
13         Feed the data: model.fit($\mathbf{Y}, \mathbf{Y\_tr}$)
14     else
15         Feed the data: model.predict($\mathbf{Y}, \mathbf{Y\_ts}$)
```

3.1.1.3 *Phase III:* Predictions

This phase operates only on the test data. The test data is fed to the sequential model using the function *model.predict*() as explained in algorithm 3.1. The sequential model works upon it and produces labels for each sentence in the test dataset.

3.1.2 Experiments & Results

3.1.2.1 Sentiment Analysis Datasets

The dataset is extracted from Sentiment Analysis of Indian Languages (SAIL) 2015 data [Patra *et al.* 2015], containing twitter data from three Indian languages: Hindi, Bengali and Tamil. Each dataset is divided into two text files: a positive sentences data file and a negative sentences data file. The whole dataset is split into training and testing data in the ratio of *70:30.* Table 3.2 enlists the number of sentences in training and testing set for each of the three Indian languages. The dataset contains monolingual tweets in one of the Indian languages. The textual data is subjected to some transformations referred to as pre-processing phase as discussed in section 3.1.1.

Table 3.2: Dataset description

Sr.no	Dataset	Class	Total Data	Training Data	Test Data
1	Hindi	Positive Negative	168 559	508	219
2	Bengali	Positive Negative	277 354	441	190
3	Tamil	Positive Negative	387 316	492	211

3.1.2.2 The Experimental Setup

The experiment aims at predicting the sentiment of sentences in one of the three Indian languages mentioned before and gauges the correctness of model using performance measures. For this purpose, all possible combinations of RNN, LSTM, CNN are modelled, and performance measures such as accuracy, F-measure, precision and recall have been calculated. Over-fitting has been visualised using accuracy versus number of epochs graph.
The experiment uses a vector-space based model that encodes continuous similarities between words as distances or angle between word vectors in a high-dimensional space [Maas *et al.* 2011]. Analysis of short text is complicated due to the limited amount of contextual data available in the text. Thus, to fill the gap of contextual information in a scalable manner, it is more suitable to use methods that can exploit prior knowledge from large sets of unlabelled texts [Dos Santos & Gatti 2014]. Hence, word embedding is used as the first layer, i.e. the word embedding layer. It takes as input an array of padded numerical arrays of sentences with each sentence padded to *30* integral words and transforms each one of them to an array of feature vectors, each vector being *20*-dimensional. The embedding layer has been trained using hold out method. The integral words are projected as random vectors in the *20*-dimensional space. The model is kept trainable so that these vectors get updated and optimised in the word-vector space during back-propagation while capturing the context of each word.
The next layer is a neural network, which can be a single layer or a combination of RNN [Elman 1990], LSTM [Hochreiter & Schmidhuber 1997] and CNN [Kim 2014]. This network takes feature vectors as input and processes it. For example, convolutional layer takes feature vectors as input, breaks them into overlapping sequences which are read using a filter of fixed window and a feature map is prepared. The feature map is fed to the

max-pooling layer which returns the largest number in the array as output. This output is fed to dense layers where activation functions are applied on them, and final output is given which is a label (*0*: negative; *1*: positive) [Wiatowski & Bölcskei 2017]. Figure 3.1 shows the block diagram of the process occurring in the experiment and figure 3.2 shows how layers for CNN and LSTM are infused to work together.

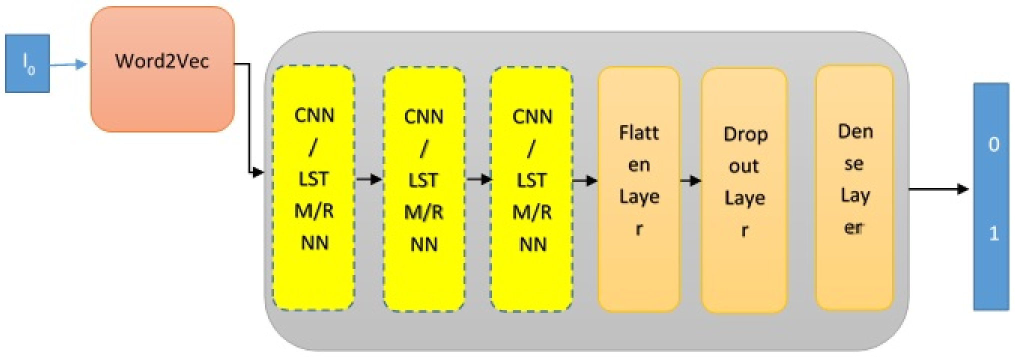


Figure 3.1: Block diagram representing the process of sentiment analysis

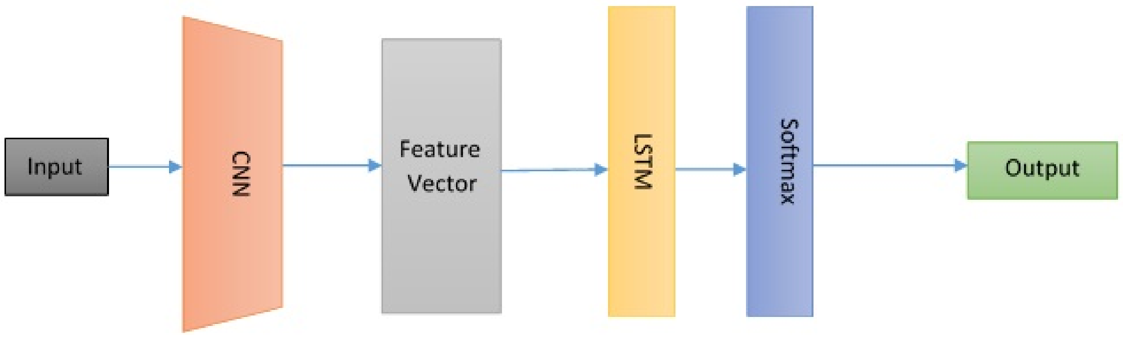


Figure 3.2: CNN+LSTM model

3.1.2.3 Parameter Setting

For model training, the number of epochs is set to eight otherwise, either the model is over-fitting or the performance measures are similar. Over-fitting occurs when the training error is decreasing at a higher rate than the testing error, and thus, the gap between training and testing accuracy increases, this is observed when epochs are higher than eight. The gap between training and testing accuracy remains same at eight epochs in some cases whereas it is decreasing in other cases. Only a few models (three or four out of *39*, that too mostly three-layered networks) underwent over-fitting. However, for Tamil dataset, there is no over-fitting at eight and seven epochs. Seven epochs has been finally chosen for the Tamil dataset because the accuracy obtained for seven epochs is better than eight epochs. A batch_size of *32* has been chosen as the dataset contains short texts of a single line.

3.1.2.4 Results and Evaluation

For evaluating the performance of sentiment analysis, accuracy and F-measure are mainly used to compare the results of different models [Sokolova & Lapalme 2009]. When evaluating sentiment analysis for business interests and political issues, it is important to accurately determine the sentiments, accuracy helps to evaluate the same. F-measure gauges how well the system is performing which is again needed to keep track of system's consistency and also, it takes into account precision and recall. The results for different data sets are discussed below:

3.1.2.4.1 Hindi dataset

The results of top performing models for Hindi dataset are summarised in Table 3.3. Figure 3.3, 3.4 and 3.5 shows accuracy and F-measure for proposed models. The single, dual or triple layer models of CNN performed well concerning accuracy due to multiple reasons. First, CNN has a non-linear activation function (*ReLu*) in itself which captures element-wise non-linearity while any other model does not. Second, CNN derives *n*-grams (sequence of *n* words) and forms feature maps using them which are fed as input to max-pooling layer. These *n*-grams provide non-linear interactions within a local context and improve the performance of the model [Lei *et al.* 2015]. Third, CNN uses multiple filters with the different window size that move over the word embeddings to perform *1*-dimensional convolution. As the filter rolls on, many sequences which capture the semantic and syntactic features in the filtered *n*-grams are generated. Many such features are combined into the feature map, and further operations are performed in the pooling layer which increases the efficiency and accuracy of the results. This allows the model to capture the sentiment of phrases such as 'not so good'. Fourth, the network can easily explore the richness of word embeddings produced by unsupervised pre-training [Dos Santos & Gatti 2014]. Fifth, for small training data, CNN performs better than any other model [Kadlec *et al.* 2015]. The models though, lagged behind in terms of F-measure which indicates that there is no consistency of the system during the iterations (as shown

Table 3.3: Summary of top performance results for Hindi dataset.

Sr.no	Model	Accuracy	F-measure
1	CNN	77.63%	67.85%
2	RNN-LSTM	77.63%	67.85%
3	LSTM	74.89%	69.08%
4	CNN-LSTM-LSTM	76.26%	70.50%
5	LSTM-RNN-LSTM	74.43%	68.8%

in Figure 3.3 and Figure 3.5). When precision and recall are looked upon separately, it is found that these models lacked precision but have high recall score. This signifies a high false positive value for few cases which can probably be due to skipping of low-level features by CNN that could have played a significant role in predicting the sentiment. This is another reason why CNN along with LSTM or RNN performed better because RNN or LSTM can also process low-level features. For Hindi, simple models perform better than combinations because of less language complexity. However, the dataset used has small amount of imbalanced data. Complex models may perform better on larger dataset. Imbalance in Hindi data set is handled by assigning weights to class label such that the cost function penalizes loss on certain class more severly. This makes the model adapt better to the characterstics of a minority class.

Table 3.4: Summary of top performance results for Tamil dataset.

Sr.no	Model	Accuracy	F-measure
1	RNN-CNN-LSTM	71.56%	69.7%
2	RNN-LSTM-RNN	69.19%	68.7%
3	RNN-RNN-LSTM	68.72%	68%
4	RNN-LSTM	67.30%	66%
5	CNN-LSTM-CNN	67.77%	49%

The RNN-LSTM model performed equally well as the CNN model. It acts as a bidirectional RNN model which has been shown to improve the performance of a model [Schuster & Paliwal 1997]. The feature vectors from the embedding layer are fed to RNN which reads the whole sentence and stores in its short term memory. The output from RNN will be fed to LSTM which has memory cells that control whether to keep the data, read it or forget it, by assigning weights to the data. However, the problem may occur when this results in loss of information due to error persistence in LSTM as LSTM's memory cells are linear neurons and cause the error to be carried forward for a longer time.

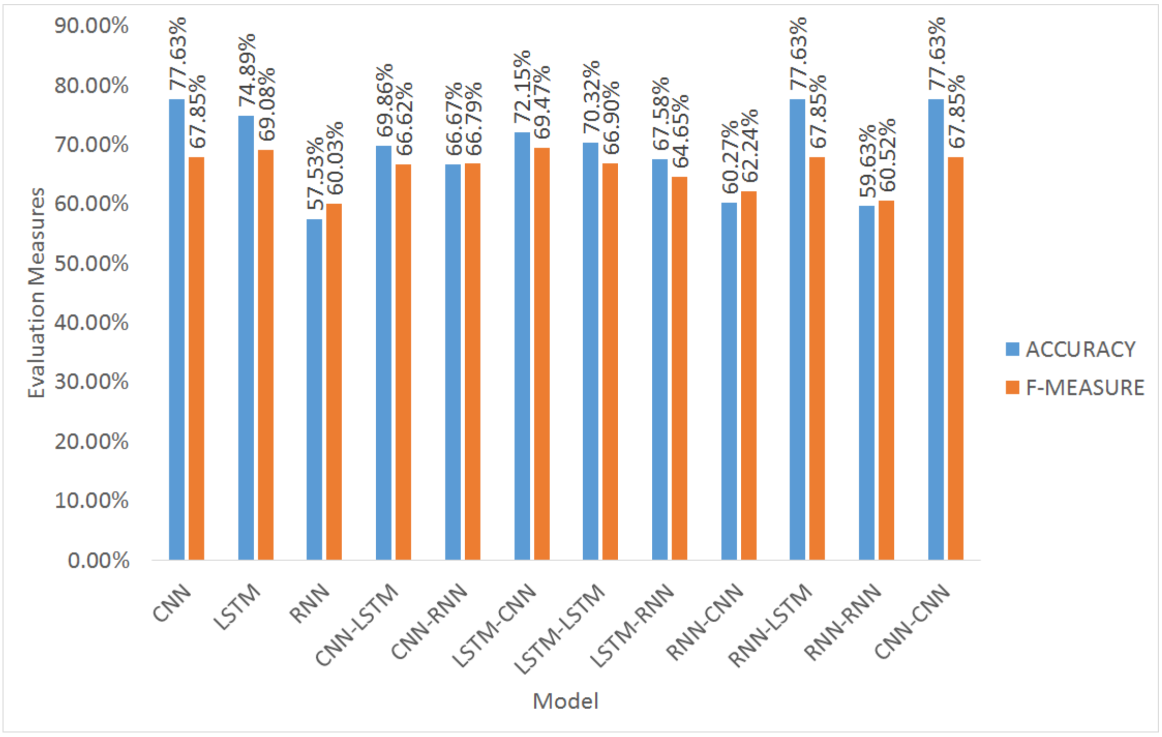


Figure 3.3: Accuracy and F-measure for one and two-layer models(Hindi)

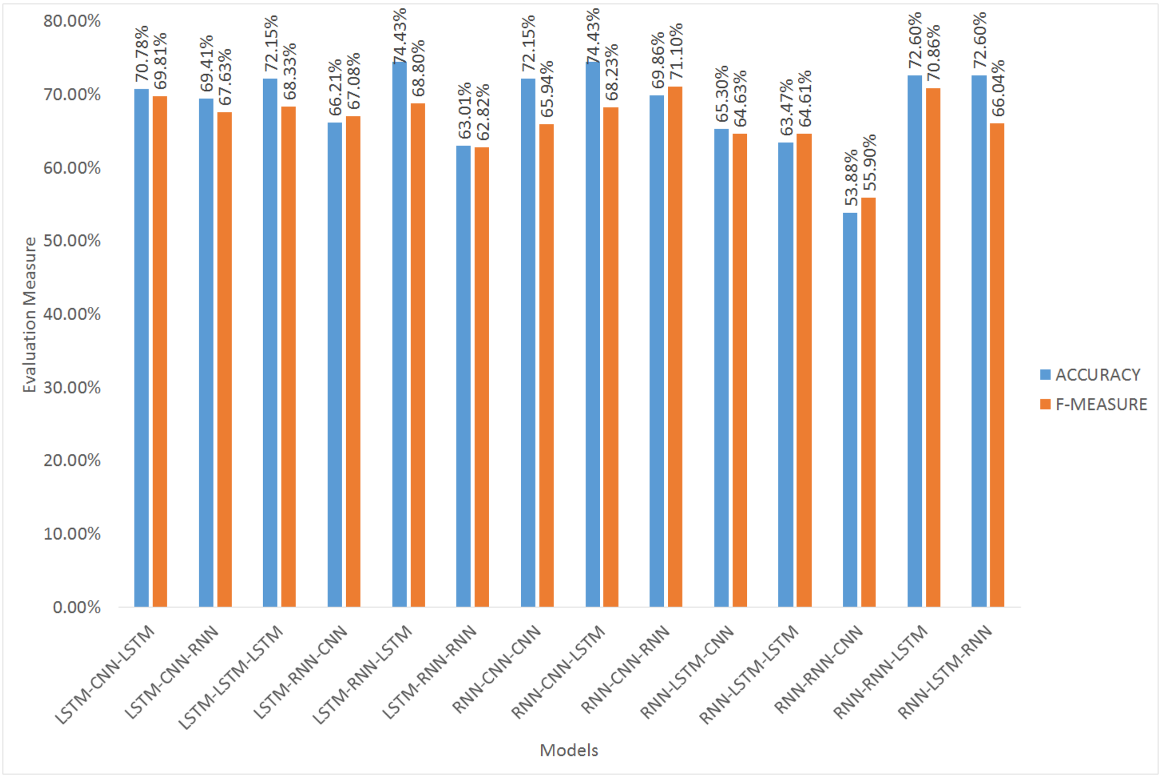


Figure 3.4: Accuracy and F-measure for three-layer models(Part-1)(Hindi)

This problem can be resolved using RNN layer before LSTM. Since RNN has nodes interconnected to other nodes in conjunction to themselves, the information lost from LSTM

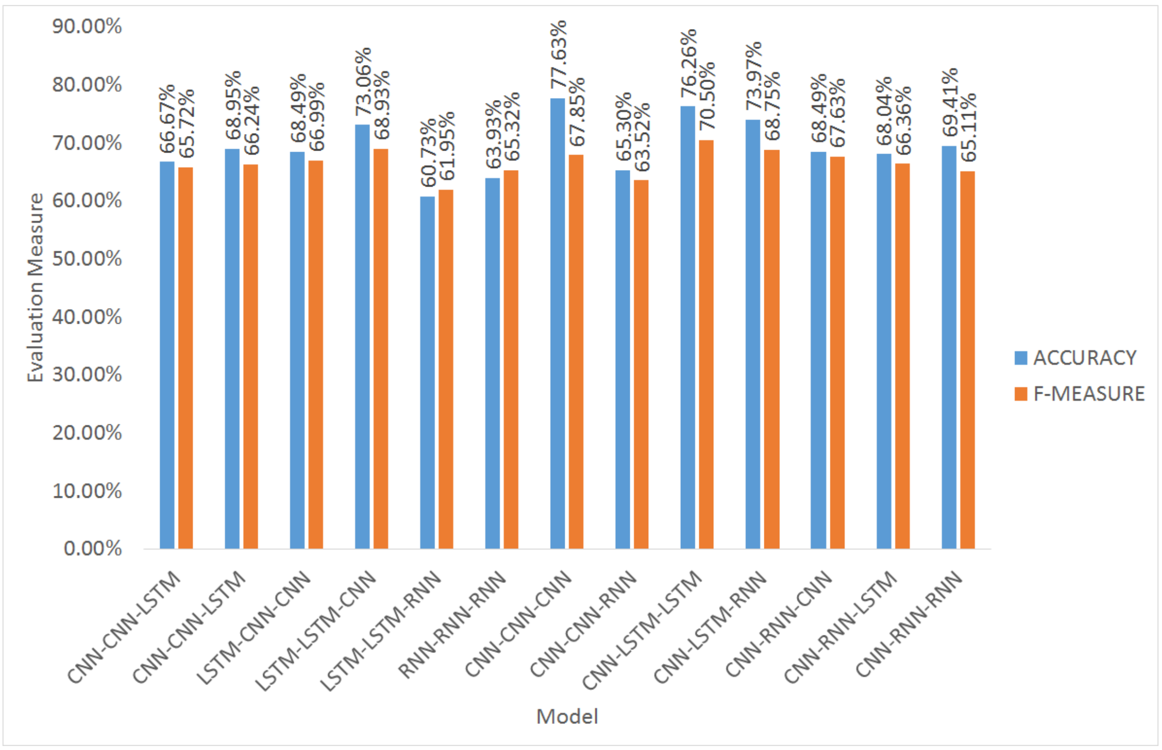


Figure 3.5: Accuracy and F-measure for three-layer models(Part-2)(Hindi)

can be stored in RNN for a while so that it is not lost and can be propagated further if required during discrepancies [Hakkani-Tür *et al.* 2016]. Thus, this network handles the loss of information using RNN and loss of efficiency using LSTM which can adapt to new data rapidly [Graves 2013]. LSTM and LSTM-LSTM performed well due to the ability of LSTM to adapt to new data rapidly [Mikolov *et al.* 2013a], but triple layered LSTM underwent some over-fitting (as shown in Figure 3.3 and Figure 3.4). RNN-CNN-LSTM performed pretty well with an accuracy of *74.43%* and F-measure of *68.23%* as shown in Figure 3.4. CNN and LSTM are complimentary, where CNN captures the local invariant regularities, LSTM is good at modelling temporal features. The RNN helps to maintain the memory of the words in the sentence [Wang *et al.* 2016].

RNN-RNN-CNN gave an accuracy of only *53.88%* (as shown in Figure 3.4), which is least among all the models and is also exceptionally low as all other accuracies range from *60%* to *70%*. This is because the model underwent over-fitting with testing accuracy decreasing at a very high rate and training accuracy increasing as shown in Figure 3.6.

RNN and RNN-RNN did not perform well as compared to CNN and CNN-CNN as shown in Figure 3.3. LSTM follows the same trend. This is probably because in case of

RNN and LSTM, increasing the number of layers would only increase the complexity and error. This is due to linear stacking of errors that reduce accuracy as the number of layers increase. While in case of CNN, increasing number of layers increases the ability of CNN to process even the smallest but relevant information, which can otherwise be missed during convolution. This means that increasing number of layers of CNN should increase the accuracy but as the number of layers increases, the error in classification also increases [Wang *et al.* 2016] and thus, compensating for the increase in accuracy and resulting in the same accuracy.

Table 3.5: Summary of top performance results for Bengali dataset.

Sr.no	Model	Accuracy	F-measure
1	LSTM-CNN	57.37%	45.8%
2	RNN-LSTM	56.84%	45.13%
3	CNN-RNN	56.32%	55.60%
4	RNN-CNN-CNN	57.37%	57.1%
5	RNN-LSTM-LSTM	54.21%	53.80%

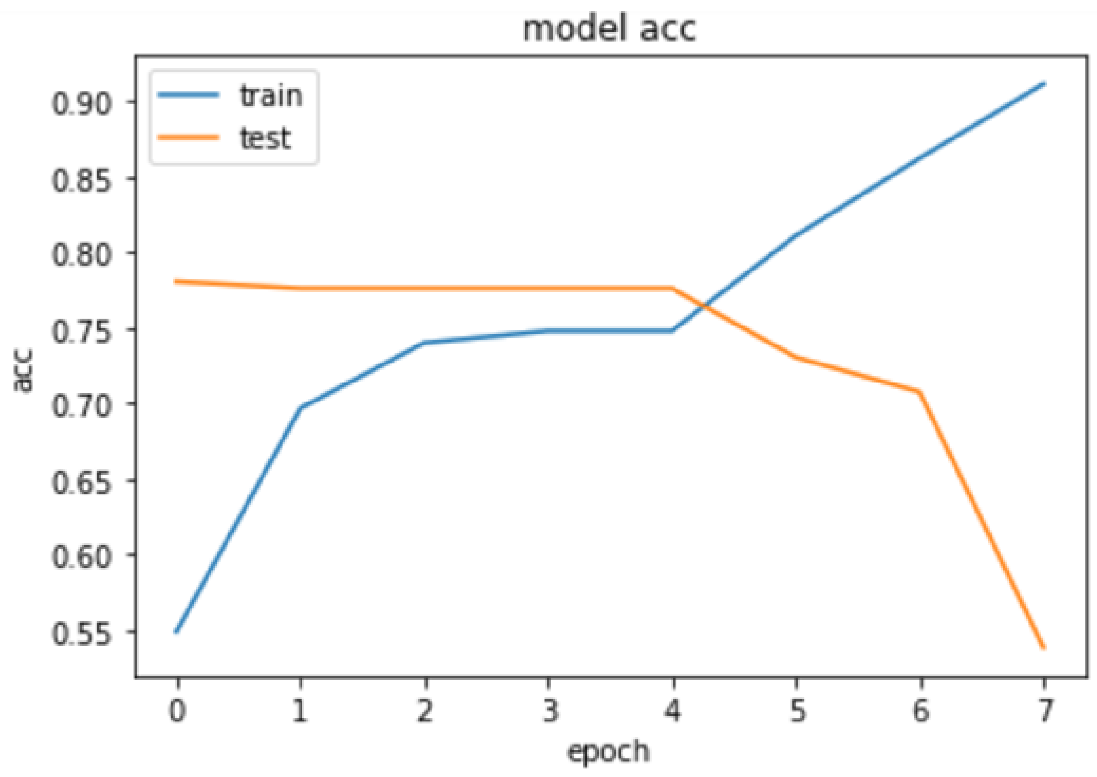


Figure 3.6: Over-fitting in RNN-RNN-CNN model(Hindi)

3.1.2.4.2 Bengali dataset

The Bengali dataset is small as compared to Hindi dataset and hence, to avoid over-fitting, the dropout units are reduced as shown in Table 3.1. The number of epochs has been kept same as discussed in section 3.1.1.

According to the results shown in Figure 3.7, 3.8 and 3.9, models with two hidden layers are more successful as compared to single or triple hidden layer models. This is because Bengali is a bit more complicated than Hindi [Bag & Harit 2011] and therefore, the feature extraction of more curvilinear fonts in Bengali need more complex models. Hence, dual-layered models performed better than single-layered models but the triple-layered models underwent error accumulation and in some cases, over-fitting due to the small dataset. Table 3.5 summarises the results for top performing models. CNN models did not perform well when compared to CNN models in Hindi dataset.

LSTM-CNN network is an excellent network for query classification as CNN and

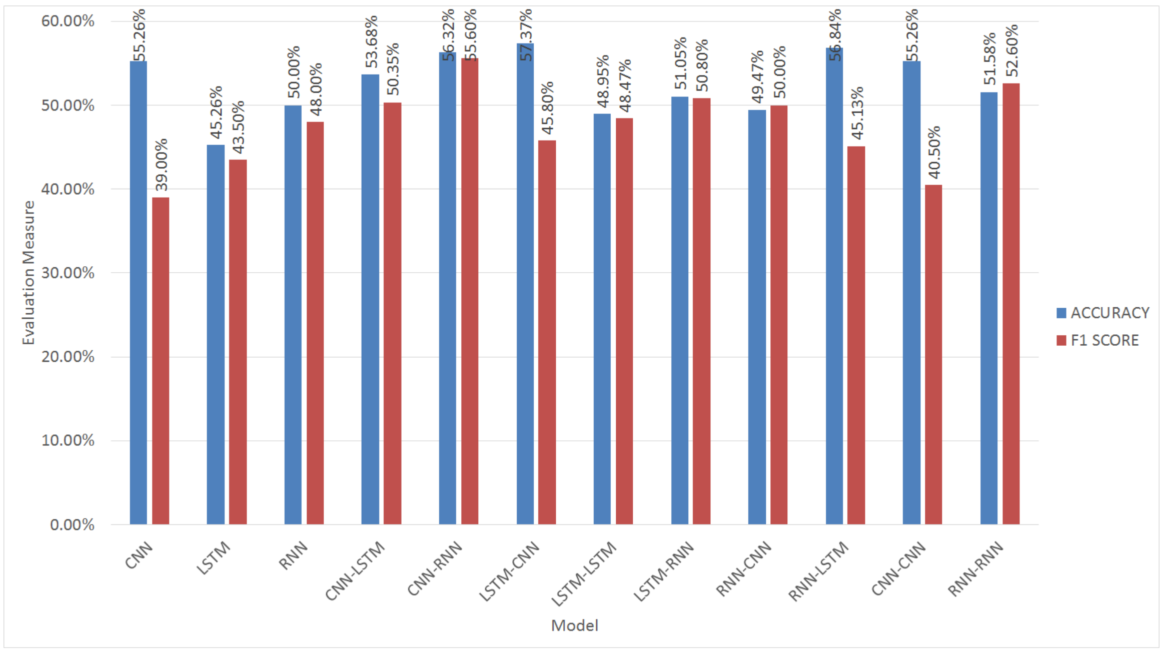


Figure 3.7: Accuracy and F-measure for one and two layer models(Bengali)

LSTM are complimentary, where CNN captures the local invariant irregularities, LSTM equipped with input, output and forget gates, extract the first order feature representation using the memory of the whole sentence. This is essential for a complicated language like Bengali (This feature is missing in RNN, and hence the performance of RNN-CNN is not at par with LSTM-CNN). LSTM can adapt to new sentences quickly which gives it an upper edge over models utilising only CNN layers for sentiment classification of new sentences [Graves 2013]. This is used along with feature extraction by CNN to give a higher accuracy.

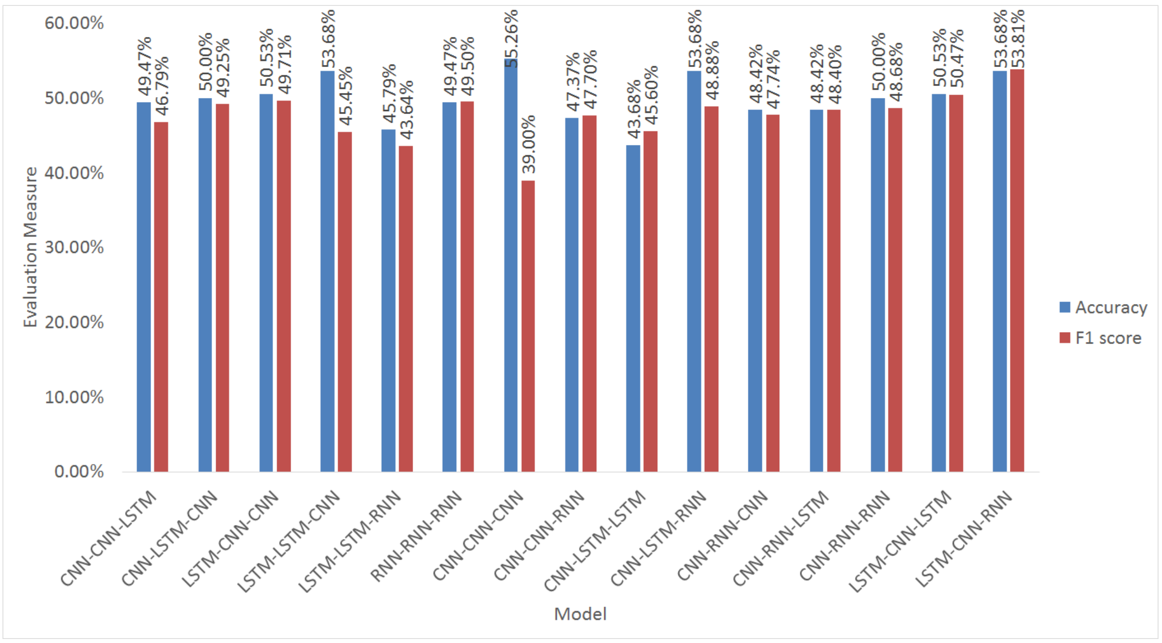


Figure 3.8: Accuracy and F-measure for three-layer models (Part-1)(Bengali)

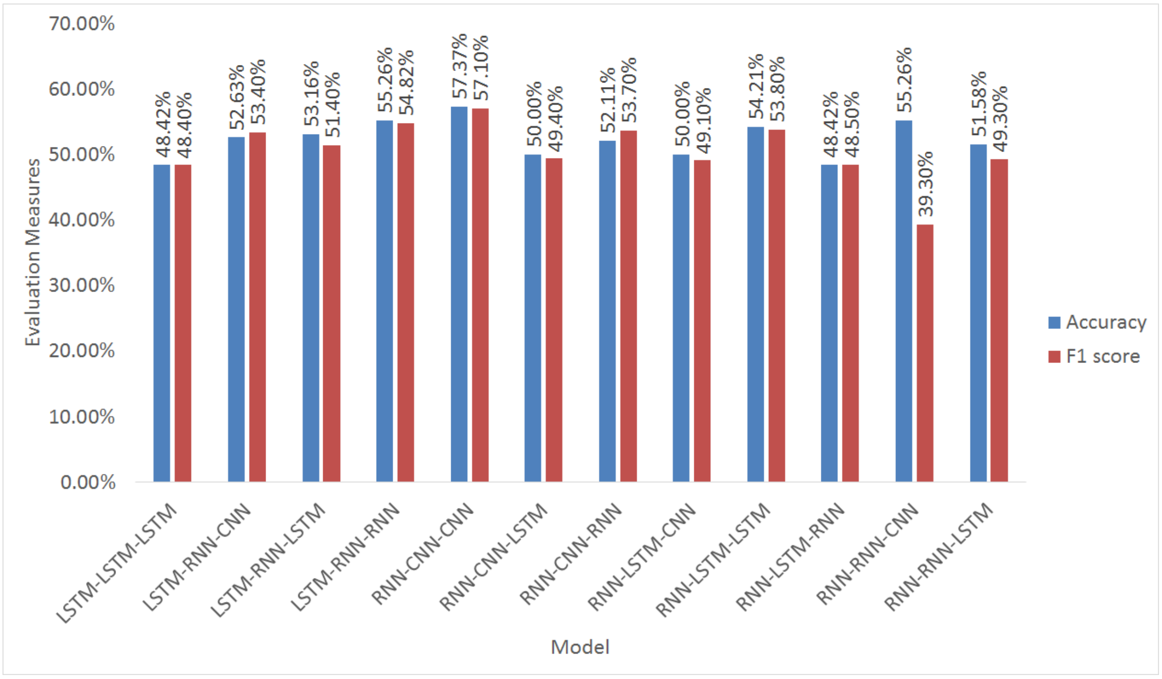


Figure 3.9: Accuracy and F-measure for three-layer models (Part-2)(Bengali)

RNN-CNN-CNN has an accuracy of *57.37%* and F-measure of *57.1%* (as shown in Figure 3.9) which seems to be the highest among all models but it turns out that the model is over-fitting due to the small dataset as shown in Figure 3.10. The training accuracy is increasing at a higher rate than test accuracy, which is a case of over-fitting.

In Bengali dataset as well, CNN models did not perform well in terms of F-measure for the same reason as explained 3.1.2.4.1. RNN-RNN-CNN too has a very low F-measure due to less consistency of the results which means it predicted some sentiments as positive while they are negative. This could have been due to small training data.

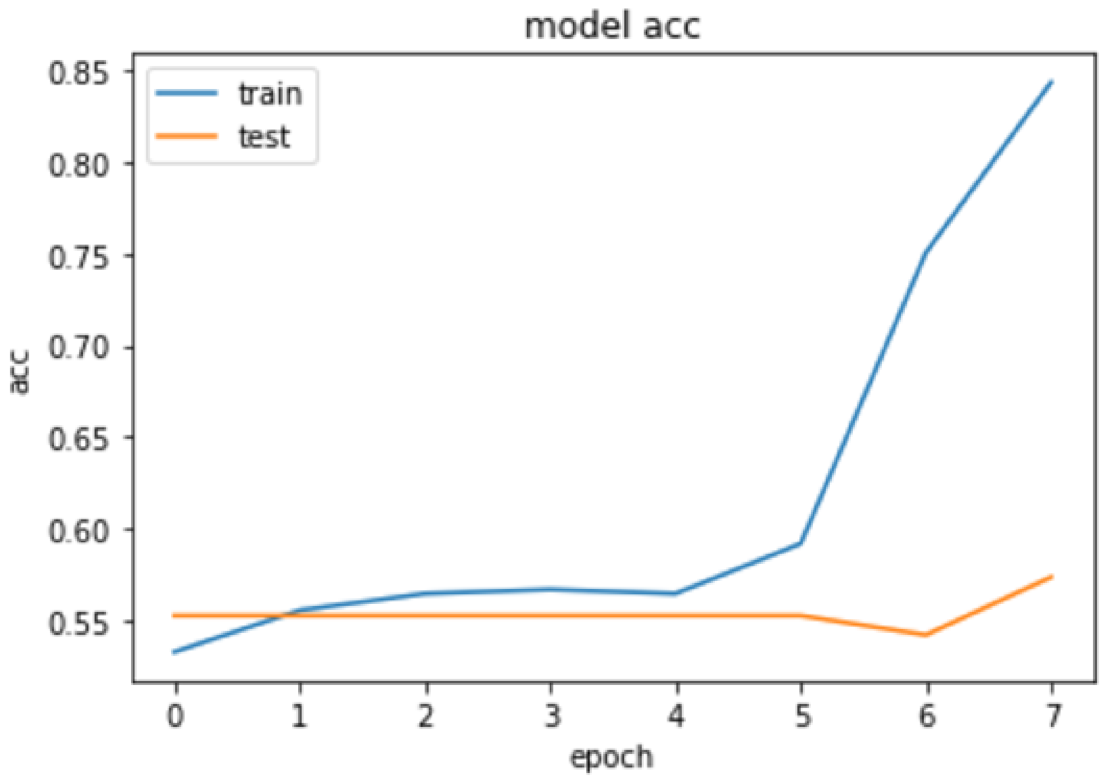


Figure 3.10: Over-fitting in RNN-CNN-CNN model(Bengali)

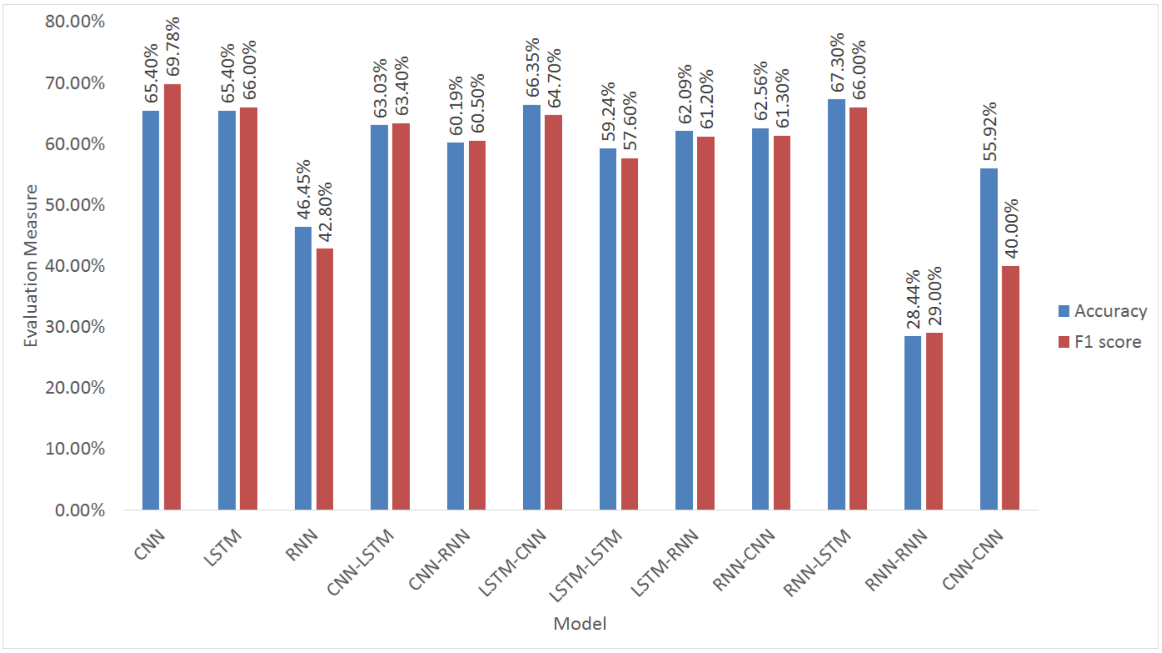


Figure 3.11: Accuracy and F-measure for one and two layer models(Tamil)

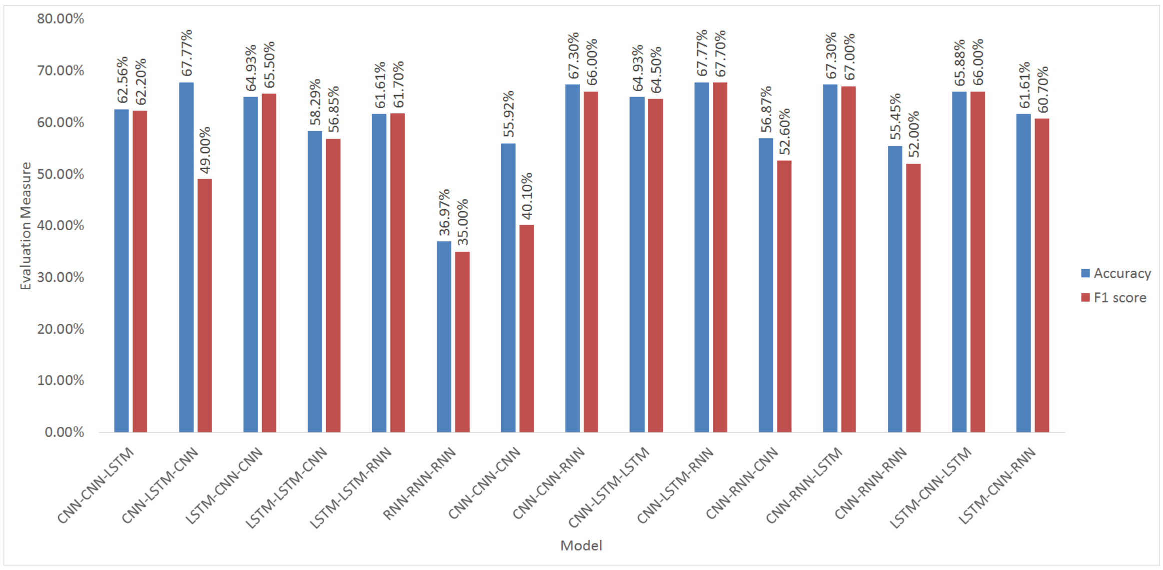


Figure 3.12: Accuracy and F-measure for three-layer models (Part-1)(Tamil)

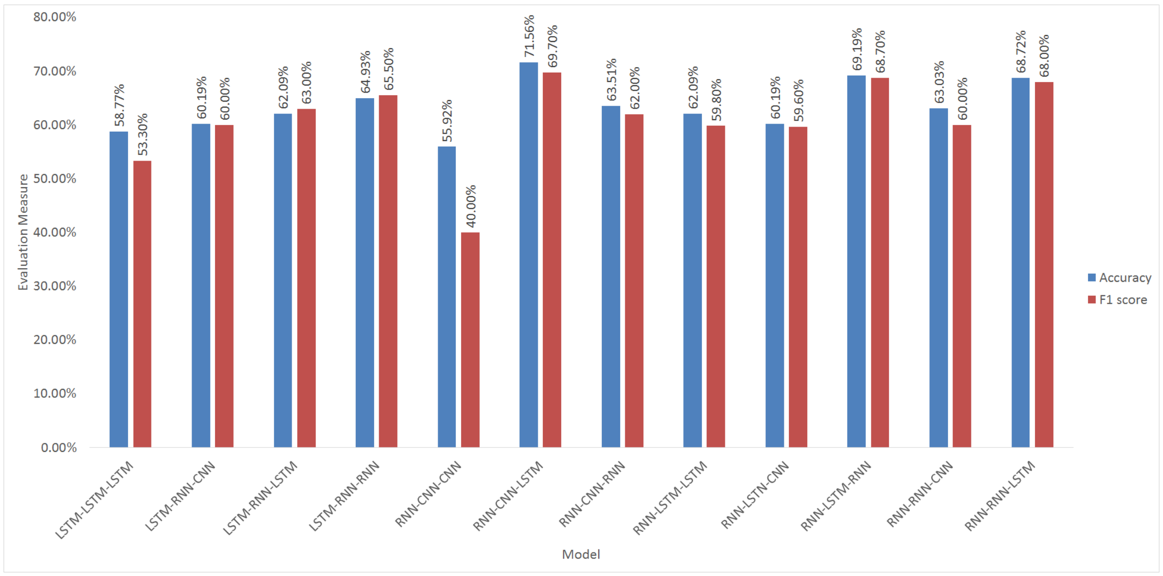


Figure 3.13: Accuracy and F-measure for three-layer models (Part-2)(Tamil)

3.1.2.4.3 Tamil dataset

Tamil dataset is small as compared to Hindi dataset and hence drop out units are reduced to avoid over-fitting as shown in Table 3.1. The results can be seen from Figure 3.11, 3.12 and 3.13. Table 3.4 summarises the results for top performing models. As can be seen from Table 3.4, three hidden layer models gave the best results without over-fitting. This may be due to the fact that Tamil is even more complicated than Bengali and Hindi [Mohanty 1998], due to its curvilinear and multi-tier structure with a contrast of *matra* combinations. Hence, it needs more complex models to get its features extracted to analyse the sentiment of its sentences.

RNN-CNN-LSTM outperformed all other models according to the results, but it turns out that the model is over-fitting with training accuracy increasing at a higher rate than testing accuracy as shown in Figure 3.14. Hence, the top model of proposed approach became RNN-LSTM-RNN which did not over-fit and gave the highest accuracy as well as F-measure.

RNN-LSTM performed well on Tamil dataset as well, as shown in Figure 3.11. The details of this model have been discussed in section 3.1.2.4.1. Adding a layer of RNN to this model only increases the complexity which is needed to extract the features from Tamil sentences. RNN would not filter the words unlike CNN and LSTM but only provide memory to the model. It acts as a bidirectional RNN-LSTM which is essential in cases when future information is as valuable as previous information [Schuster & Paliwal 1997]. The same reasoning applies to RNN-RNN-LSTM model whose accuracy is almost equal to RNN-LSTM-RNN and F-measure is precisely same (as shown in Figure 3.12). Using LSTM as the common layer increases the accuracy in case of RNN-LSTM-RNN due to its tendency to keep a memory of long sentences and filtering techniques.

CNN-LSTM-CNN performed well concerning accuracy but lagged a great deal concerning F-measure and hence, can't be considered as one of the best models as the results are being analysed considering both accuracy and F-measure. RNN-RNN model resulted in over-fitting (as shown in Figure 3.15) and obtained an accuracy which is not comparable to other models. It attained its highest accuracy in the early epochs and underwent over-fitting afterwards. RNN-RNN-CNN resulted in a very low F-measure due to the same reasons as discussed in section 3.1.2.4.1.

It is observed that on all the three data sets, RNN-LSTM performed well but the accuracies and F-measure are different as shown in Figure 3.16. One can deduce from the bar graph that more the data available in a dataset, better is the performance of a given model on the dataset. It also shows that Hindi being the least complex language [Bag & Harit 2011], to extract features from, performance measures for Hindi dataset are better than for other two languages. It is worth noting that although Tamil is more complicated than Bengali in terms of language complexity [Mohanty 1998], however, Tamil dataset showed more accurate results than Bengali dataset. This is because the data available in Tamil dataset is larger than Bengali dataset and hence, more accurate results are obtained.

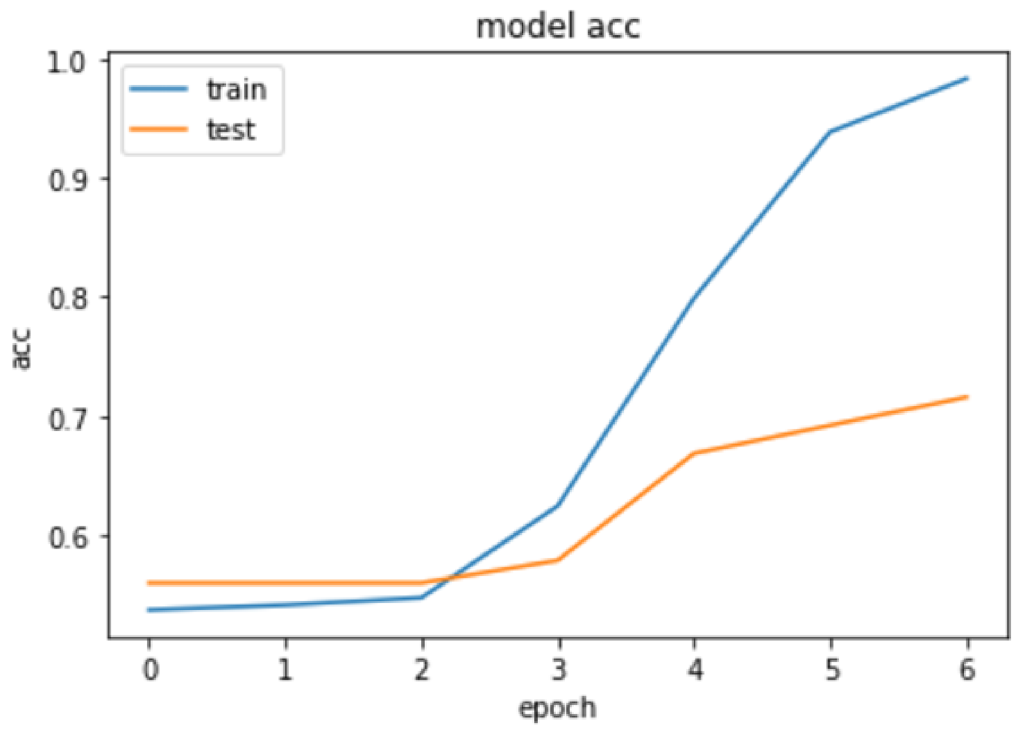


Figure 3.14: Over-fitting in RNN-CNN-LSTM model(Tamil)

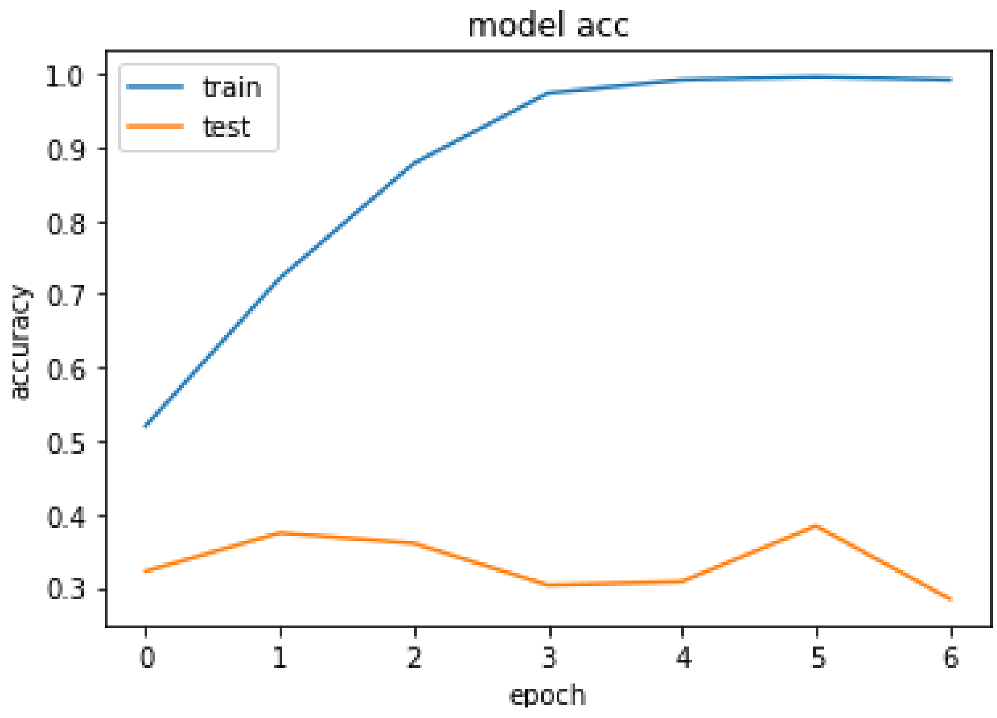


Figure 3.15: Over-fitting in RNN-RNN model(Tamil)

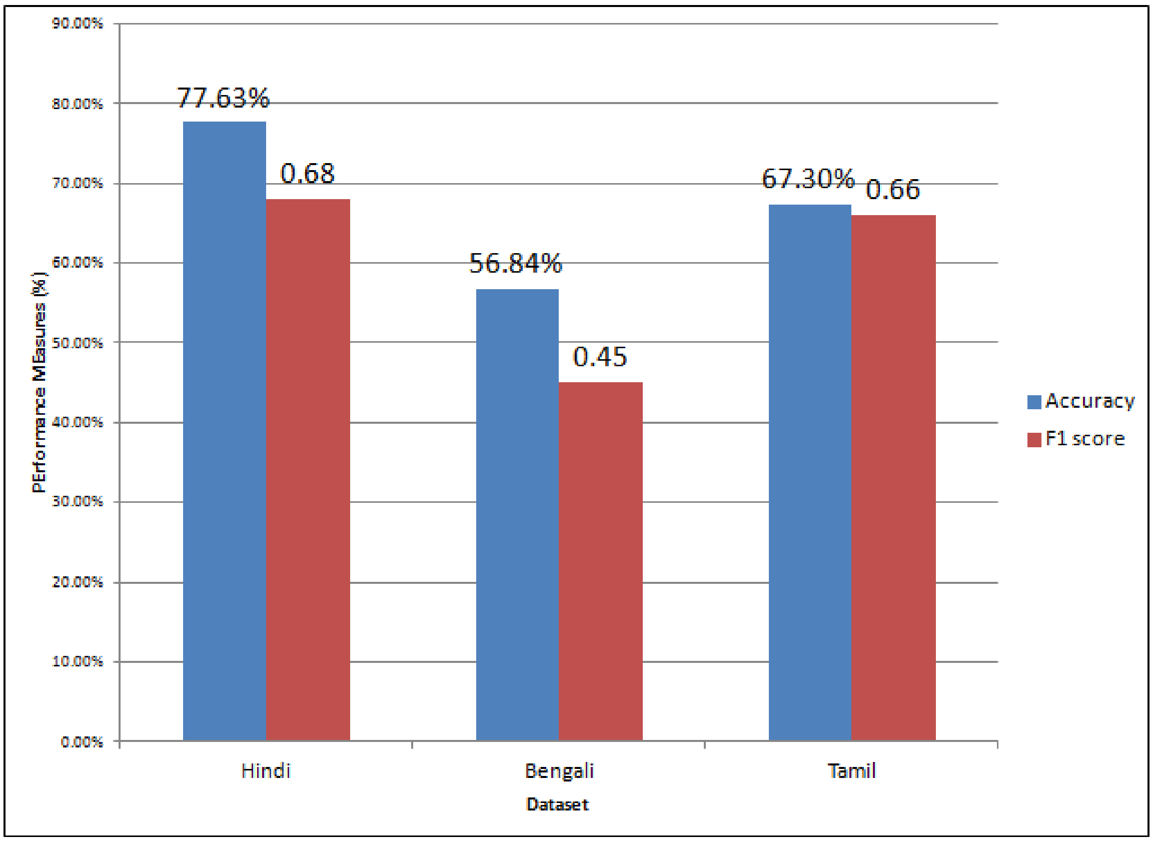


Figure 3.16: Comparison of performance of RNN-LSTM on different datasets

3.1.2.5 Comparison with other existing approaches

The best approach obtained in SAIL 2015 shared task [Kumar *et al.* 2015a] on basis of the results is used as baseline approach. Most of the approaches in SAIL task used SentiWord-Net (Indian Languages) for training [Patra *et al.* 2015]. Approaches extracted word uni-grams and bigrams from the dataset, trigrams and quad-grams for word prefixes and suf-fixes. The lexical expansion is used to take into account the rare and unseen words[Kumar *et al.* 2015a]. However, the approach proposed in this work obtains better results (as shown in Table 3.6) than any of the models used in SAIL 2015. One of the reason is use of neural networks in proposed approach instead of traditional machine learning techniques. The highest accuracy scored [Kumar *et al.* 2015a] is using five-fold cross-validation technique using SVM. Instead of K-Fold cross-validation, hold out method is used for evauation, due to the following reasons[Krueger *et al.* 2015]:

1. K-Fold cross-validation is computationally expensive and so, is not worth the trouble for validating *39* different models for each of the three languages.

2. The datasets used are small. K-Fold cross-validation undergoes under-fitting on small datasets.

For comparison, the experiment is performed using five-fold cross-validation. The results are remarkably good for hybrid models (as shown in Table 3.6) as compared to the highest accuracy model [Kumar *et al.* 2015a]. Advantages of neural networks over traditional machine learning techniques are as follows:
Firstly, to predict the sentiment of a sentence, there is a need to look at the sentence as a whole. The sentiment of a sentence can be entirely different from the sentiment of its constituent words. Therefore, a memory factor is required to remember the previous and future (in case of bidirectional neural networks) words to predict the probability of the present word. This can be achieved using neural networks only.
Secondly, traditional machine learning techniques are non-parametric models and cannot learn highly complex functions. On the other hand, the neural network is a deep architecture and can combine lower-level features to high dimensional representations. Using the concept of co-occurrence dependencies, i.e. words having similar sentiments occur together in high-dimensional space, the neural network can make progress on high complexity tasks without human intervention [Bengio *et al.* 2007].

Table 3.6: Comparison with SAIL 2015

Dataset	Proposed approach (RNN+LSTM)	[Kumar *et al.* 2015a]
Hindi	68.51%	55.67%
Bengali	52.62%	43.20%
Tamil	59.88%	39.28%

3.2 Temporal Sentiment Analysis

Temporal and burst analysis helps to identify trending features and their impact on the market. Hence, presenting an opportunity to the company for coming up with new features, user satisfaction, user demand and align with the trends. Temporal analysis can provide list of what all features are trending and which feature has become obsolete according to customer's sentiment towards the brand and aspects. This helps the companies to identify their strengths and weaknesses. Using this information companies can improve the user experience and make advancements.Temporal sentiment analysis

can provide valuable insights and thus help organizations to formulate effective business strategies. It can help firms to monitor brand and product performances, handle customer grievances, get in-depth information for strategic analysis. Temporal sentiment analysis can help to track and come up with effective marketing campaigns. With temporal sentiment analysis may increase product quality. Market research teams would be able to gauge consumer needs and preferences better. Ideas for product improvement can also be obtained from targeted customers. It can help to identify opportunities for up-selling, reduce customer churn, increase customer acquisition, improve customer retention and handle customer grievances. Temporal sentiment analysis opens a whole new avenue and growth opportunity in terms of a new level of customer engagement and reputation management. Market shares of brands and products can be regularly monitored. Innovative need-based products can be delivered and built. Brand awareness and brand reputation can be monitored by monitoring customer sentiment in real-time and over period of time. It also helps to identify business problems before they aggravate in proportion. So, smart product and marketing strategies can be developed by keeping in mind the consumer needs. It can also be used for keyword monitoring for marketing professionals. It can also help to identify new business opportunities. Specific phrases and texts used by audiences can be monitored to effectively generate new leads. Competitor performances can also be evaluated by monitoring mentions of the competing brands. It can also help to identify marketing campaigns that are not working well. Firms can then effectively modify or withdraw those campaigns. Such timely preventive actions help the brands to grow and prevent negative brand image. Temporal sentiment analysis has become the gateway to understanding consumer needs, extending the customer base and expectations. It helps to pinpoint the problem and give solutions effectively. It can also help in customer segmentation by identifying segments that feel strongly about a brand or service.
A framework has been proposed which targets identification of temporal aspects and their changing opinions with time. A more general and complex task is to predict the aspect mentioned in reviews/opinions and the sentiment associated with each one of them. This task is usually known as fine-grained opinion mining or aspect-based sentiment analysis. For example, "Food is decent, but service is bad", contains positive sentiment to the feature/aspect of food but a strong negative opinion for the service. Predicting the overall

sentiment for the review would mark it as a negative sentiment overall but it neglects the opinion that food is good. Feature extraction is identifying various aspects/subjects which are referred and talked about in the reviews. These features may be directly mentioned in the review or may be present implicitly.

The framework based on hybrid techniques has been proposed to tackle the task of feature extraction, aspect-based opinion mining and analyses them over the time. It uses pre-trained word embeddings [Pennington *et al.* 2014] and studies the semantics and syntactic information encoded in the embeddings. These embeddings help in better initialisation of deep network in the framework, without any domain-specific feature engineering effort. The framework also identifies features from the temporal stream of opinions and gain insights such as, how various features and aspect affect the overall sentiment, which essential features plays a significant role in determining the sentiment and find their variation with time?

3.2.1 Dataset Description

Dataset has been assembled from the various data sets given by SemEval [Pontiki *et al.* 2016] with additional temporal information. Dataset has approximately *3000* reviews for the Laptops. The data consists of reviews broken into sentences, and each sentence is annotated with the categories of aspect present and the corresponding polarities of sentiment for each of the categories. The aspect category consists of various entities E such as a laptop, keyboard, battery, screen and attributes A such as performance, design, build. The *E#A* pair defines the entity-aspect category. Each of the *E#A* pair assigned to the review sentence is given polarity from a set $P = \{positive, negative, neutral\}$.

The annotations are assigned at sentence level, taking into consideration the context of whole review. In the Laptop dataset, there are in total *22* entities and *9* attributes assigned to various entities, thus generating *(22* x *9) E#A* pairs.

3.2.2 Proposed Approach for aspect based temporal opinion mining

The proposed framework for aspect based temporal opinion mining has been divided into three components:

1. Aspect based Sentiment Analysis
2. Temporal Unsupervised Feature Extraction
3. Temporal Analysis of Sentiment and Features

Data is preprocessed before aspect-based sentiment analysis. In preprocessing, sentences of the reviews are tokenised. Numeric or a single character token is discarded to reduce the noise. This data is then used to generate word embeddings. Further, the aspect class distribution is analysed, and the aspects which are less frequent in the training dataset are clubbed together to a newly created aspect class called *'OTHER'*. As dataset might contain reviews for which no aspect has been described, another new aspect *'NONE'* is assigned to them. Aspect list includes the frequently common aspects (static) along with *'OTHER'* and *'NONE'* aspects, and is further used to train the aspect model. Reviews which have *'OTHER'* and *'NONE'* aspects are used to identify new aspects in the second phase of the framework, i.e. temporal feature extraction.

3.2.2.1 Aspect-based Sentiment Analysis

Figure 3.17 represents the block diagram of aspect-based sentiment analysis model. This phase is divided into two parts. The first part includes aspect model which will predict aspect from the reviews whereas the second part predict its associated sentiments.

3.2.2.1.1 Aspect model

This sub-module takes word embeddings of sentences as input and outputs distribution of probabilities for the aspect classes. Here, multi-layer perceptron model for the aspect classification task has been proposed. Pre-trained word embeddings [Pennington *et al.* 2014] has been used. These embeddings are kept static and not fine-tuned while training to avoid overfitting on the small amount of data.

The layered neural network model is used for the identification task. A fully connected layer of the neural network with 'ReLu' activation function is used as a first layer and followed by softmax layer which yields the output distribution over the aspect classes. Dense layer and dropout layer are present in between these layers to avoid overfitting. The representation for the sentence is chosen to be the average of the word vectors, for words in the sentence. Binary cross entropy is used as the loss function where the output $y_i = 1$, if the aspect is present and *0* otherwise. A threshold t is taken (for which F-measure score is maximum) such that aspect is predicted whenever output y_i is higher than the threshold. The number of epochs (for which the model is trained) and the threshold t are the hyperparameters which are fixed by using validation data.

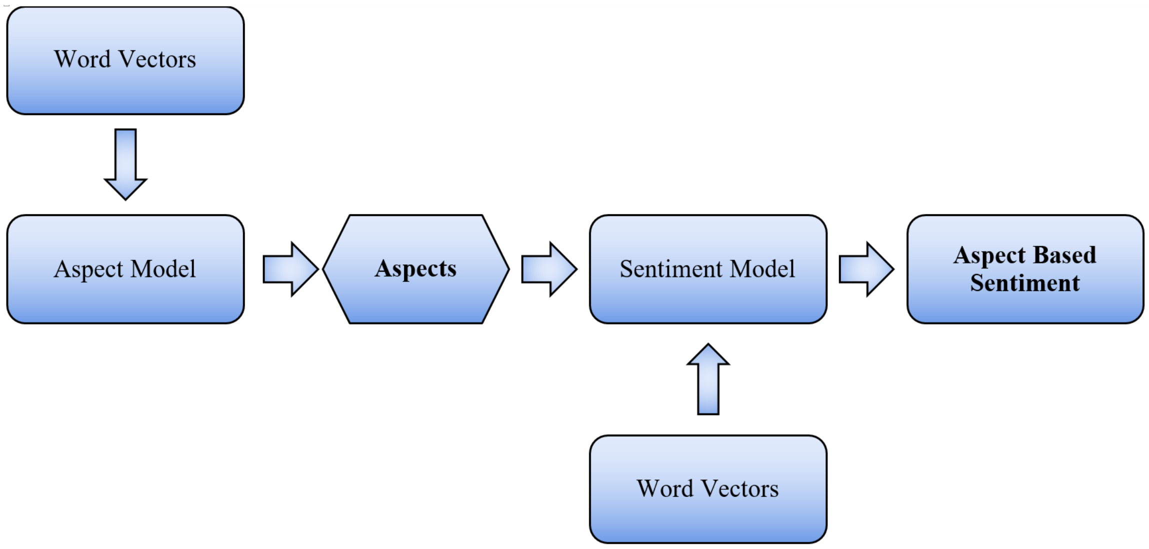


Figure 3.17: Block diagram for aspect-based Sentiment Analysis

3.2.2.1.2 Sentiment model

Sentiment model considers the probabilities of various aspects predicted by the aspect model as input and yields the polarity of sentiments attached to each of the aspects as output. The sentiment is assigned to the predicted aspects by re-scaling the word embeddings with factors dependent on the aspects. The sentence representation used in the aspect model no longer works for the sentiment model as the interaction and location of the tokens are also essential for predicting sentiments.

CNN architecture is used for this model, which applies a convolution to a window of n number of consecutive words. Max-over-time pooling is applied to the values which are obtained as feature map from all the filters. The input passed to the CNN is re-scaled word vectors of sentences. The largest value out of the feature map is selected by the pooling layer which implies that the magnitude of the word vectors has a strong influence on the behaviour of the model. Hence, the impact of the word vector can be enhanced and dampened, by scaling it up and down respectively.

For all the predicted aspects, the scaling factor is calculated for each of the tokens. The scaling factor for the word-vectors is decided by using the probabilistic mass p_i of word i in a sentence towards the particular aspect and distance between the words. The probabilistic mass p_i is calculated by passing the single token to the aspect model and calculating the probability for the required aspect. $d_{i,j}$ is defined as the distance between the tokens in the dependency tree obtained from the Stanford parser. $p_{i \to j}$ denotes the probability of propagation from word i to word j denoted by equation 3.7. p_j denotes the aggregated probability for word j. Equations 3.7, 3.8, 3.9 are used to compute the probabilities mentioned above.

$$p_{\mathrm{i} \to \mathrm{j}} = p_{\mathrm{i}} \exp \frac{-d_{ij}^2}{2h} \qquad \forall i! = j \tag{3.7}$$

$$p_{\mathrm{i} \to \mathrm{i}} = 1 + p_{\mathrm{i}} \tag{3.8}$$

$$p_{\mathrm{j}} = \Sigma p_{\mathrm{i} \to \mathrm{j}} \tag{3.9}$$

In equation 3.7, h is the height of the Stanford dependency parse tree and acts as a normalisation constant to adjust the differences between the shorter and longer sentences.

$$\vec{V}_{\mathrm{j}} = \hat{p}_{\mathrm{j}}.\vec{V}_{\mathrm{j}} \tag{3.10}$$

The re-scaled word embeddings are then passed to the CNN. The learning model is designed as a combination of multiple n-gram convolutional models. The output from each of the convolutional model is then flattened and concatenated to pass as input to the fully connected dense layer. The final softmax layer produces the output probability distribution over the output classes, i.e. positive, negative and neutral. Overall the aspect-sentiment model is summarised in Figure 3.18.

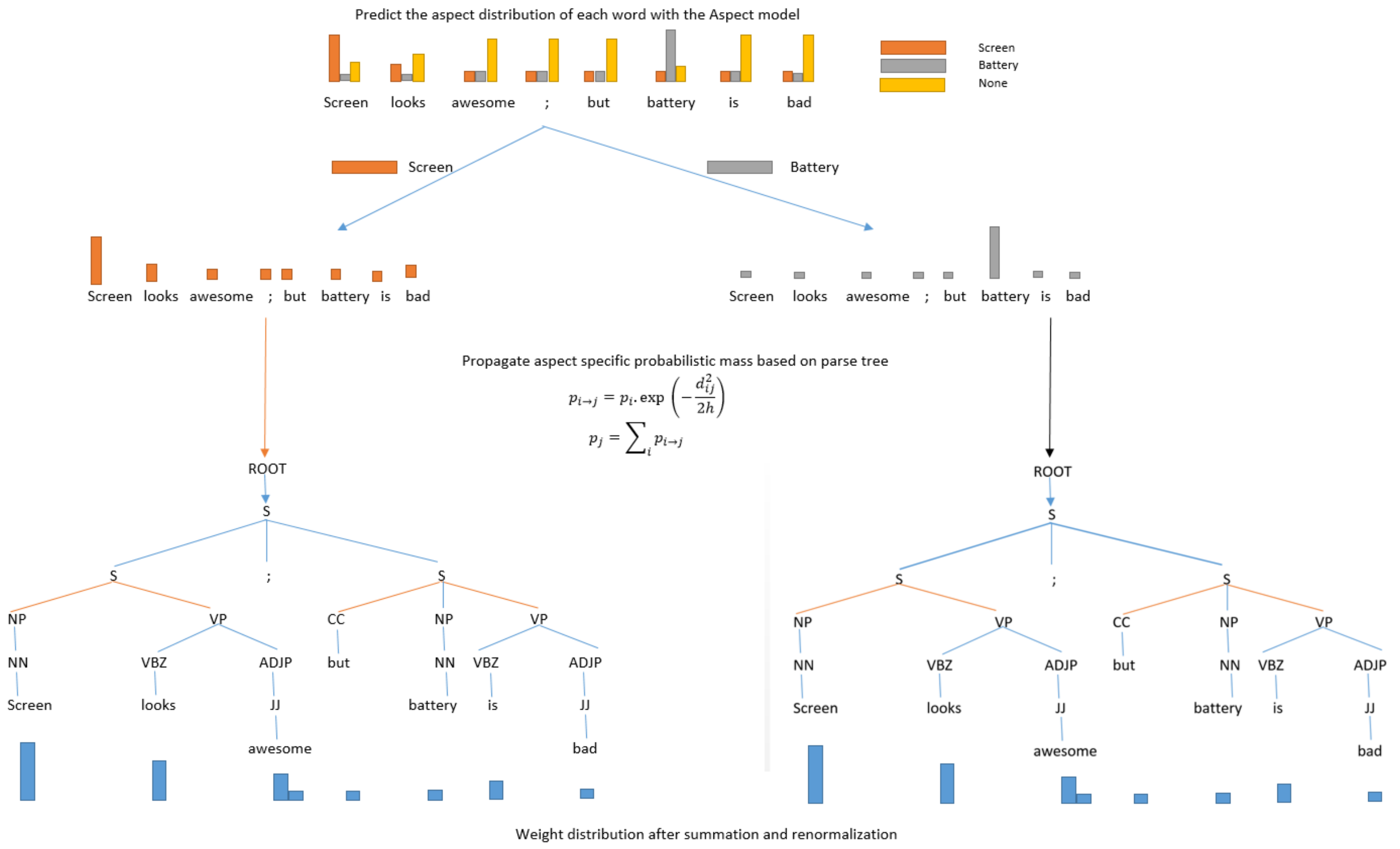

Predict the aspect distribution of each word with the Aspect model
Screen
Battery
None
Screen looks awesome ; but battery is bad
Screen
Battery
Screen looks awesome ; but battery is bad
Screen looks awesome ; but battery is bad
Propagate aspect specific probabilistic mass based on parse tree
$p_{i \to j} = p_i . \exp\left(-\frac{d_{ij}^2}{2h}\right)$
$p_j = \sum_i p_{i \to j}$
ROOT
S
S ; S
NP VP CC NP VP
NN VBZ ADJP but NN VBZ ADJP
Screen looks JJ battery is JJ
awesome bad
ROOT
S
S ; S
NP VP CC NP VP
NN VBZ ADJP but NN VBZ ADJP
Screen looks JJ battery is JJ
awesome bad
Weight distribution after summation and renormalization

Figure 3.18: Aspect-based Sentiment Model[Wang & Liu 2015]

3.2.2.2 Unsupervised Temporal Feature Extraction

This component deals with the identification of the new aspects from the reviews which are categorised in the *'OTHER'* and *'NONE'* aspect in the preprocessing step stated earlier. This is important as aspects may keep changing with time. Some of them may become important, and some of them may become obsolete. To encapture the temporal effect on the aspect, this component has been added in the proposed framework. Here, unsupervised feature extraction approach has been proposed which uses semantic clustering with word vectors in conjunction with TF-IDF score. This also includes word/token meaning along with its frequency. The preprocessing step for this module is to extract various unique nouns present in the reviews by using part of speech tagging and checking whether tagged nouns are present in the dictionary or not, to remove slang words. Nouns are further lemmatised to remove the different tenses present. These nouns are passed as an input to semantic clustering module of the second phase.

3.2.2.2.1 Semantic Clustering

Semantic clustering involves grouping the words that convey the same information or intent. Two essential requirements for semantic clustering are similarity measure and the clustering algorithm. Some of the text similarity measures that can be used for semantic clustering are cosine similarity of TF-IDF vectors, knowledge-based methods, i.e. quantifying the semantic relatedness of words using a semantic network like WordNet and the word embeddings.

Word embeddings have been used for clustering the aspects. The clustering algorithm used here is *k*-means++ clustering [Arthur & Vassilvitskii 2007], which is robust and straightforward technique and have a better allocation of initial cluster centre than the traditional *k*-mean algorithm. The clustering algorithm requires the number of clusters, i.e. the number of different type of new aspects present in the *'OTHER'* aspect which are not known beforehand. The value of *k*, i.e. number of clusters can be calculated using following two ways.

1. Assume *k*= dimension of the sentence (maximum). This is a simple and efficient way to assume the value of *k*. However, it is usually efficient in case of large dataset.

2. Silhouette scores are used to decide k value. This is an efficient technique to calculate the number of clusters.

Here, Silhouette scores method has been used for evaluating the value of k as the dataset used is small. k value is assumed to be in $10 - 30\%$ of the distinct nouns present, and then final value k is decided based on the maximum silhouette score calculated. Higher silhouette score implies that words in the same clusters are more similar to each other than the words of another cluster. After generating clusters, a cluster score is defined for each cluster by equation 3.11.

$$Cluster\_Score = Density * Distance \tag{3.11}$$

$$where, \begin{cases} \text{Density is number of words in the cluster and} \\ \text{Distance is defined as the distance between clus-} \\ \text{ter centre with the nearest cluster centre} \end{cases}$$

A cluster is defined as good cluster if it has a higher cluster score than a threshold value. The threshold value is a hyperparameter which is calculated based on the different cluster score obtained. After extracting good clusters, feature in each cluster is selected based on maximum TF-IDF score obtained after generating TF-IDF score for each feature in that cluster. *TF-IDF* score is calculated as shown in equation 3.12, where $number_of_document$ represents number of the documents in which aspect is appearing.

$$TF - IDFScore = TF * IDF \tag{3.12}$$

$$where, \begin{cases} TF = \log\,(term\_frequency\_in\_all\_documents) \\ IDF = \log\,(\frac{N}{number\_of\_document}) \end{cases}$$

3.2.2.3 Temporal analysis of sentiment and features

This phase is important in the framework to capture changing sentiments about an aspect over time. Considering different possibilities, this phase consists of two subparts.

1. Temporal Analysis of Single Aspect

2. Temporal Analysis of Multiple Aspects

3.2.2.3.1 Temporal Analysis of Single Aspect

There are two types of temporal analysis proposed in the framework, opinion visualization and burst detection.

1. Opinion Visualisation

   An essential part of the framework is to visualise the change in sentiment over the time. The most common way to accomplish the task is to make a two-dimensional line graph with the y-axis as sentiment value and x-axis as time. For a particular aspect, there are three types of visualisation proposed: day wise analysis, year wise analysis and month wise analysis for a particular year.

   - Day wise visualisation

     Given an input of date range, this sub-module visualises the change of sentiment score over days. For each day, the sentiment score is taken as average sentiment score of reviews present in the particular day.

   - Year wise visualisation

     This sub-module visualises the change of sentiment score over the years. For each year, the sentiment score is taken as average sentiment score of reviews present in that particular year.

   - Month-wise analysis for a particular year

     This sub-module takes year as input. There are two types of analysis proposed in this sub-module.

     (a) Analysis for a particular month.

     For a particular month, it extends the day wise visualisation module.

     (b) Month-wise analysis

     This sub-module visualises the change of sentiment score over months. The monthly score is calculated by averaging the sentiment score of review present in that month.

2. Burst analysis

   Burst analysis algorithm has been proposed to detect sudden changes in sentiments score, i.e. burst. This module extends the previous module as the average monthly score for all aspect is calculated along with the number of reviews present for that particular month. If the difference of sentiment score between consecutive months is higher than a threshold value,and number of reviews for these months is higher than a threshold count, burst is detected.

3.2.2.3.2 Temporal Analysis of Multiple Aspects

Temporal analysis of multiple aspects has been analysed in this phase. The preprocessing part of this module involves determining the review score of given aspect list for a year and later for a month. This module supports two types of opinion visualisation.

1. Year wise

   This sub-module visualises the change of sentiment score over the years. For a given aspect, for each year the sentiment score is taken as average sentiment score of reviews present in that year.

2. Month-wise

   This sub-module visualises the change of sentiment score over a month for a given year. For each month, the sentiment score is taken as average sentiment score of reviews present in that month.

3.2.3 Experiments & Results

3.2.3.1 Aspect Model

The data is split randomly into training (80%) and validation set (20%) to select the hyperparameters. The number of epochs (100) that yields the lowest validation loss value is selected as shown in Figure 3.21 and the threshold which gives the highest F-measure on the validation set is used. 'binary_crossentropy' is considered as loss function and 'rmsprop' is used as an optimizer. The threshold value is chosen out to be *0.11* as F-measure is highest at that value as shown in Figure 3.19. With the mentioned parameters evaluation measures obtained are a follows:

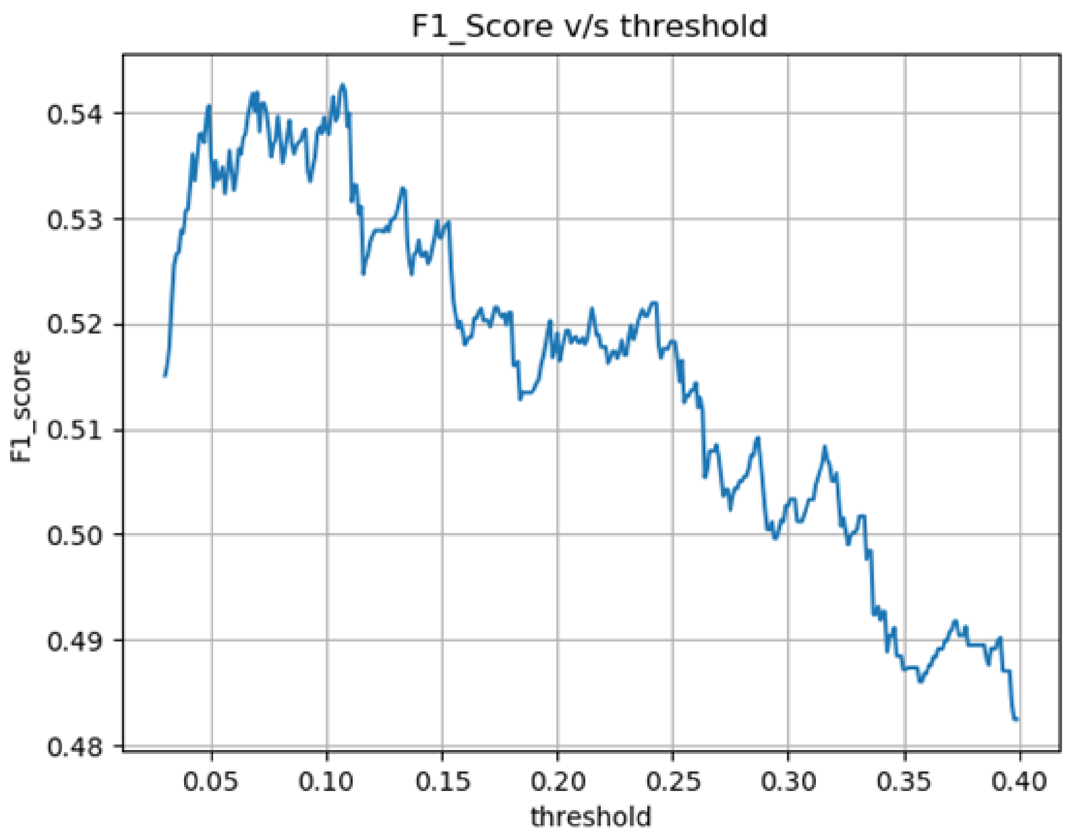


Figure 3.19: F-Measure v/s Threshold for aspect-based model

1. Accuracy:- *96.5%*

2. F-measure:- *54.3%*

3.2.3.2 Sentiment Model

The data is split randomly into training (*80%*) and validation set (*20%*) to select the hyper-parameters, similar to Aspect model. The number of epochs (*100*) that yielded the lowest validation loss value is selected, and the threshold which gives the highest F-measure on the validation set is used. The threshold value is chosen out to be *0.30* as F-measure is highest at that value as shown in Figure 3.20. Evaluation obtained using the parameters are mentioned below.

1. Accuracy:- *83.5%*

2. F-measure:- *60.8%*

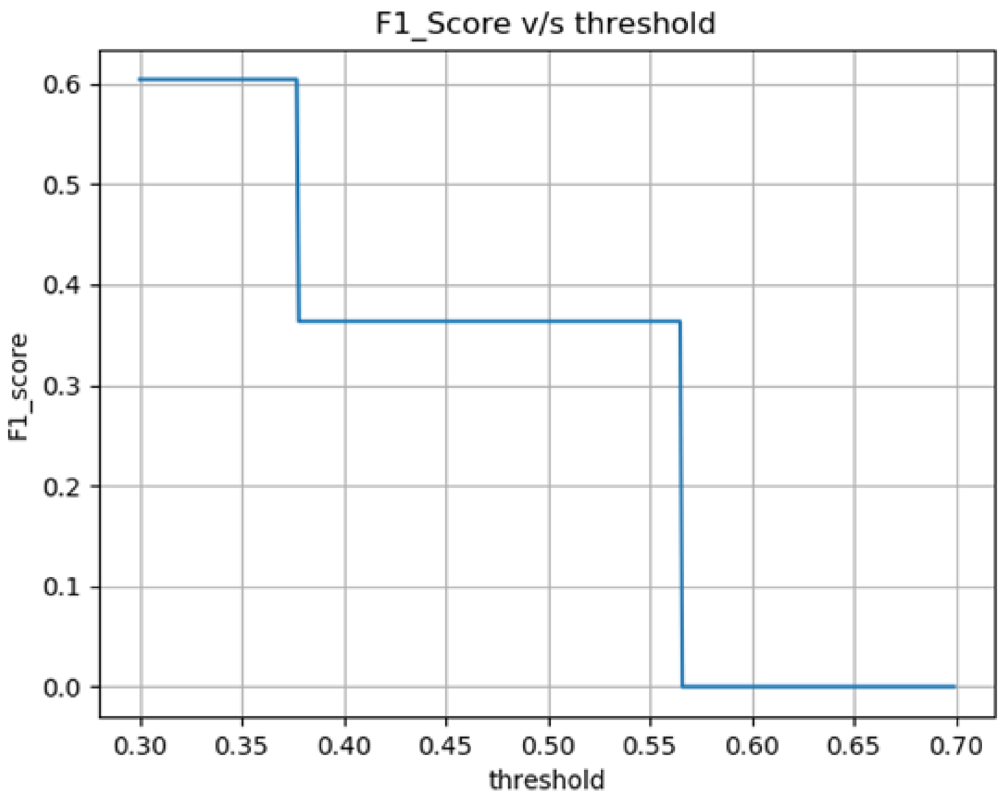


Figure 3.20: F-Measure v/s Threshold for Sentiment Model

3.2.3.3 Unsupervised Feature Extraction

For *k* (value = *83*), silhouette score obtained is *0.2235*. This silhouette score is low because the framework has been evaluated only on single domain dataset in which nouns present are quite similar to each other. Hence, silhouette score obtained is not quite good. The threshold value is a hyperparameter which is calculated as the median of the cluster score. Aspects extracted are (cluster threshold= *35*) shown in Table 3.7. Selected aspect is chosen from the aspect list, fed into the aspect model and aspect model is re-trained.

3.2.3.4 Temporal Opinion Mining for Single Aspect

1. Opinion Visualization

   Figure 3.22 represents the day wise analysis of aspect Laptop#General from the start date (1/1/2008) to end date (6/9/2008). Figure 3.23 represents year wise analysis of Laptop#General. Figure 3.24 represents the month wise analysis for Laptop#General

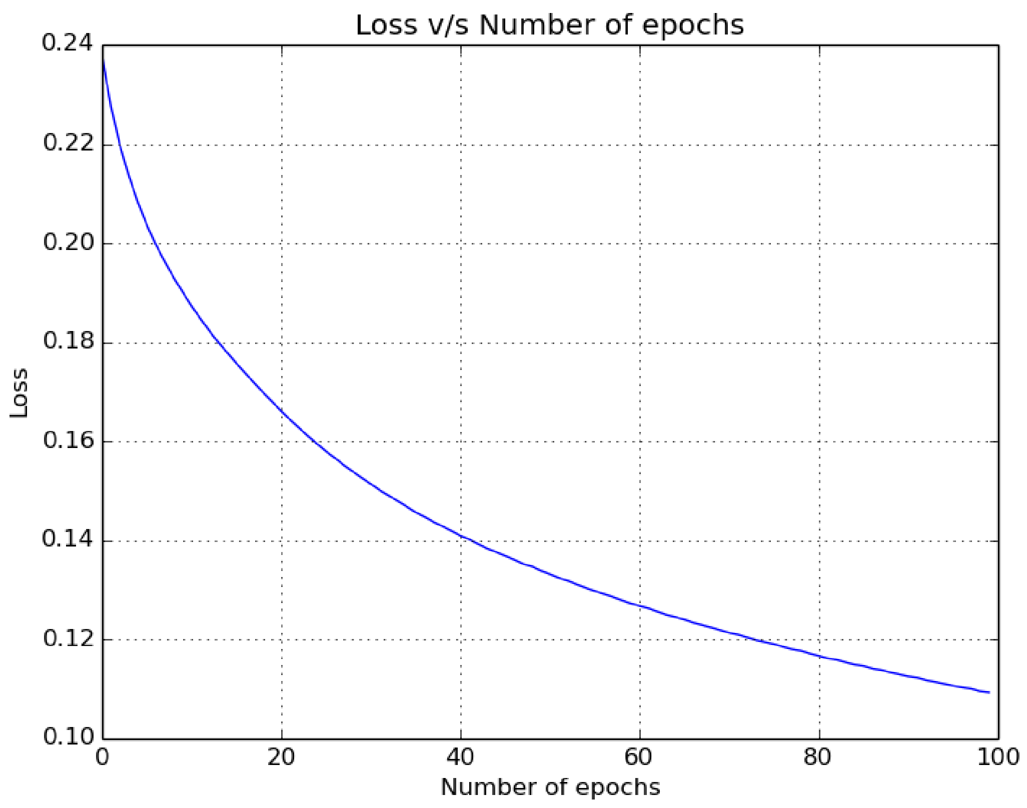


Figure 3.21: Validation Loss v/s Number of epochs

Table 3.7: Aspects extracted

S.No	Aspect	TF_IDF_SCORE
1	graphic	38.5514606589
2	office	27.0335962363
3	power	43.3703932413
4	thing	66.9271010503
5	game	30.4214820639
6	time	86.3525235395

aspect in the year 2008 whereas Figure 3.25 represents the analysis for a particular month (February) in the year 2008.

3.2.3.5 Temporal Opinion Mining for multiple aspects

Figure 3.26 represents year wise analysis for multiple aspects shown whereas Figure 3.28 represents the month wise analysis for the year 2009.

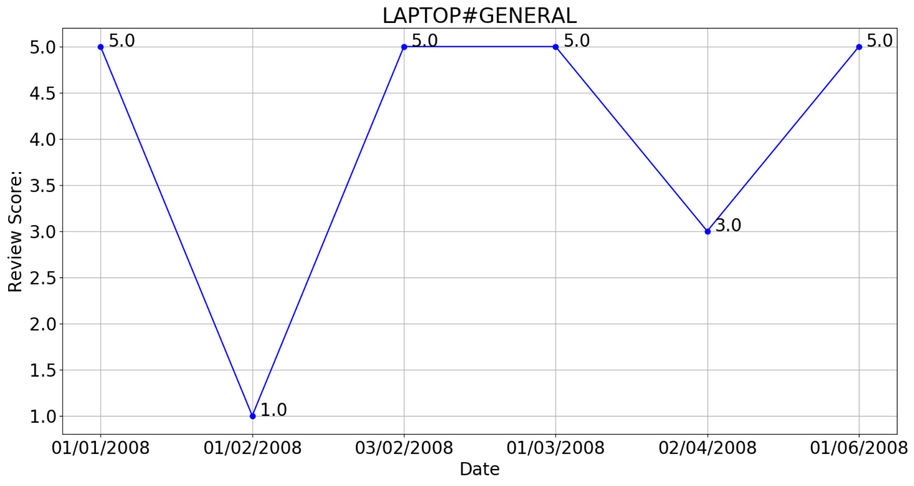


Figure 3.22: Day Wise Analysis of Laptop#General

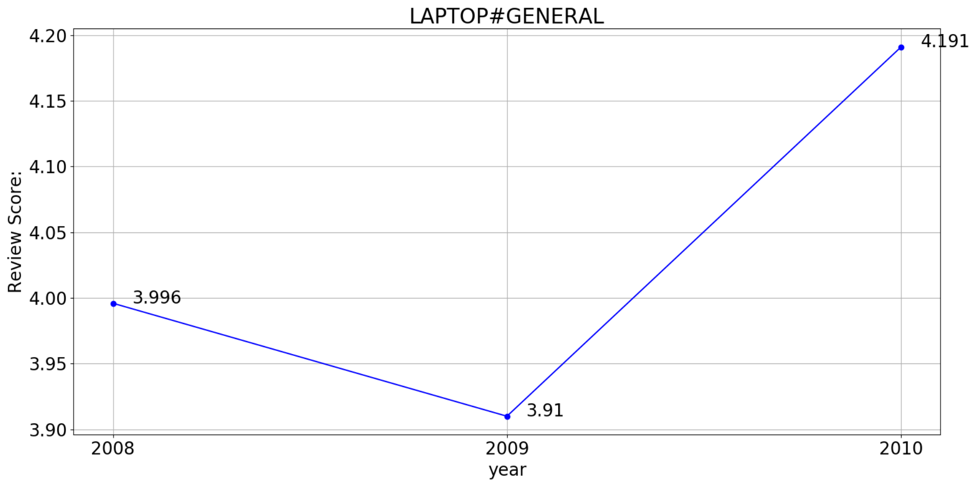


Figure 3.23: Year-wise Analysis of Laptop#General

3.2.3.6 Burst Analysis

Figure 3.27 represents burst obtained with sentiment_threshold:*1.0* and count_threshold:*10*. Burst is obtained for Laptop#General aspect as this aspect covers about *60%* of the dataset. Table 3.8 elaborates the burst obtained.

As from June to July, the difference between the sentiment score is more than the sentiment threshold and review count for each month is higher than count threshold and

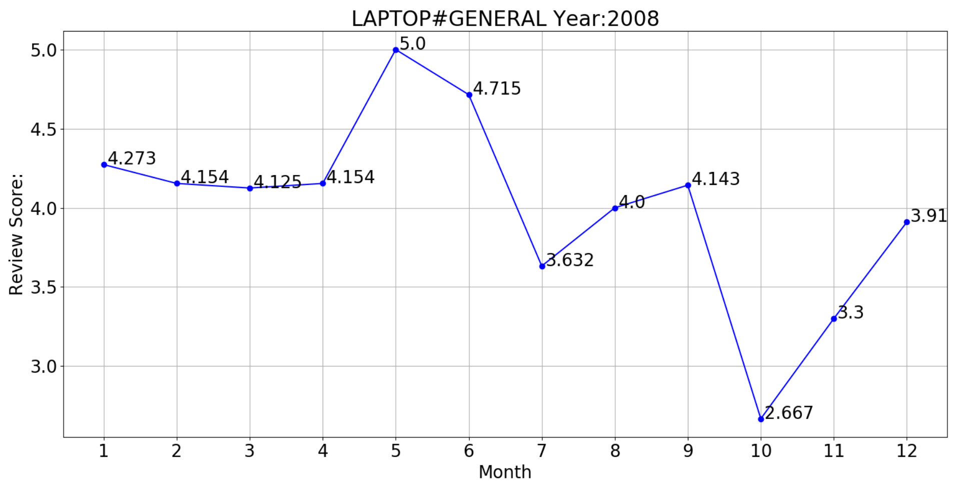


Figure 3.24: Month-wise analysis for Laptop#General in the Year 2008

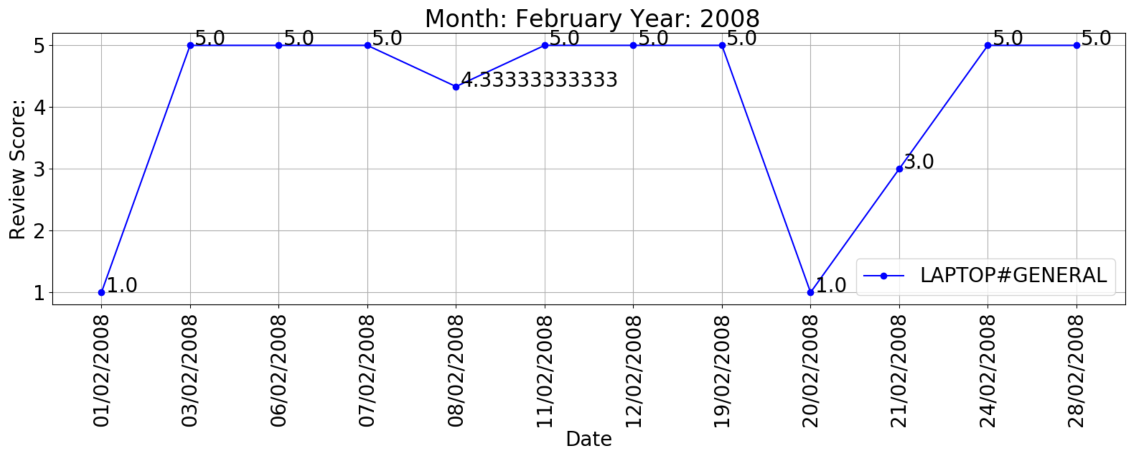


Figure 3.25: Analysis for a particular month

Table 3.8: Burst obtained

Aspect	Duration
Laptop#General	June/2009-July/2009
Laptop#General	September/2009- October/2009
Laptop#General	October/2009- November/2009
Laptop#General	January/2010-February/2010

hence, burst is detected. Similarly, from July to August the sentiment score difference is also higher than the threshold, but the review count for August is less than count threshold, so it is not a burst.

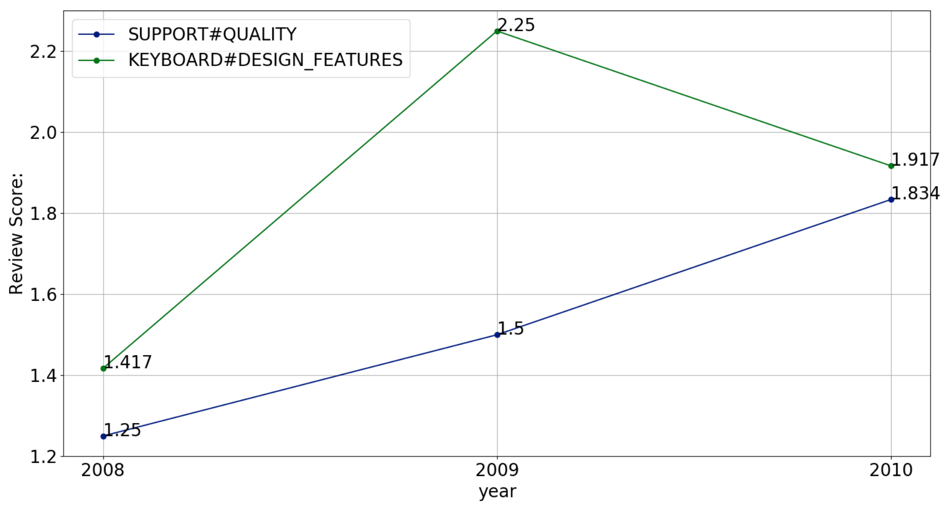


Figure 3.26: Year-wise analysis of multiple aspects

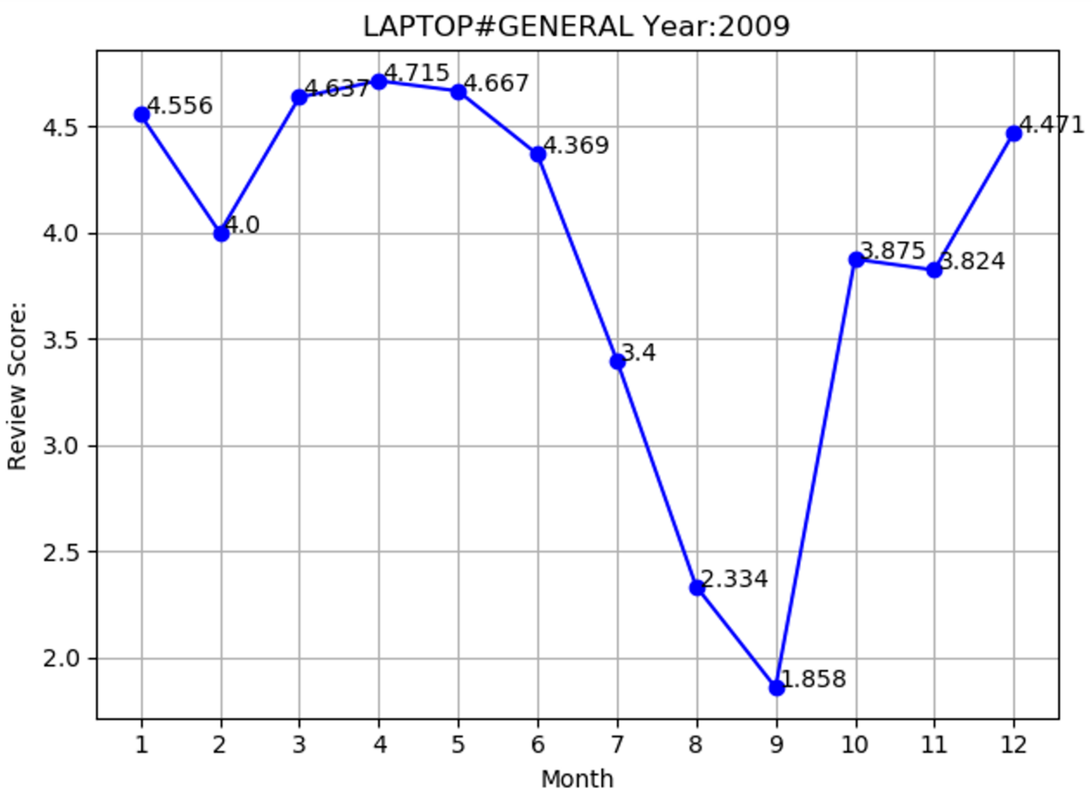


Figure 3.27: Burst Analysis

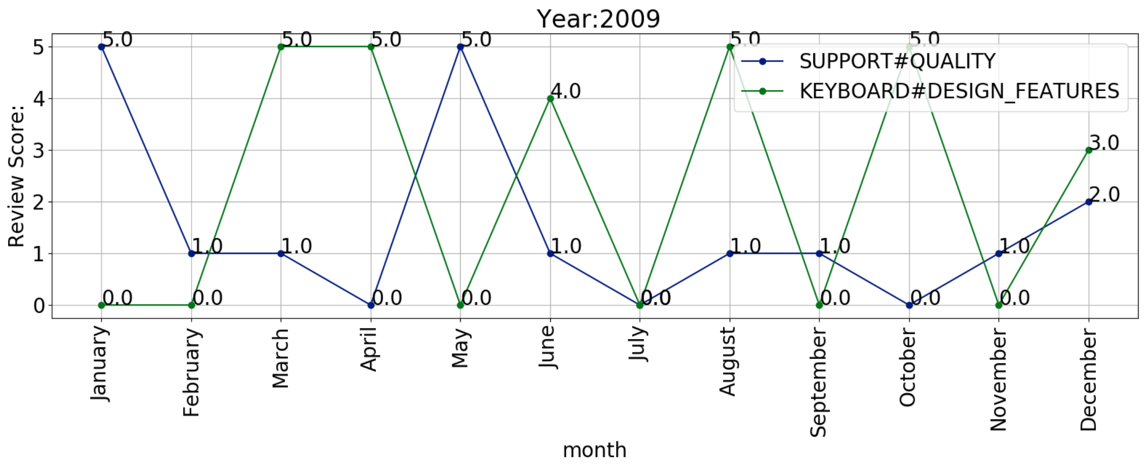


Figure 3.28: Month-wise analysis for the year 2009 for multiple aspects

3.3 Concluding Remarks

The approach in this chapter presents sentiment analysis of SAIL 2015 data consisting of monolingual tweets in one of the three Indian languages (Hindi, Bengali and Tamil). Word embeddings have been used as input to the Sequential model. The approach investigates *39* different hybrid models of neural network layers (RNN, CNN and LSTM). The highest accuracy has been obtained using CNN (*77.63%*) and RNN-LSTM (*77.63%*) on Hindi dataset, LSTM-CNN (*57.37%*) on Bengali dataset and RNN-LSTM-RNN (*69.19%*) on Tamil dataset. The results have been carefully scrutinized to conclude that as the complexity of the text in the dataset increases, accuracy decreases for each model. Models with more number of hidden layers provide better accuracy for more complex text as the increase in the number of hidden layers adds to the non-linearity of the model to predict complex situations. It has also been observed that neural networks perform better than other traditional machine learning techniques such as SVM and decision trees in predicting the polarity of a sentence.

In current scenario, pre-processing of the textual data is done to remove the emoticons, exclamations. However, they may be playing a role in determining the sentiment of the sentence. For example: 'What!' and 'What?' represent two different meanings which can't be distinguished using present day sentiment analysis approaches as they tend to pre-process the sentences to remove the punctuation. Few users use only emoticons to express their sentiments which can be embedded and worked upon for making the sentiment analysis approaches even more reliable. This work can also be extended to classify sentences into multiple classes such as extremely negative, negative, neutral, positive and extremely positive, which gives a more precise sentiment to a sentence.

A framework for temporal sentiment analysis has also been proposed. The framework is divided into three parts: aspect model, temporal feature extraction and temporal analysis. Aspect model is used for identifying the aspects present in the reviews. The model is tuned and parameters are determined using the validation data. The CNN based sentiment model predicts the polarity of the sentiments for each of the aspects, identified using static pre-trained word vectors for improving the performance and decreasing overfitting. Aspects are updated from time to time to incorporate the aspect trends that change with time using semantic clustering. Temporal analysis is done by dividing data into each segment, as specified by input (day, month and year) and plotting the resultant data on two dimensional graphs. Yearly, monthly and day wise analysis are shown for individual aspect as well as for the group of aspects. The individual graph shows the variation of sentiment score with respect to time. Burst analysis has been done which extracts the aspects for which there is a sudden variation of sentiment score in the consecutive months.

Chapter 4

Text Summarization

Post the advent of the world wide web the amount of data and information that is accessible has increased tremendously. The extent of information is such that it has now become practically impossible for any single entity to process all the data and summarize it. Consumers are disinterested in reading a long piece of text and usually tend to skip important portions. Given this scenario, the need for automated text summarization arose. Text summarization is condensing of text such that, redundant data is removed and important information is extracted and represented in a concise manner. With the explosion of the abundant data present on social media, it has become important to analyze the text for seeking information and use it to the advantage of various applications and people. From past few years, the task of automatic summarization has stirred the interest among communities of natural language processing and text mining. The technique of text summarization can be classified into two major categories abstractive and extractive. Another possible classification for text summarization is single [Litvak & Last 2008] vs multi-document [Barzilay *et al.* 1999] summarization and mono-lingual vs multi-lingual summarization [Radev *et al.* 2004].

Abstractive text summarization aims to achieve the task of generating summaries closer to human generated summaries and present the gist of the text. It involves using generative approaches which can produce meaningful sentences and at the same time preserve the semantics of the original text. It is viewed as a highly difficult problem to solve and many new approaches [Nallapati *et al.* 2016][Lopyrev 2015] are being proposed.

Extractive text summarization is a robust way of generating summaries by selecting salient sentences from the given text and presenting it to the user. Each sentence is attached with some sort of a saliency score and highest scored sentences are chosen to be the part of the summary. This is relatively simpler in contrast to abstractive summaries which involve generating phrases and words, organizing them to form meaningful sentences and at the same time presenting an interpreted gist of the text. It would involve a high degree of natural language processing and hence, is a much more difficult task.

This chapter aims to summarize important extracts and reduce redundancy. In section 4.1 two approaches for paraphrase detection has been explained to identify similar sentences. Two approaches for extractive text summarization are discussed in section 4.2 using hybrid scoring based technique and CNN(s) respectively. In section 4.3 two approaches are discussed for abstractive text summarization where the first approach infuses sentences based on sentiments and second approach uses a generative adversarial network.

4.1 Paraphrase Detection

Paraphrase detection is the task of determining if two sentences convey the same meaning where sentences need not be of the same length. Paraphrase identification has applications in question answering [Duboue & Chu-Carroll 2006], [Fader *et al.* 2013], information retrieval, text summarization [Barzilay 2003], plagiarism detection, semantic parsing [Berant & Liang 2014] etc. Plagiarized texts usually copy phrases as it is or replace some words with similar words. Paraphrase detection will help in detecting plagiarized work and ensure that the documents written are unique and not copied. Question answering system uses paraphrases to find the appropriate answers to question queried. A lot of work has been done in paraphrase detection for English language [Vo *et al.* 2015] [Sundaram *et al.* 2009] [Yin & Schütze 2015]. However, for Hindi and other Indian languages, not much work has been done for paraphrase detection. In literature, paraphrase detection has been modelled as a classification problem.

Paraphrase detection can be a useful tool for reducing redundancy in text summarization. For example, one would want to avoid sentences like the following:

- *Amrozi accused his brother, whom he called the witness, of deliberately distorting the evidence.*

- *Referring to him as the only witness, Amrozi accused his brother of deliberately distorting his evidence.*

In simple lexical matching approaches, words with similar meanings are not taken into account. Various tools for measuring the similarity between word pairs are available and WordNet [Leacock & Chodorow 1998]; [Mihalcea *et al.* 2006] is one of the most popular resource used by the researchers. Mreover, similarity amongst word pairs does not imply that the sentences will be paraphrases. In WordNet based measures, correct sense of words might not necessarily be present in candidate paraphrases.

Machine learning based approaches have used dependency parser [Malakasiotis 2009] and lesk algorithm [Mihalcea *et al.* 2006] for paraphrase detection. Indian languages lack efficient resources such as named entity taggers and POS taggers, which are typically used for feature detection in machine learning based approaches. Moreover, lack of annotated datasets has hindered developing state of the art approaches for paraphrase detection in Indian languages. Deep learning techniques such as recursive auto-encoders [Socher *et al.* 2011] and CNN(s) [Yin & Schütze 2015], [He *et al.* 2015] have been explored for monolingual paraphrase detection in foreign languages but to the best of our knowledge, no attempt has been made for identification of paraphrases in Indian languages using deep learning based approaches.

Two approaches have been proposed for identifying paraphrases in Indian languages. First approach presented in section 4.1.1 is based on traditional machine learning techniques and uses features such as Soundex, POS tags and stemming. Second approach explained in section 4.1.2 proposes neural network models based on CNN and RNN to identify the paraphrases.

4.1.1 Hybrid Approach for Paraphrase Detection

The proposed approach HAPD (Hybrid Approach for Paraphrase Detection) has been divided into multiple phases as shown in Figure 4.1. The first phase processes the training data to extract important features. The following three features are extracted for the proposed training model:

1. **POS tags:** POS tags are labels that are given to words to identify the part of speech or lexical categories of words. The eight parts of speech are verb, noun, pronoun, adjective, adverb, preposition, conjunction and interjection. Words that have the same POS tags play similar roles in the grammatical structure of sentences. RDRPOSTagger[1] [Nguyen *et al.* 2014] is used for POS tagging of Hindi words. RDRPOSTagger takes the pair of sentences as input and generates the respective POS tags next to each word. POS tags corresponding to each word in the sentence are extracted from the output and appended to form a string of POS tags for each sentence.

2. **Stem of the words:** Stemming is a process of extracting the 'word stem' or 'root' of the word. For extracting the stem of Hindi words, a Hindi stemmer[2] is used, which implements the suffix-stripping algorithm described in [Ramanathan & Rao 2003]. A string for each sentence with the corresponding stems of the Hindi words is obtained.

3. **Soundex codes:** Soundex is a phonetic algorithm for indexing names by sound as pronounced in English. Soundex[3] provides an implementation of the modified version of Soundex algorithm for Indian languages including Hindi. A string comprising of Soundex codes corresponding to each sentence is generated using soundex codes for each word.

After extracting the above features, the similarity scores (of both the sentences) corresponding to each feature is calculated. Each similarity score lies in the range [*0,1*] and uses Levenshtein distance to calculate the differences between string sequences. The Levenshtein distance[4] between two words is the minimum number of single-character edits (i.e. insertions, deletions or substitutions) required to change one word to the other. The similarity score is calculated for each pair of sentences with POS tags (feature 1), sentences with the stem of the words (feature *2*) and sentences with Soundex codes corresponding to the Hindi words (feature *3*). Finally a feature vector is created with the similarity scores corresponding to the sentence pair.

[1]https://rdrpostagger.sourceforge.net
[2]http://research.variancia.com/hindi_stemmer/
[3]https://pypi.python.org/pypi/soundex/
[4]https://pypi.python.org/pypi/fuzzywuzzy

After feature vector generation, different traditional machine learning techniques are used for training so that the best model for predicting the labels could be chosen after analysis.

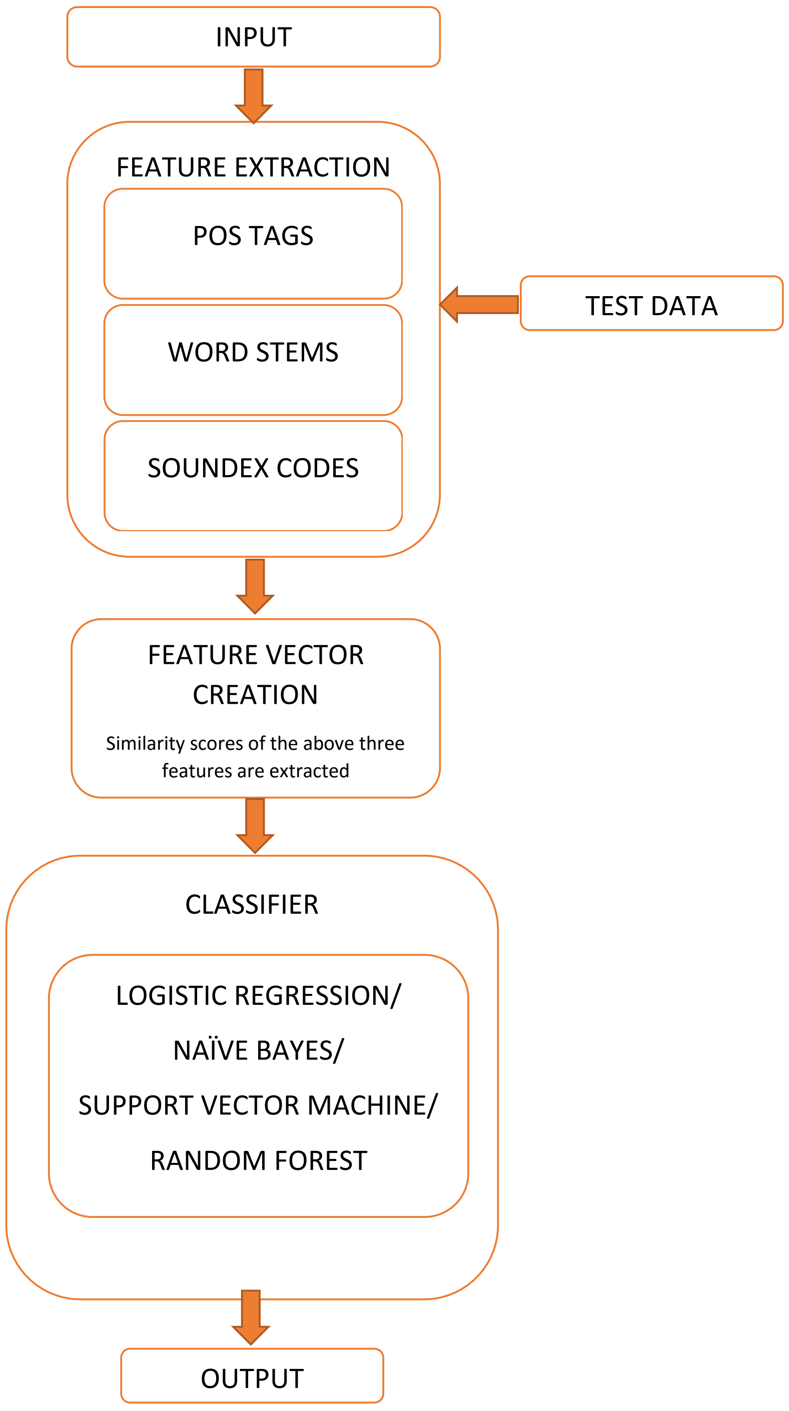


Figure 4.1: Block diagram for Paraphrase Detection

4.1.1.1 HAPD Algorithm

Proposed algorithm, HAPD takes the pair of sentences as input where each pair (*PS[i]*) contains two Hindi sentences (*PS[i].Sentence*) and outputs a label for its corresponding paraphrases. Each function (*PosTags*, *WordStem* and *Soundex*) take a pair of a sentence (*PS*) as its parameter and return the array of corresponding POS tagged sentences, WordStem sentences and sentences with Soundex codes respectively. *SimilarityScore* function generates the similarity score for each of its input array. *SimScore1*, *SimScore2*, and *SimScore3* are the individual vectors for the three features, which are then passed to the *CreateVector* function to form the final *FeatureVector*. *Classifier* function takes the *FeatureVector* as input, assigns labels to the Pair (*PS*) and then returns a *LabelVector*. *Classifier* function implements different models (*Logistic Regression, Naive Bayes, Support Vector Machine and Random Forest*) for predicting labels.

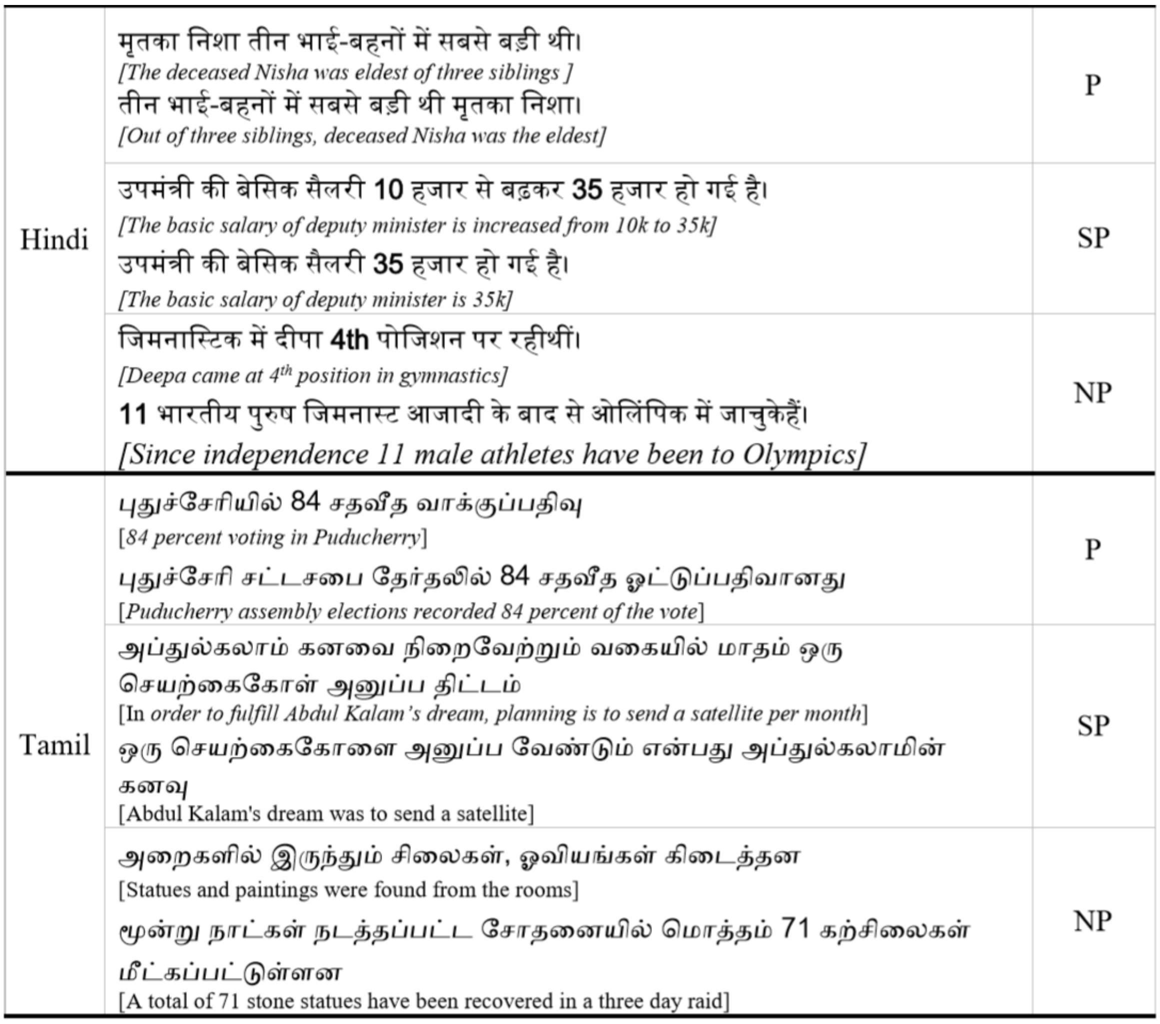
Algorithm 4.1: Algorithm for Detecting paraphrases

```
input : Paraphrase P, where all paraphrases have a unique id and contains two
        sentences (Hindi)
output: LabelVector gives the corresponding labels for the paraphrases.
        Depending upon the task it can have value of P, NP and SP
1 Initialization: SimScore1[]=0,SimScore2[]=0,SimScore3[]=0;
2 for i ← 0 to PS.Count do
3     Pos[]=PosTags (PS[i].Sentence);
4     Stem[]=WordStem (PS[i].Sentence);
5     Sound[]=Soundex (PS[i].Sentence);
6     SimScore1.append (SimilarityScore(Pos[]));
7     SimScore2.append (SimilarityScore(Stem[])) ;
8     SimScore3.append (SimilarityScore(Sound[])) ;
9 FeatureVector=CreateVector(SimScore1, SimScore2, SimScore3)
   LabelVector=Classifier(FeatureVector)
```

4.1.1.2 Data Analysis

The dataset [Anand Kumar *et al.* 2016] is from newspaper domain and contains pairs of sentences. Dataset has two versions, first version (Dataset$_1$) of dataset has the pair of sentences labelled with P and NP whereas second version (Dataset$_2$) of dataset contained sentences classified as P, NP and SP. Example for Paraphrase (P), Not Paraphrases (NP) and Semi Paraphrases (SP) are shown in Figure 4.2 for Hindi and Tamil languages.

4.1.1.2.1 Dataset$_1$ The pair of sentences in the training dataset contains *1000* 'Paraphrases' (*P*) and *1500* 'Not Paraphrases' (*NP*). Test dataset for Dataset$_1$ consisted of *900* pairs for the Hindi Language. The number of paraphrases with different number of common words is shown in Figure 4.3. For example, a point *(5,72)* represents *72* paraphrases which have five common words.



Hindi	मृतका निशा तीन भाई-बहनों में सबसे बड़ी थी। *[The deceased Nisha was eldest of three siblings]* तीन भाई-बहनों में सबसे बड़ी थी मृतका निशा। *[Out of three siblings, deceased Nisha was the eldest]*	P
	उपमंत्री की बेसिक सैलरी **10** हजार से बढ़कर **35** हजार हो गई है। *[The basic salary of deputy minister is increased from 10k to 35k]* उपमंत्री की बेसिक सैलरी **35** हजार हो गई है। *[The basic salary of deputy minister is 35k]*	SP
	जिमनास्टिक में दीपा **4th** पोजिशन पर रहीथीं। *[Deepa came at 4th position in gymnastics]* **11** भारतीय पुरुष जिमनास्ट आजादी के बाद से ओलिंपिक में जाचुकेहैं। *[Since independence 11 male athletes have been to Olympics]*	NP
Tamil	புதுச்சேரியில் 84 சதவீத வாக்குப்பதிவு [*84 percent voting in Puducherry*] புதுச்சேரி சட்டசபை தேர்தலில் 84 சதவீத ஓட்டுப்பதிவானது [*Puducherry assembly elections recorded 84 percent of the vote*]	P
	அப்துல்கலாம் கனவை நிறைவேற்றும் வகையில் மாதம் ஒரு செயற்கைகோள் அனுப்ப திட்டம் [In *order to fulfill Abdul Kalam's dream, planning is to send a satellite per month*] ஒரு செயற்கைகோளை அனுப்ப வேண்டும் என்பது அப்துல்கலாமின் கனவு [Abdul Kalam's dream was to send a satellite]	SP
	அறைகளில் இருந்தும் சிலைகள், ஓவியங்கள் கிடைத்தன [Statues and paintings were found from the rooms] மூன்று நாட்கள் நடத்தப்பட்ட சோதனையில் மொத்தம் 71 கற்சிலைகள் மீட்கப்பட்டுள்ளன [A total of 71 stone statues have been recovered in a three day raid]	NP

Figure 4.2: Example for Paraphrase, Not Paraphrase and Semi Paraphrase

4.1.1.2.2 Dataset$_2$ In Dataset$_2$, training dataset consisted of *1000* pairs of sentences that are Paraphrases (*P*), *1000* pairs that are Semi-Paraphrases (*SP*) and *1500* that are Not Paraphrases (*NP*). For test dataset, *1400* pairs of Hindi sentences are provided. The number of Paraphrases and Semi-Paraphrases with common words versus the number of common words is shown in Figure 4.4.

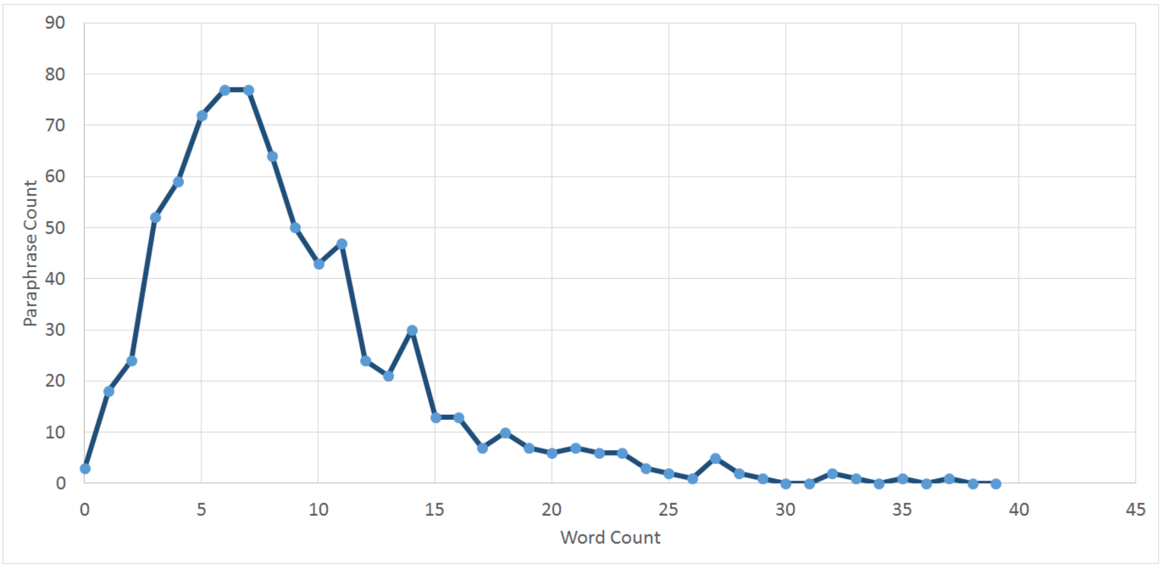


Figure 4.3: Data Analysis of Paraphrase for Dataset$_1$

4.1.1.3 Experiments and Results

To test the accuracy and F-measure, training dataset is divided into a ratio of *75%* and *25%* for training and testing respectively. The results (accuracy and F-measure) are evaluated using sklearn [Pedregosa *et al.* 2011b] for the different models (Logistic Regression, Naive Bayes, Support Vector Machine and Random Forest). Results obtained for Dataset$_1$ are shown in Figure 4.5. The proposed system gave an accuracy of *90.4%* and F-measure *87.6%* for Logistic Regression followed by Naive Bayes and Random Forest, both with *89.5%* accuracy. Logistic Regression performs better than other traditional machine learning techniques in case of binary classification because it assigns labels by calculating odds ratio and then applies a non-linear log transformation. Moreover, the performance can be fine-tuned by changing and adjusting parameters in the functions provided by sklearn [Pedregosa *et al.* 2011b] for Logistic Regression. As Dataset$_1$ is a binary classification problem and hence, results obtained by Logistic Regression are better than others. On the other hand, Dataset$_2$ is a multi-class classification problem (labels-*P, NP or SP*). Hence, in this case, the Random Forest gave the best results with *69.2%* accuracy and *68.8%* F-measure followed by Naive Bayes (*64.6%* accuracy and *62.4%* F-measure) as shown in Figure 4.6. Random Forest calculates labels by using subsamples of the dataset and uses averaging to improve the accuracy whereas Naive Bayes uses a conditional probability approach for assigning labels.

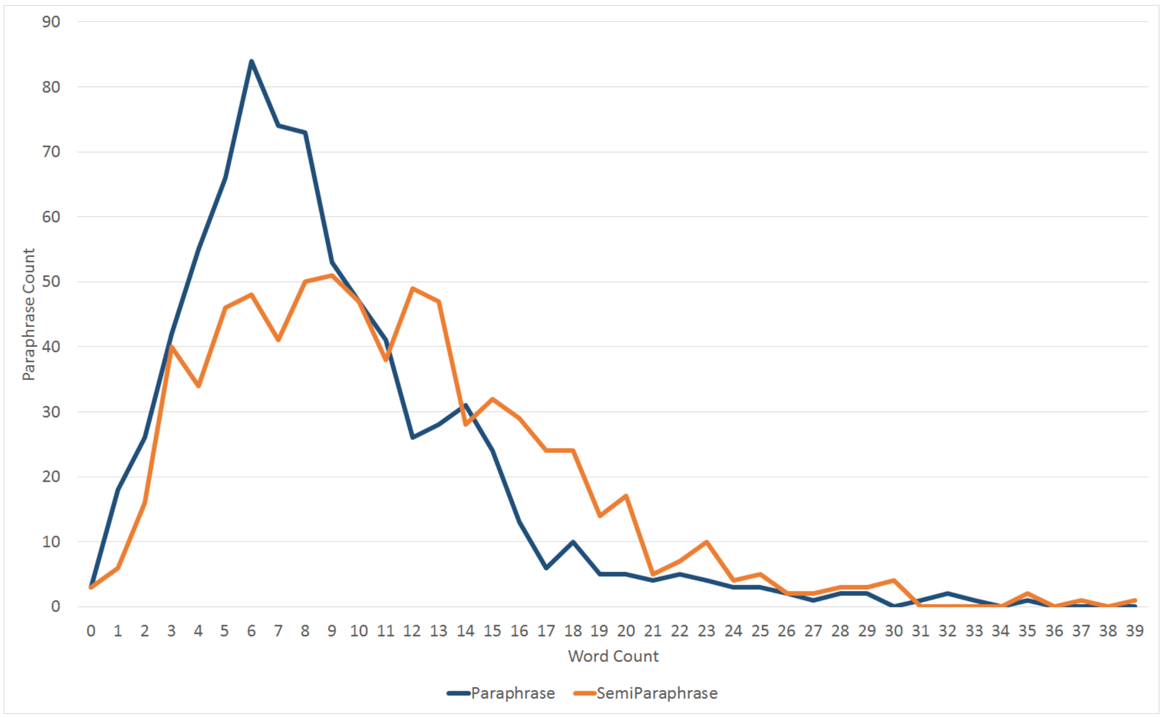


Figure 4.4: Data Analysis of Paraphrase and Semi Paraphrase for Dataset$_2$

Logistic regression and Random forest are chosen as the final classifier for Dataset$_1$ and Dataset$_2$ respectively. For testing data, an accuracy of *0.897* and F-measure of *0.89* are obtained as shown in Figure 4.7 for Dataset$_1$. In Dataset$_2$, the proposed technique obtained accuracy and F-measure of *0.717* and *0.712* as shown in Figure 4.8 respectively for the finalized proposed system.

4.1.1.3.1 Error Analysis Errors occurred in classification can be attributed to the following:

1. [Nguyen *et al.* 2014] states that the RDRPOSTagger achieves a very competitive accuracy in comparison to the state-of-the-art results but a different Hindi POS tagger can also be used. RDRPOSTagger can also be combined with an external initial tagger to increase its accuracy.

2. Similarly, the Hindi stemmer used might have incorrectly returned the stem words, which can be a reason for wrongly classified Paraphrases. The algorithm for extracting the root words can be improved further to better the results.

3. Other factors that could have led to errors are the accuracy of Soundex library and similarity measure used.

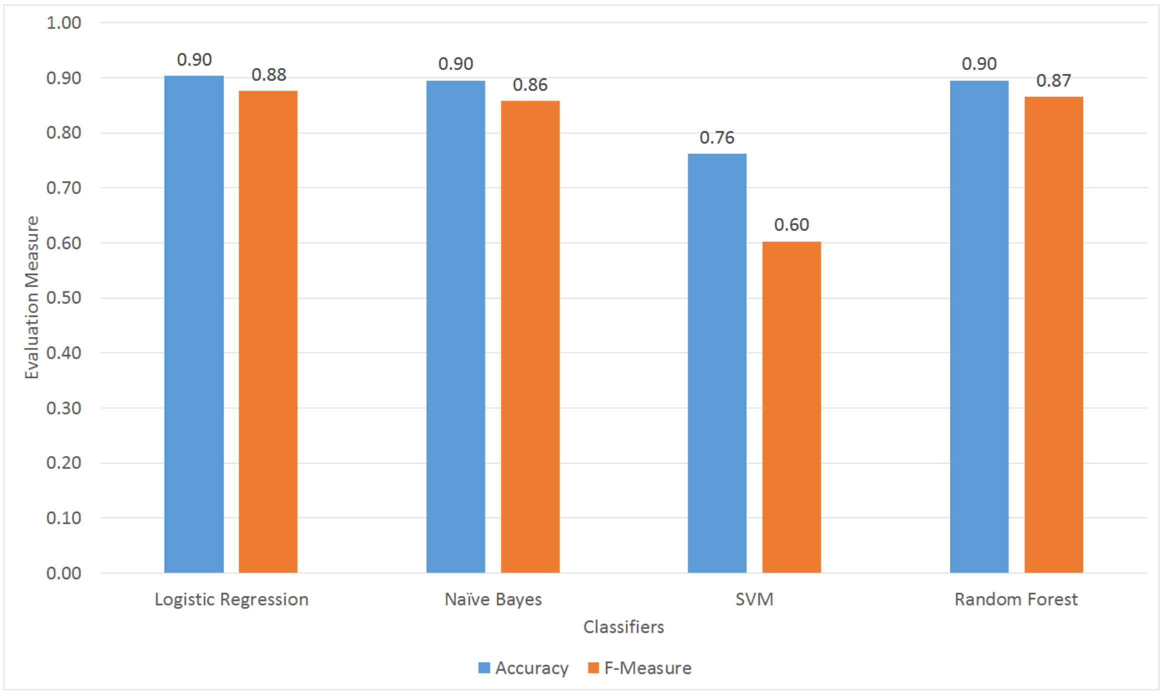


Figure 4.5: Results for Dataset$_1$ using different classifier for proposed system

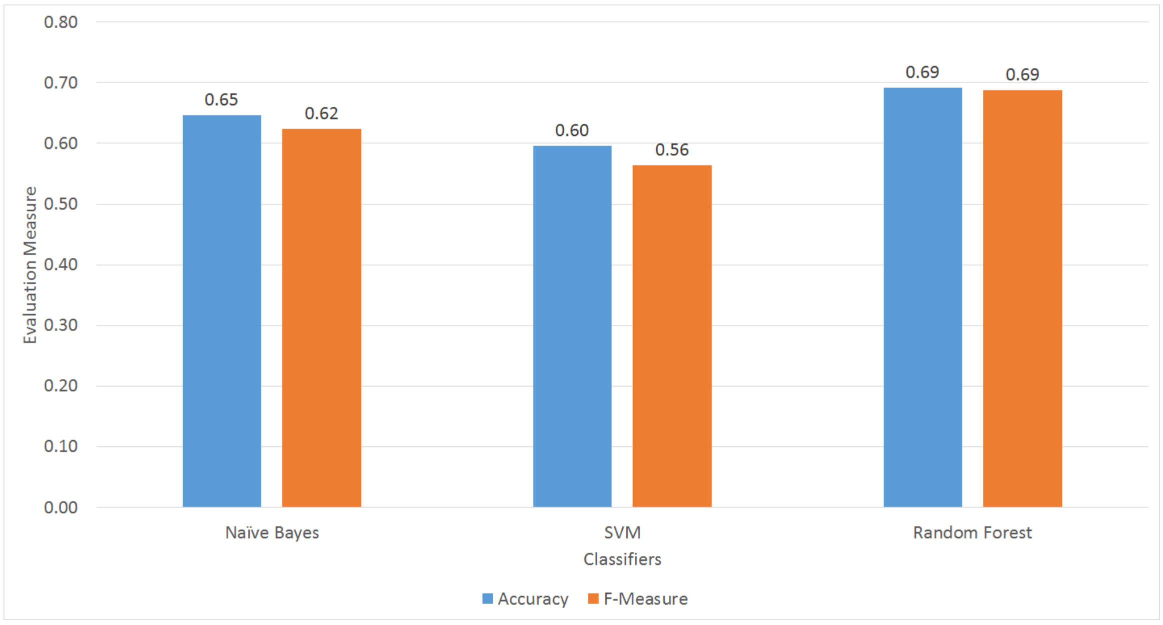


Figure 4.6: Results for Dataset$_2$ using different classifier for proposed system

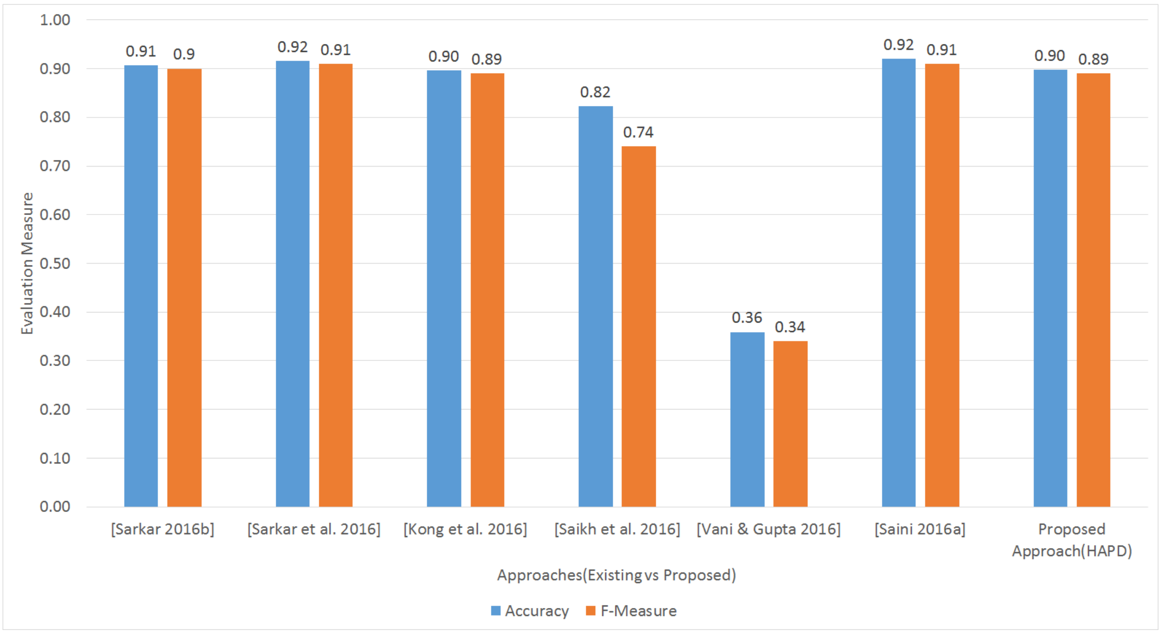


Figure 4.7: Comparison of proposed approach with existing approaches on Dataset$_1$

4.1.2 Detecting Paraphrase using Deep Learning in Indian Languages

Different approaches are proposed for paraphrase detection. Approaches are tested and analyzed for English, Hindi, Malayalam, Punjabi and Tamil languages. Microsoft research paraphrase corpus introduced by [Dolan *et al.* 2004] is used for English. DPIL@ FIRE 2016 [Anand Kumar *et al.* 2016] dataset is used for Indian languages.

4.1.2.1 Preprocessing

1. Class labels and sentences are separated from the data. Stemming and case conversion is performed to normalize the text.

2. Two types of input matrices are prepared as the input to the deep learning models. The first type is used as an input to both CNN and RNN. The second type is used as an input to CNN-WordNet.

   (a) Input$_1$

      i. The pair of sentences to be classified are appended together. The sentence is then padded to the maximum sentence length. Padding sentences to the same length are useful because it allows to efficiently batch data since each example in a batch must be of the same length.

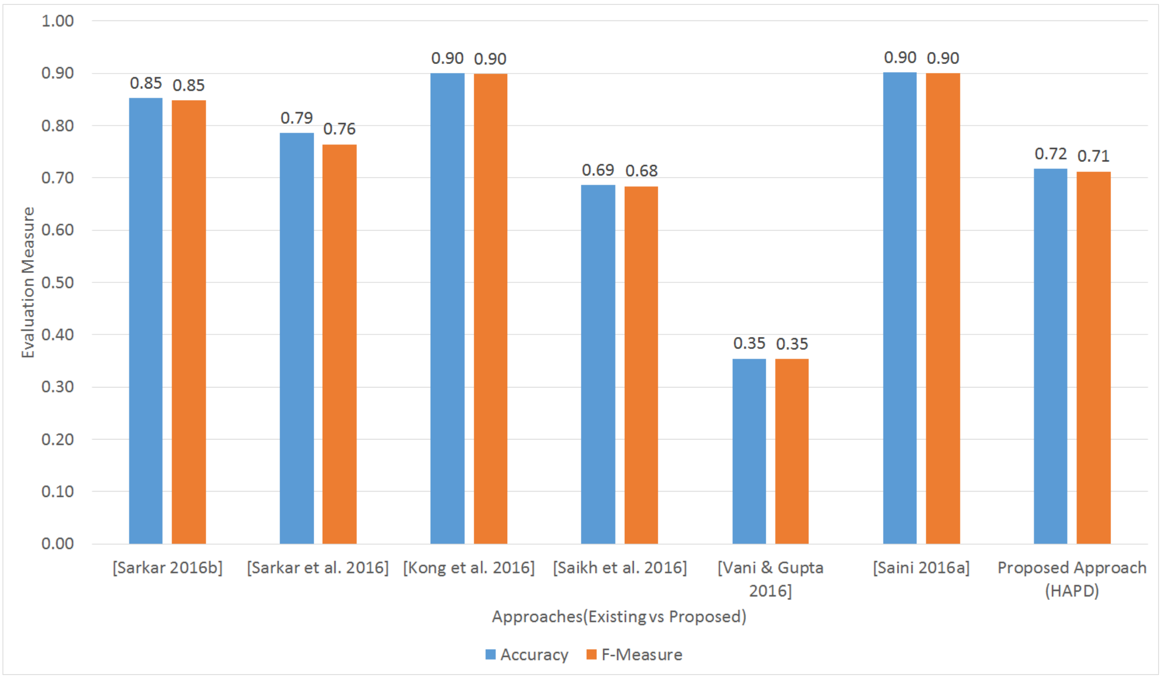


Figure 4.8: Comparison of proposed work with existing work on Dataset$_2$

$$\begin{bmatrix}
 & amrozi & accused & brother & called & witness & deliberately & distorting & evidence \\
referring & 0 & 0.222 & 0 & 0 & 0 & 0 & 0 & 0 \\
witness & 0 & 0 & 0 & 0.545 & 1.0 & 0 & 0 & 0.142 \\
amrozi & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
accused & 0 & 1 & 0 & 0.142 & 0.142 & 0.333 & 0.222 & 0.181 \\
brother & 0 & 0 & 1 & 0.6 & 0.6 & 0 & 0 & 0.166 \\
deliberately & 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 \\
distorting & 0 & 0 & 0 & 0.133 & 0.133 & 0.285 & 1.0 & 0.166 \\
evidence & 0 & 0 & 0 & 0.142 & 0.142 & 0 & 0 & 1.0
\end{bmatrix}$$

Figure 4.9: Matrix built using WordNet

ii. For English, word2vec is used to generate input matrix for the CNN and RNN. These vectors are pre-trained by [Mikolov *et al.* 2013b] on Google news (*100* billion words)[1]. For Indian languages, one hot vector encoding is used instead of word2vec (since no pre-trained vectors are available for these languages).

(b) Input$_2$

i. The pair of sentences to be classified are padded individually to the maximum length. A matrix (as shown in Figure 4.9) is then built where WordNet is used to calculate the similarity between the two words. For Indian languages, the sentences are translated into English (using Google Trans-

[1]https://code.google.com/p/word2vec/

late API[1]) to find the WordNet similarity between two words. In CNN-WordNet (section 4.1.2.2), this matrix is used as input to CNN.

4.1.2.2 Approach based on Convolutional Neural Network

Two variants of approach are proposed based on CNN, where both of them differ on the basis of their input used. Variants are named as CNN based and CNN-WordNet based approach. For CNN approach, consider a sentence of length n which is a concatenation of the pair sentences to be classified. It comprises of the words a_1, a_2, and so on till a_n. Let $a_{i:i+j}$ refer to the window of words $a_i, a_{i+1}, ..., a_{i+j}$. A convolution operation involves a filter w which is applied to a window of h words to create another element. For example, an element g_i is created from a window of words $a_{i:i+h}$ by $g_i = f(wa_{i:i+h} + b)$. Here $b \in R$ is a bias term and f is a non-linear function either rectified linear unit (*ReLU*) or tan hyperbolic function. This filter is applied to every conceivable window of words in the sentence $a_{1:h}$, $a_{2:h+1}$, . . . , $a_{nh+1:n}$ to deliver a feature map = $[g_1, g_2, ..., g_{nh+1}]$, with $c \in R^{nh+1}$. For CNN-WordNet, filters are convolved over input$_2$ obtained from section 4.1.2.1 and then a feature map g is generated.

A maximum pooling operation is then applied over the feature map and the maximum value of *max{g}* is then selected as the feature corresponding to this filter. The idea behind this is to capture the most imperative component of every feature map. The model uses numerous channels (with shifting window sizes) to obtain multiple features. These features are accumulated in the last but one layer. The accumulated features are then passed to the last softmax layer which produces the output. The output is the probability distribution over the class labels. Parameters of convolutional layer comprise of an arrangement of filters that can be learned. Even though each filter is small in scope, it reaches out through the full depth of the information volume. In the forward pass, the filter slides over every channel in the input vector and computes the dot products between the sections of the filters and the input. Filter sliding over the input produces an activation map that represents the relationship between filter response and spatial position. Feature detection activates the filters which are learned by the model. After convolutional layer, pooling is done to ensure that over-fitting does not occur and to reduce the amount of

[1]https://translate.google.com/

computation and parameters in the network.
The primary layers of CNN insert words into low-dimensional vectors. The following layer performs convolutions over the inserted word vectors utilizing different filter sizes. For instance, sliding more than three, four or five words at once. Next, the result of the convolutional layer is max-pooled into a long feature vector, which includes dropout regularization. Classification is done by utilizing the final softmax Layer.

4.1.2.3 Approach based on Recursive Neural Network

Bidirectional Long Short-Term Memory Recurrent Neural Network (BRNN) have been used in the past for various applications that involves sequence learning and have shown to be very effective. Some application involves predicting sequential data like handwriting and speech.
Neural network solves the problem of sequential information awareness and the problem of vanishing gradients. The attraction that deep learning holds is that one does not require to identify features or build resources like a dictionary for morphemes and other linguistic units. Second approach based on LSTM and BRNN is described in this section. LSTM performs better as compared to simple recurrent neural networks, hidden markov models, and other alternative neural networks because of its ability to learn from past experience. In BRNN (Figure 4.10), the neurons of a standard RNN are split into two directions, a positive time direction (forward states), and a negative time direction (backward states). By utilizing two directions, input data from the past and the future of the present time frame can be utilized. For forward pass, forward states and backward states are passed first, then output neurons are passed. For backward pass, output neurons are passed first, then forward states and backward states are passed next. After forward and backward passes are done, the weights are updated.
Traditional neural networks face the problem of rigidity in the sense that the input and output is a fixed size vector. The number of computational steps in the neural network is also fixed. The recurrent neural network is superior to traditional neural network because they can handle sequences. Traditional neural networks are limited by the number of layers in the model which can only perform a fixed number of computations. The appeal of recurrent neural network lies in the fact that they can operate on sequences and hence

are more capable of building intelligent systems. At every state, RNN combines the input vector with the state vector using a step function to create a new state vector. The output vector obtained from the RNN is an amalgamation of the current input and the inputs that are fed into the RNN in the past, ensuring that the RNN remembers the context. The internal state of the RNN is updated every time step function is called.
Each pair of sentences is mapped into a real vector domain. Words are encoded as real-valued vectors in a high dimensional space, where the similarity between words in terms of meaning translates to closeness in the vector space. LSTM is used as the next layer

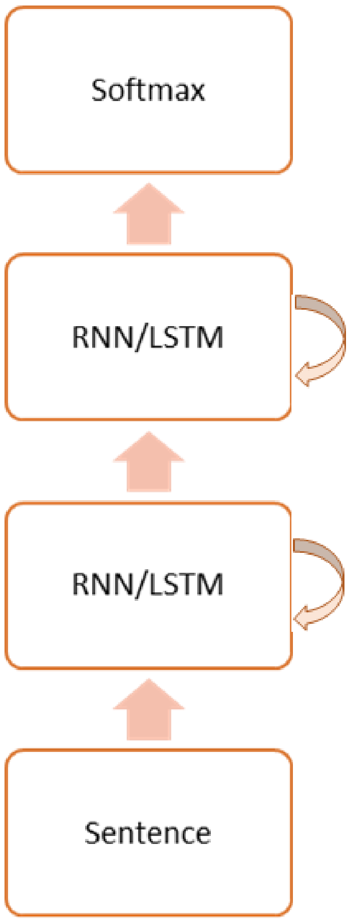


Figure 4.10: RNN Model

with *100* memory units (smart neurons). A dense output layer with a single neuron and a sigmoid activation function are used to make *0* or *1* predictions for the two classes (paraphrase or not paraphrase). As it is a binary classification problem, logrithmic loss is used as the loss function. An efficient *adam* optimization algorithm is used in the proposed approach.

4.1.2.4 Algorithm for Detecting Paraphrase in Indian Languages

Algorithm 4.2 describes the CNN and CNN-WordNet approach for paraphrase identification. CNN approach builds the embedding matrix using word2vec [Mikolov *et al.* 2013b]. CNN-WordNet builds a matrix using WordNet to find similarity between any two words in pair of sentences to be classified as paraphrases. An example matrix can be seen in Figure 4.9. For English, NLTK WordNet library is used. For the Indian languages, translation to English is performed because WordNet is not avaialble for these languages. The CNN is initialized with the following hyperparameters.

1. The sequence length is the maximum length of the sentence in the dataset.

2. The number of classes is two. The embedding size is set to *300*.

3. filter_size refers to the number of words required in convolutional filters. The value of the filter size lies in the range [*3, 5*] which means that the filters will slide over three, four and five words.

In CNN approach, the function $CreateEmbedding()$ defines the embedding layer which maps vocabulary word indices to vector representations. This can be equivalent to creating a look-up table from data. Embedding matrix, W is created during training. In CNN-WordNet approach, matrix W is used instead of the WordNet score between two sentences. Figure 4.9 shows a sample matrix for W. Since the filters are of different sizes and each convolution produces vectors of different shapes, a layer is created for each of them and results are merged into one feature vector. Once the output vectors are pooled from each filter size, they are combined to give the final vector. The vector from max-pooling is used to generate predictions by performing matrix multiplication and then choosing the class with the maximum score.

Algorithm 4.3 describes an approach based on LSTM. Four variants of LSTMs have been explored as discussed in section 4.1.2.3. First, an embedding matrix is created for the pair of sentences using Word2Vec [Mikolov *et al.* 2013b]. Following functions are used to update the parameters:

$$i_{\mathrm{t}} = (W_{\mathrm{i}}[h_{\mathrm{t\text{-}1}}, x_{\mathrm{t}}] + b_{\mathrm{i}}) \tag{4.1}$$

$$f_{\mathrm{t}} = (W_{\mathrm{f}}[h_{\mathrm{t\text{-}1}}, x_{\mathrm{t}}] + b_{\mathrm{f}}) \tag{4.2}$$

$$q_{\mathrm{t}} = tanh(W_{\mathrm{q}}[h_{\mathrm{t\text{-}1}}, x_{\mathrm{t}}] + b_{\mathrm{q}}) \tag{4.3}$$

$$o_{\mathrm{t}} = (W_{\mathrm{o}}[h_{\mathrm{t\text{-}1}}, x_{\mathrm{t}}] + b_{\mathrm{o}}) \tag{4.4}$$

$$c_{\mathrm{t}} = f_{\mathrm{t}} \odot c_{\mathrm{t\text{-}1}} + i_{\mathrm{t}} \odot q_{\mathrm{t}} \tag{4.5}$$

$$h_{\mathrm{t}} = o_{\mathrm{t}} \odot tanh(c_{\mathrm{t}}) \tag{4.6}$$

The function f_t in equation 4.2 is a control mechanism that is responsible as to how much old information is to be remembered in the current cell and it takes care of how much new information is to be stored. The function o_t in equation 4.4 controls the output given the cell c_t. The model has a learning rate of *0.001*, an embedding size of *300*. The final output vector is calculated by applying a softmax classifier over the hidden state cell h_t (equation 4.6) and initial word embeddings matrix.

4.1.2.5 Dataset

1. Microsoft Research Paraphrase Corpus(MSRPC)[Dolan *et al.* 2004]
   The MSRPC consists of *5801* pairs of sentences which have been extracted from news sources. The dataset is split into *4076* training examples and *1725* test examples. The dataset contains *67%* paraphrase pairs and *33%* non-paraphrase pairs.

2. Detecting Paraphrases in Indian Languages (DPIL) @ FIRE [Anand Kumar *et al.* 2016]
   DPIL dataset for identification of paraphrase or non-paraphrase (Dataset$_1$ as mentioned in section 4.1.1.2) is used for evaluation. The DPIL dataset consists of four Indian languages namely Hindi, Tamil, Malayalam and Punjabi. In Hindi, Malayalam and Tamil dataset, there are *2500* training and *900* testing sentence pairs. For Punjabi dataset, there are *1700* training and *500* testing sentence pairs, where training dataset consists of *700* instances of paraphrases and *1000* instances of non-paraphrases. Hindi, Malayalam and Tamil dataset contains approximately *40%* paraphrase pairs and *60%* non-paraphrase pairs.

Algorithm 4.2: Paraphrase Detection using CNN

```
input : Concatenated pair of sentences, I. Corresponding training class
        labels, X.
output: An output sequence P, the predicted class labels.
1  Initialization: the CNN with sequence_length=25, num_classes = 2,
    embedding_size = 128, filter_sizes = [3,4,5], num_filters = 3,
    pooled_outputs=[];
2  Preprocessing;
3  if CNN then
4      CreateEmbedding(I, X);
5  if CNN-WordNet then
6      for sentence pair in input do
7          W[len(Sentence1)][len(Sentence2)];
8          for word1 in Sentence1 do
9              List1 = WordNet.synsets(word1);
10             for word2 in Sentence2 do
11                 List2 = WordNet.synsets(word2);
12                 if List is empty or List2 is empty then
13                     if word1 == word2 then
14                         W[word1][word2] = 1;
15                     else
16
17                         W[word1][word2] = 0;
18                 else
19                     W[word1][word2] = Similarity(word1,word2);
20 for i in filter_sizes.length do
21     Initialize filter matrix F, bias matrix b;
22     conv = CreateConvolutionalLayer(F,b);
23     h = Non-linearity(conv,b);
24     pooled = Max-pool(h);
25 Combine pooled features(pooled);
26 Prediction = max(I*Feature_Vector);
```

Algorithm 4.3: Paraphrase Detection using RNN

```
input : A pair of sentences concatenated together. Corresponding
        training class labels Y.
output: An output sequence P, the predicted class labels(0/1).
1  Initialize hidden_dimensions = 128, word_dimensions = Number of
   words in vocabulary;
2  CreateEmbedding(W);
3  if 1 Layer LSTM then
4      update parameters;
5  if 2 Layer LSTM then
6      update parameters;
7      send parameters as input to Layer 2;
8  if 1 Layer Bi-LSTM then
9      forward pass for forward state LSTM;
10     forward pass for backward state LSTM;
11     update parameters ;
12 if 2 Layer Bi-LSTM then
13     forward pass for forward state LSTM;
14     forward pass for backward state LSTM;
15     update parameters;
16     send parameters as input to Layer 2;
17     forward pass for forward state LSTM;
18     forward pass for backward state LSTM;
19     update parameters;
20 output_vector = softmax();
21 Prediction = max(output_vector);
```

4.1.2.6 Experiments and Analysis

Figure 4.11 describes the results of RNN. Bidirectional LSTM (Bi-LSTM) with two layers performs the best for all languages. Bi-LSTM is more context-aware than LSTM, resulting in it's better performance. Performance of RNN is better for English than Hindi because the size of English dataset is approximately twice the size of Hindi dataset. The results for Hindi and Punjabi are better than Malayalam and Tamil. This may be due to the complex morphology of Tamil and Malayalam languages. Figure 4.12 describes the results of CNN. CNN performs better than CNN-WordNet (Figure 4.13), because of the use of word2vec. For Indian languages, no such tool as word2vec is present. As a consequence, the results are lower for all Indian languages when compared to English. The results obtained for CNN based approach are approximately equivalent to RNN based approach. CNN-WordNet performs equally well as CNN approach and slightly better for English. As can be seen from Figure 4.14, CNN outperforms RNN and CNN-WordNet. Results for English language are higher than other languages which can be attributed to the availability of word2vec and better WordNet scores. Results of Hindi-Punjabi and Malayalam-Tamil are comparable due to the similarity in underlying semantics for the languages pairs of Hindi-Punjabi and Tamil-Malayalam. RNN model worked well at *14* epochs because it quickly overfits the data, due to the small amount of data being used for training.

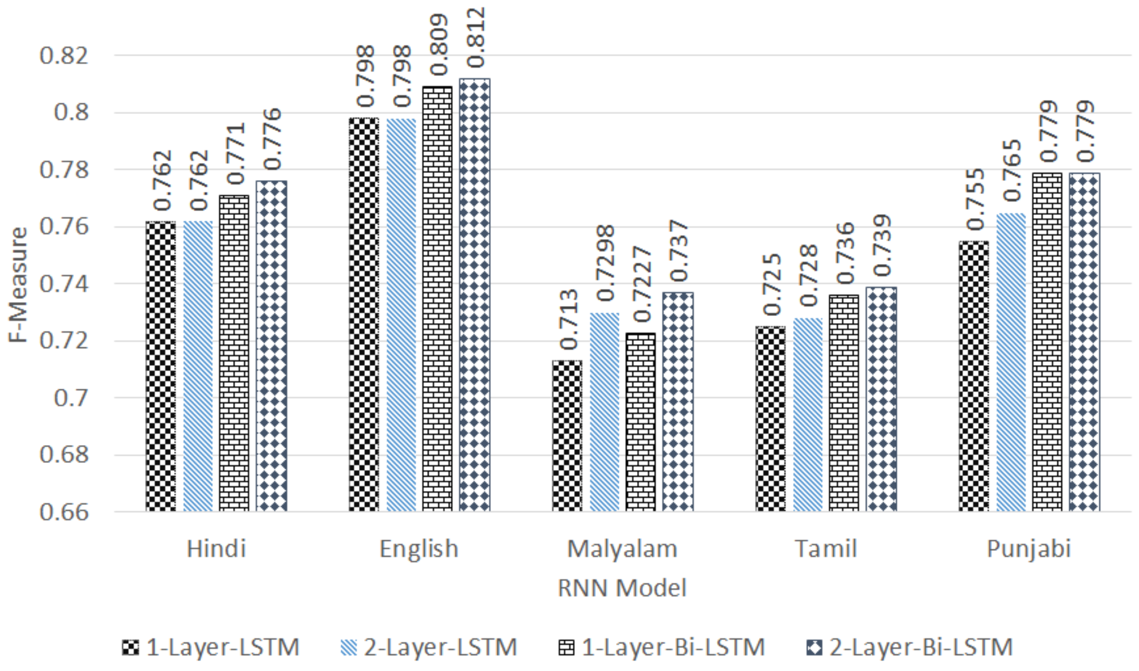


Figure 4.11: F-measure scores for RNN

4.1.2.6.1 Error Analysis

Due to the incorrect word similarity measures from WordNet scores, errors might have occured. Similarly, the stemmer used for Indian languages might have incorrectly returned the stem words, which could be a reason for wrongly classified paraphrases. Due to the mistranslation of few words while creating WordNet scores for Indian languages errors might have occured. This observation is consistent with the results as can be seen from Figure 4.14. The scores from CNN-WordNet are less than CNN for all the Indian languages.

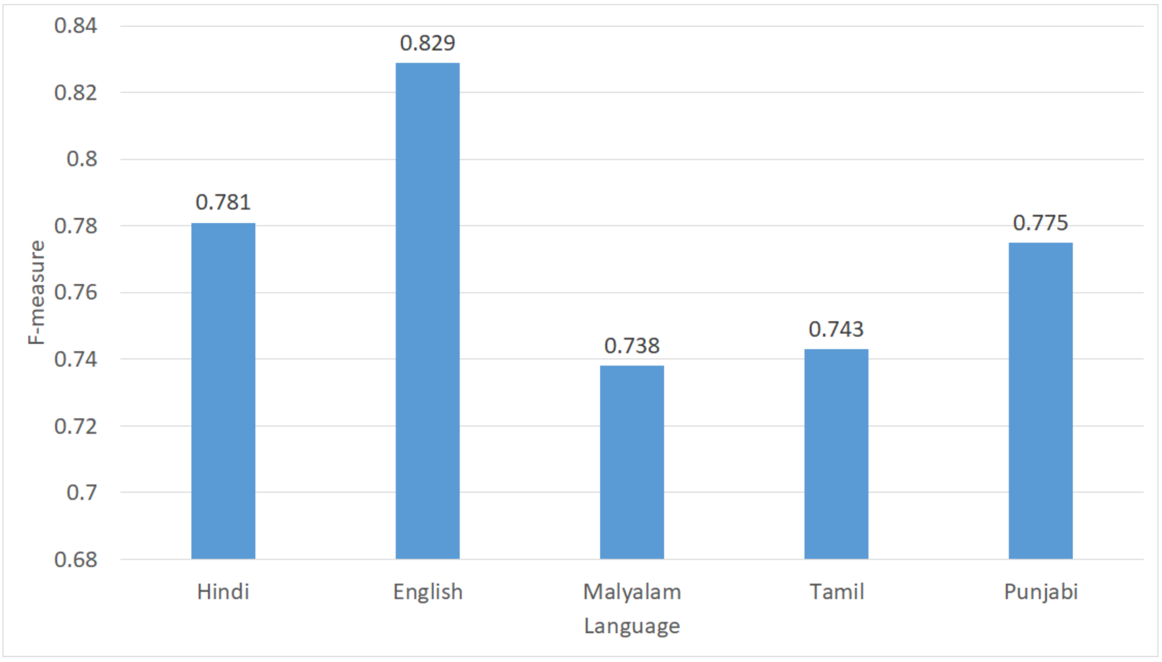


Figure 4.12: F-measure scores for CNN

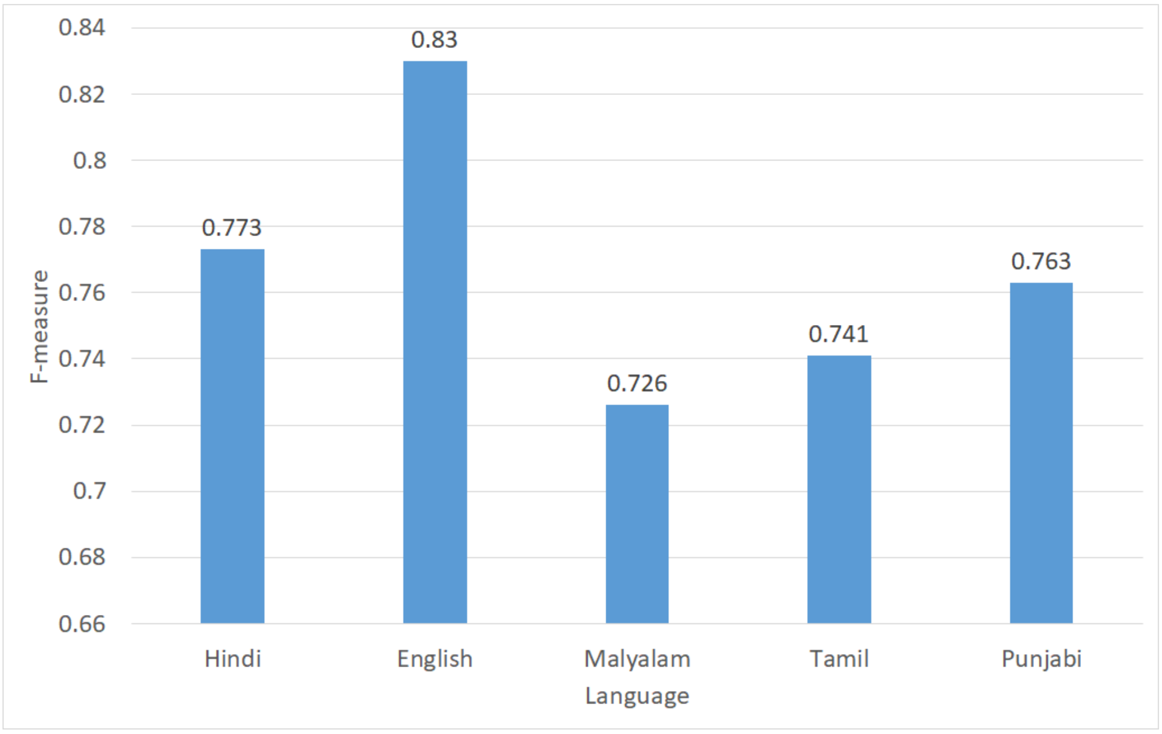


Figure 4.13: F-measure scores for CNN-WordNet

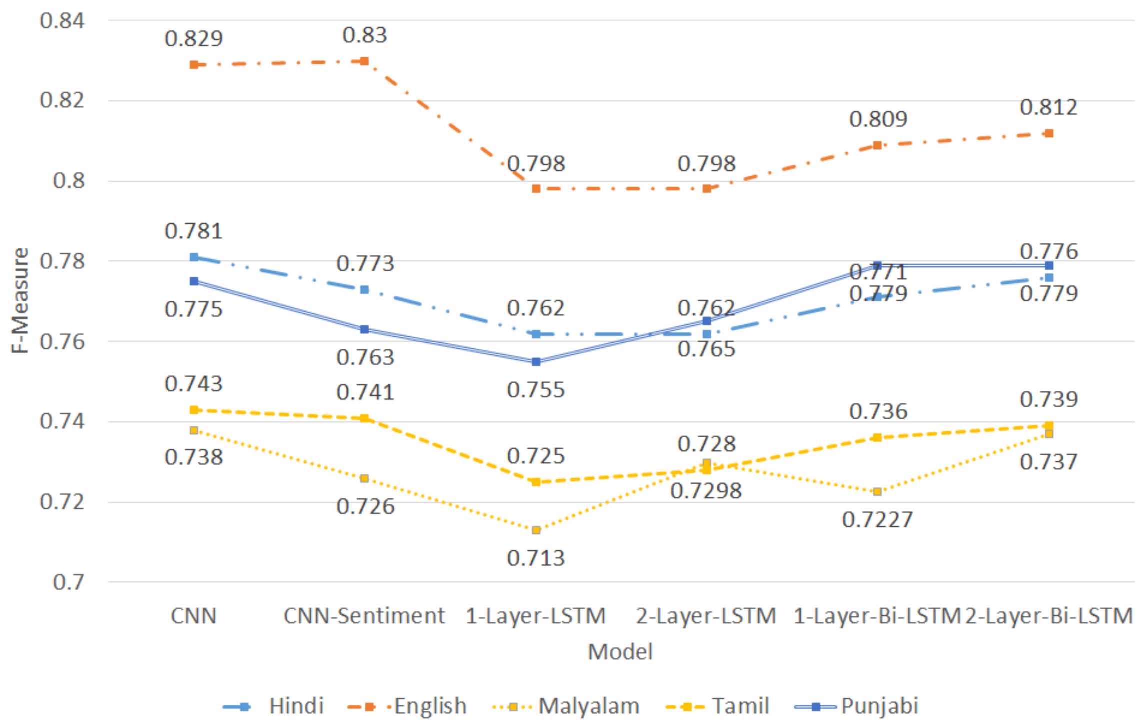


Figure 4.14: F-measure scores for all approaches

4.2 Extractive Text Summarization

The goal of automatic summarization is to take an information source, extract important information from it and present it to the user in a short form and in a manner useful to the user's need. The continuing growth of online text resources has resulted in the well-recognized problem of information overload. As a result, it is especially useful to have tools which can help users digest information content by presenting the most relevant information. Extractive summaries are particularly useful in extracting and representing important snippets of information. The challenge is to generate useful summaries automatically. The approach, application and the end-objective of summarization of documents determine the type of summary generated. A generic summary gives a high-level information of the document's content while a query based summary returns the text that is closely related to the input query. Similarly, it can also be categorized based on application, to develop a single or multi-document summarization. The widely used technique of extractive summarization involves selection of a subset of the document sentences which are representative of its content and creating the summary by concatenating selected excerpts from the original document. This section proposes an approach which aims to achieve the goal of text summarization by generating extractive summaries using deep learning techniques. This includes processing text and generating the list of sentences which might be the most useful and contain the major gist of the text. Humans generally do not perceive summaries as sentences extracted verbatim from the text but rather try to summarize it in a format that conveys the same meaning as the given text. However, extractive summaries do contain an important piece of text of the whole content. This provides an idea of what the text is about and at the same time certain sentences which can be used to quote or refer to for some other purpose. Another motivation that drives this work is to try out the approach of abstract generation using data-driven approaches for Indian languages. The idea of the approach is to generate labels using paraphrasing algorithm and then use them for extractive summarization.

Approaches involving deep-learning have shifted the focus from manually engineering the features to a more data-driven approach wherein a neural network with sufficient depth can be used to extract features and use them to classify sentences as being important to be put into the summary or not. The sentences are converted to vectors in a suitable space and are fed to the network as numbers. It has been observed, given sufficient data and a suitable network architecture, ANN learns to represent words in the space in such a manner that it captures various linguistic properties and maps it to simple vector operations. This representation of words is called embedding and the vectors of the words are called word vectors [Mikolov *et al.* 2013b]. The proposed approach uses fully connected CNN.

Sentences can be viewed as sequences of the word and possess some memory of previous words seen, this concept has proven to be useful in classification. Regardless of all approaches mentioned above to achieve the task, one of the major challenges that are faced in this domain of automatic text summarization is the problem of result evaluation. Most of the works rely on some form of human intervention to evaluate the results as datasets with gold standard extractive summaries are a rarity and adding to this even automated evaluation such as ROUGE depend on human summaries to be provided to them. Hence, for training purposes, proposed approach uses an approach of generating the labels for each sentence in whether it should belong to summary or not by using the technique of paraphrasing on the original text and abstractive gold summaries available for the dataset.

4.2.1 Workflow for Proposed approach

The proposed approach can be divided into two phases. The first phase generates a true label for each sentence to be in summary or not whereas the second phase focuses on model creation for summarizer. The model generation phase can be further divided into two stages, the first of them being a convolutional layer to act as a feature map generator and the second being densely connected layers of neurons or otherwise called MLP. It is a binary classification problem and hence model generates two scores for a given sentence, one for each class.

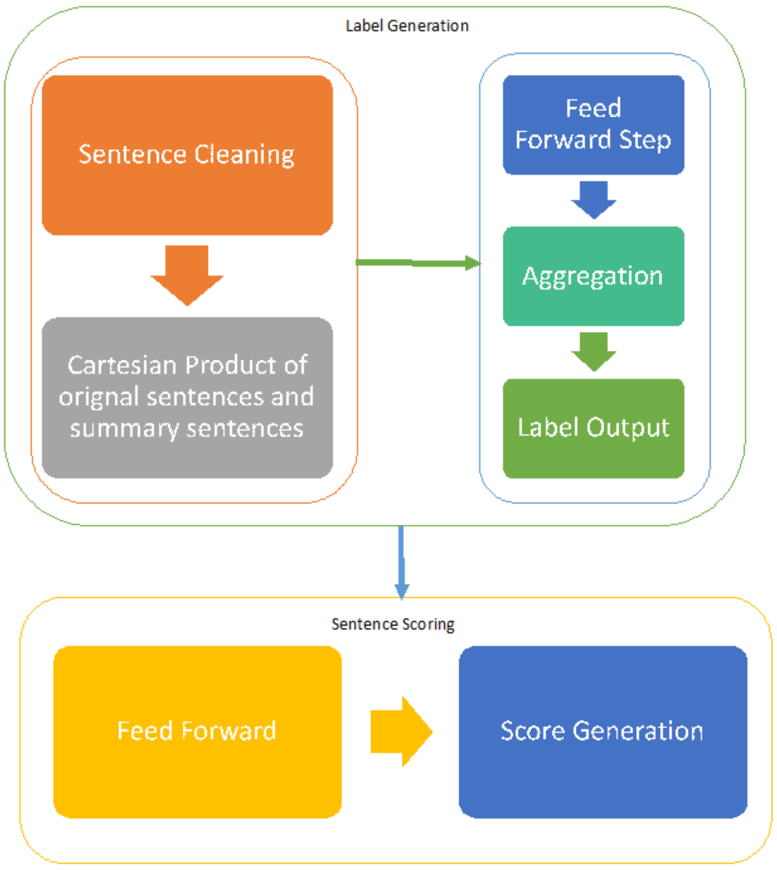


Figure 4.15: Block Diagram showing basic workflow

4.2.2 True Label Generation

Extractive text summarization like any other machine learning task requires that the data be of appropriate nature. The summaries of the training data presented to the system are required to be in extractive form for current problem. Most of the available public datasets

are abstractive in nature and the problem is more so prevalent in the field of Indian languages which does not have any gold-standard dataset for summarization. For proposed approach, gold-summaries are generated from the human-generated summaries using the help of paraphrase detection approach (proposed in section 4.1.2)
For the proposed solution, ground truth for a document D is generated by making $|\mathbf{D}|\ X\ |\mathbf{S}|$ pairs (where $|\mathbf{D}|$ stands for number of sentences in document D and $|\mathbf{S}|$ stands for number of sentences in S, abstractive gold summary) and running a paraphrase detection system on all the pairs. The label for a sentence d is given as:

$$label(d) = \begin{cases} 1 & \text{if } d \in \mathbf{P}, \\ 0 & \text{otherwise} \end{cases}$$

Here, $\mathbf{P}$ denotes the set of tuples which have been detected as paraphrases. The model used for the summary generation is discussed in detail below.

4.2.3 Model Generation

Convolutional Layer

Document D is fed to the network on a sentence-to-sentence basis, where sentence d is represented by a list of words. The list is then passed through the embedding layer which results in the list being converted to a word matrix $\mathbf{W} \in \mathbb{R}^{f \times d}$, where f stands for the maximum length of sentence in the corpus and d stands for the dimension of embedding vector. Each row of the matrix W_i is trainable vector representation of the word in the sentence. The matrix $\mathbf{W}$ is then fed to the convolutional layer which performs a convolution operation between $\mathbf{W}$ and a kernel $\mathbf{K} \in \mathbb{R}^{h \times d}$ of width h as follows:

$$f_j = ReLU(W_{j:\,j+h} \odot K + b)$$

where b is the shared-bias and f_j is the j^{th} element in feature map f. Following this, a max-pooling operation is done to select one feature which would represent the sentence under the kernel.

$$d_K = \max_j f_j$$

Multiple such kernels (K) are used with different widths to produce multiple feature maps.

Dense Layer

The outputs from each of the max-pooled layers above for different kernels K are then flattened and concatenated to produce a vector I which is then fed through a fully connected layer and finally a softmax layer designed for two-class classification is applied.

4.2.4 Algorithm: Extractive Text Summarization using Deep Learning

Algorithm 4.4: ETSDL: Extractive Text Summarization using Deep Learning

```
1  Function SummarizeTrain(textfile,summaryfile)is
2      Call LabelGen($textfile, summaryfile$); // It produces a file
           with all labels
3      ($text\_batches, label\_batches$) ← Call BG($textfile, labelfile$);
4      for ($text\_batch, label\_batch$) ← $zip(text\_batches, label\_batches)$ do
5          Call Train($text\_batch, label\_batch$);
6  Function Train(text_batch, label_batch)is
7      ($embedding\_layer \leftarrow embeddinglookup(text\_batch)$);
8      for ($i \leftarrow filter\_map$) do
9          $conv \leftarrow Conv2D(text\_batch)$;
10         $mp2d \leftarrow MaxPool2D(conv)$;
11         $flatten \leftarrow flatten(mp2d)$;
12         $flattens \leftarrow flattens.append(flatten(mp2d))$;
13     $pool \leftarrow concatenate(flattens)$;
14     $dropout \leftarrow dropout(pool)$;
15     $dense \leftarrow Dense(dropout)$;
16     $softmax \leftarrow softmax(dense)$;
17     $optimizer(softmax, test\_labels)$;
```

The *SummarizeTrain* function shown in algorithm 4.4 creates batches of the dataset and passes them to the training module. The model presented here is trained via batch gradient descent. The *Train* function performs both, a forward pass through the network to compute the output given by current network on given data and also a back propagation and weight update step to change the weights. It passes the given batch of data through the network shown in Figure 4.16. The step shown in *SummarizeTrain* is for one iteration through the dataset (one epoch). The same operations are performed for multiple

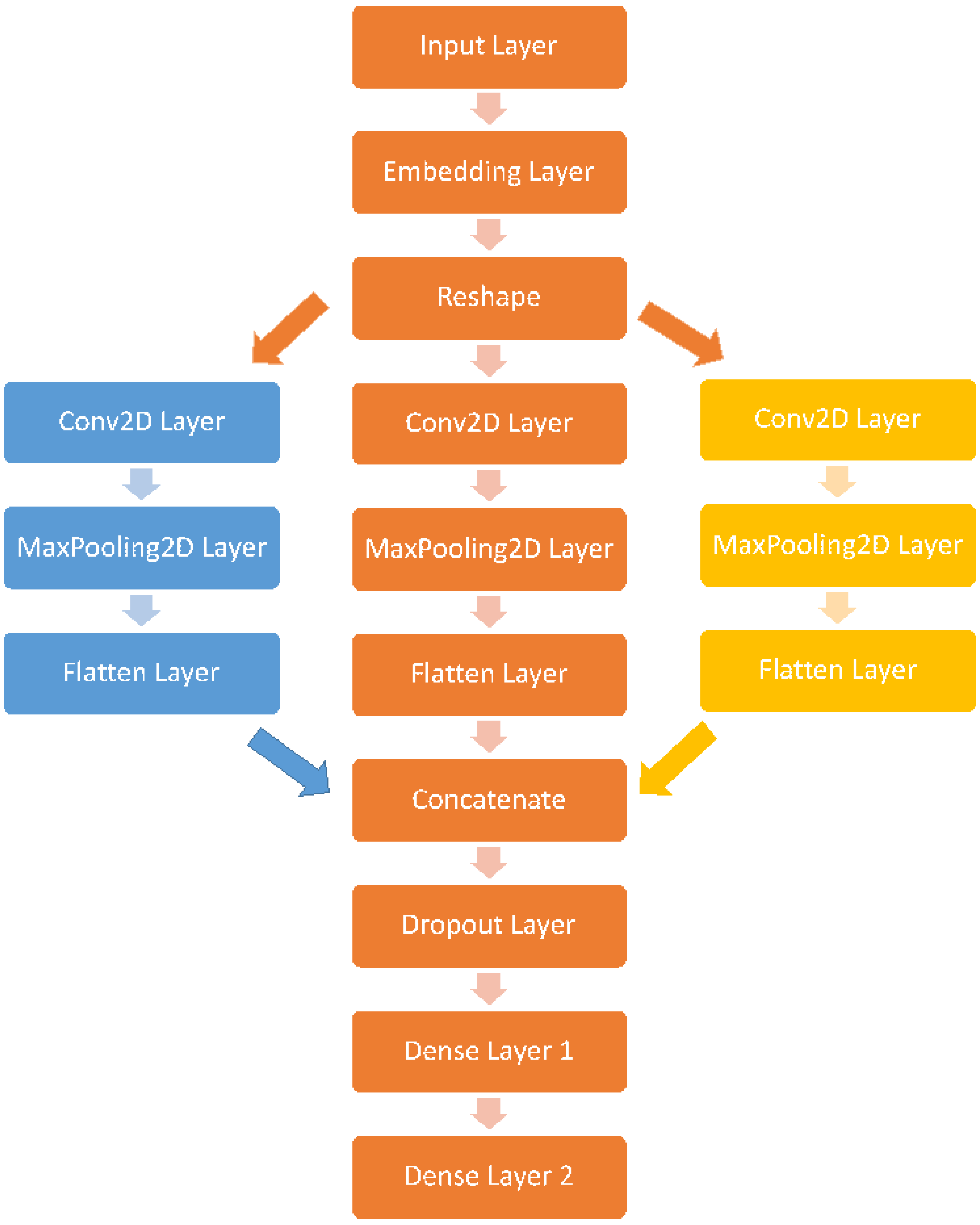


Figure 4.16: Network Architecture as a block diagram

epochs. The *Train* module shows the network feed through in a highly simplified manner and in the form of functional code. The actual parameters and settings are described in section 4.2.5.2.

4.2.5 Experiments & Results

The proposed model has many hyperparameters which are tuned for the specific dataset. Para Multiling *2015* dataset is used for evaluation. This dataset is designed for the pilot task of Multilingual Single-document Summarization (MSS). The dataset consists of *40* different languages and for each language, *30* documents are given. The documents are in UTF-8 without markups and images. For each document of the training set, the human-generated summary is provided along with character length. Indian Language being the prime target, results have been evaluated in three languages (English, Malayalam, Hindi) only.

4.2.5.1 Parameters

The given model has hyper-parameters in both the stages, label generation and summary generation. The model that has been saved is at the point when the loss factor is the least. The model is checkpointed via a call-back function which after every epoch decides to the checkpoint if the loss function has improved from the previous best-saved model instance. For the label generation, hyperparameters used are mentioned in Table 4.1. The above mentioned parameters are arrived upon to avoid overfitting because of small

Table 4.1: Parameter settings for Label Generation

Parameters	Value
Number of neurons in Dense Layer	64
Embedding size	128
Kernel sizes	[3,4]
Number of maps produced	64
Learning Rate	0.003
Regularization Rate	0.003 (L2 regularization)

dataset. The parameters when set to values higher than current values (number of filters to *5* and number of maps produced to *128*), produced drastic over-fitting. The same is also expected since increasing value of parameters results in increase of number of trainable parameters, which in turn would require a higher amount of data to prevent the network from over-fitting. The values could be set to values lower as well but that would result in the decrease in learning capacity of the network. In proposed summarization model, the

Table 4.2: Parameter settings for Summary Generation

Parameters	Value
Number of neurons in Dense Layer	64
Embedding size	128
Kernel sizes	[3,4,5]
Number of maps produced	64
Learning Rate	0.003
Regularization (L2) Rate	0.003

learning rate has arrived at this value by using a decaying rate wherein a callback function is used to decrease the learning rate after every few epochs. Parameters related to building model are mentioned in Table 4.2. One more factor that has to be considered is the factor of drop-out, which is kept to be *0.5*. This means that on an average activity of *50%* of the neurons are not passed during the feed forward stage preventing the network from rote learning the activities and hence reducing the effect of over-fitting to some extent. The reason for such a high drop-out rate is the fact that the model without the drop-out is highly volatile and hence is over-fitting heavily even in such a shallow network. A general value for drop-out would range anywhere between *0.2* to *0.5*. Since the size of the matrix that is fed to a convolutional network has to be fixed, sentences are padded to a length that is equal to the maximum of all lengths of the training samples.

4.2.5.2 Evaluation & Discussion

This section presents the results of the proposed model and discusses the possible reasons why certain results are as they have been reported. Results for English dataset are shown in Table 4.3, whereas results for Malayalam are shown in Table 4.4. It can be observed from the tables that although accuracy is greater than *90%*, other evaluation measures are low. This may be due to the fact that the generated dataset is based on the human-based summaries. However, for Malayalam dataset a recall of *0.8459* is obtained which shows the trend that the original gold summaries contained most of the words from the original text itself. In proposed approach, summary generation is converted to a classification task of identifying if the sentence belongs to summary or not and hence, accuracy, precision, F-measure and recall metric is considered instead of ROUGE.

Table 4.3: Preliminary results for Summary Generation in English

Parameters	Value
Training Accuracy	0.9615
Validation Accuracy	0.7063
Testing Accuracy	0.757
Testing Precision	0.25
Testing Recall	0.4287
Testing F-Score	0.3157

Table 4.4: Preliminary results for Summary Generation in Malayalam

Parameters	Value
Training Accuracy	0.9114
Validation Accuracy	0.7346
Testing Accuracy	0.5933
Testing Precision	0.3919
Testing Recall	0.8459
Testing F-Score	0.5357

4.3 Abstractive Text Summarization

There is a large amount of data on the web which expresses the same opinion over and over again and thus summarization of redundant content has become a necessity. While viewing multi-document summaries or the summaries of highly redundant text, extractive summarization would not be of any help as the extractive summaries would be very verbose and biased. Sentences also tend to be longer, hence non-essential parts of the sentence also get included. Relevant information is spread across the document and this can't be captured in the extractive summaries. Extractive summaries also face the problem of 'dangling' anaphora, implying that sentences that contain pronouns lose meaning when extracted out of context, the resolution of this problem is presented in [Steinberger *et al.* 2007].

While there has been a lot of work done in the field of extraction based summarization, abstraction based summarization is difficult because of the simple reason that while the computers can statistically select the most important sentence from the text, it is difficult

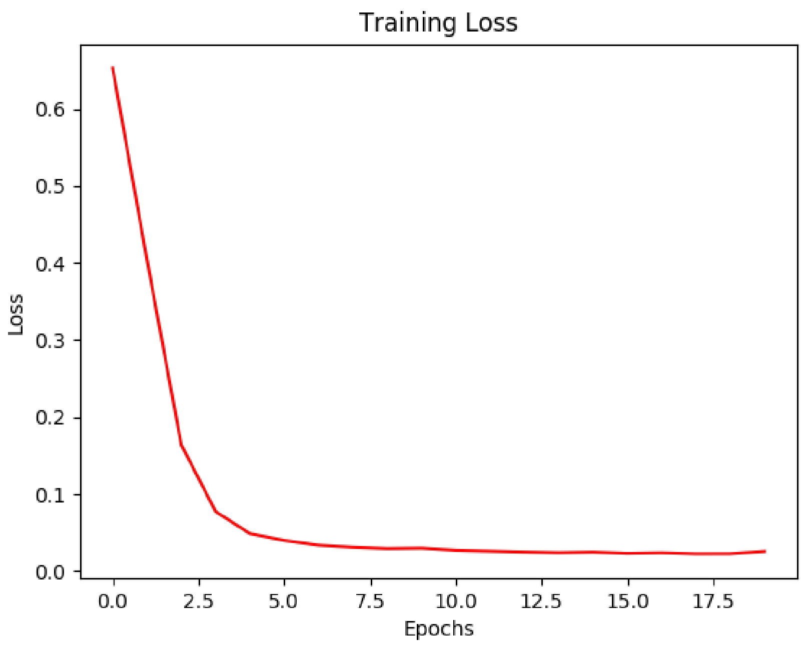


Figure 4.17: Training loss vs number of epochs

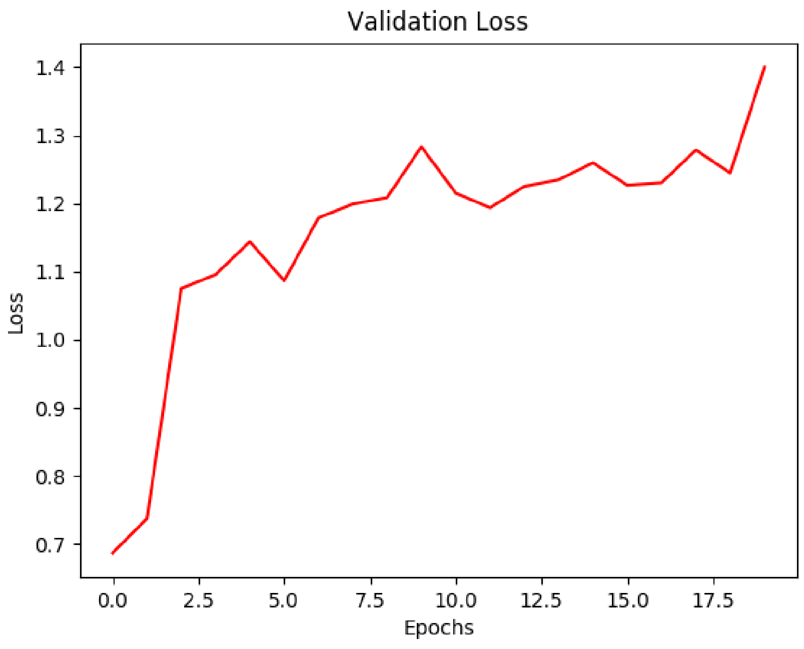


Figure 4.18: Validation loss vs number of epochs

for them to combine important sentences and generate a coherent and concise synopsis. Demand for the high-quality summary is on the rise whether it is regarding summarization of textual content (for example books etc.) or multimedia content like video transcripts etc. [Ding *et al.* 2012].

4.3.1 Abstractive Text Summarization using Sentiment Infusion

It has been demonstrated that abstractive summaries are better than extractive summaries [Carenini & Cheung 2008] whenever documents with a lot of redundant content are considered for summarization (for example, product reviews, blogs and news articles). This is because abstractive summaries are compact and present the useful information and are not verbose. However, generating abstract summary is a tougher task than the generation of extract summary. Single document summarization differs from multi-document summarization since single documents contain lesser data. Hence, more efficient strategies are required to generate abstractive summaries in case of single documents. An approach named Abstractive Text Summarization using Sentiment Infusion (ATSSI) is proposed for compressing and merging information based on word graphs and then summaries are generated from the resulting sentences. The approach assumes no domain knowledge and leverages redundancy in the text. The results show that the summaries generated are agreeable to human compendium and are concise and well formed.

4.3.1.1 Building the word graph

Graph data structure is used in ATSSI to represent the text. Graphs have been frequently used for abstractive text summarization ([Kumar *et al.* 2013], [Liu *et al.* 2015], etc.) and have shown promising results. In state of art, graphs are used in different form for the text summarization. [Kumar *et al.* 2013] use graph to represent the bi-gram relationship between the words in the text. In approach proposed by [Liu *et al.* 2015], semantic information is embedded in the graphs. Proposed approach in this work uses graph differently from above mentioned forms, as each node represents a word in the text along with the information of the position of the given word in the sentence and the edges represent the adjacency of the words in the sentence. A document is represented as a directed graph where V $=v_i, v_{i+1}...v_n$ is a set of vertices that represent words in the text. Each vertex node stores the information about the POS tag of the word in that node, the position of the word in the sentence and the position of the sentence in the document. The graph naturally captures the redundancy in the document since words that occur more than once in the text are mapped to the same vertex. Furthermore, the graph construction does not require any domain knowledge. The graph also captures the minor variations in the sentences. For example, Figure 4.19.

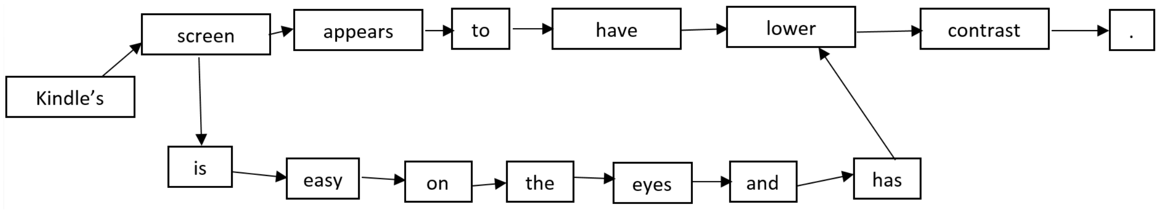


Figure 4.19: Graph Capturing Redundancy in the text

4.3.1.2 Ensuring the sentence correctness

The correctness of sentence is ensured using the following set of Part of Speech constraints:

- A sentence can contain noun followed by a verb and an adjective or an adjective followed by a noun and a verb or a verb followed by an adjective and a noun or an adverb followed by an adjective and a noun or an adverb followed by a noun.
- The start of the sentence should contain a word whose average position in all sentences is lower than the threshold, called Start Node. This threshold is enforced to corroborate that the sentences occurring in the summary do not start with words that occur somewhere in the middle of a sentence.
- The sentence should not end in a conjunction like but, yet, etc.

4.3.1.3 Getting abstractive summary

1. Scoring of paths:

   The paths are then scored based on the redundancy of the overlapping sentences. This redundancy can be calculated using the intersection of the position of the words in the sentences *(P)* such that the difference between the positions is no greater than a threshold, P. This redundancy helps us in deciding the number of sentences discussing something similar at each point in the path.

The scores can simply be based on the calculation of the overlap or can include the length of the path as well because if the path is longer, higher redundancy is expected than in a shorter path since longer paths provide more coverage.

2. Fusing sentiments:

   A node is considered to fuse sentences if its POS tag is a verb. If a vertex V is being considered as a node that can be used to fuse sentences, then previous vertices are traversed in the path currently being considered to look for a connector as shown in Figure 4.20. An alternative approach is to calculate the sentiment of both the sentences to be fused and to look for a connector that can be accurately used. This sentiment is calculated using SentiWordNet 3.0 [Baccianella *et al.* 2010b].

   Once the sentiment has been calculated, the connector is chosen from a pre-existing

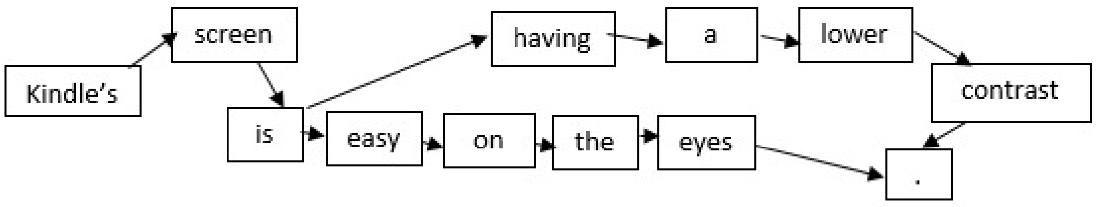


Figure 4.20: Example Sentences that can be fused together

   list. For example, if the sentiments of the two sentences are contradictory, 'but' is used as a conjunction. If both sentences are positive, then depending upon the context conjuctions such as 'and' and 'or' may be used.

3. Summarization

   After all the paths are scored and the sentences have been fused, sentences are ranked in descending order of their scores. Duplicate sentences are removed from generated summary using Jaccard similarity coefficient. The remaining top S (number of sentences specified by the user to be in summary) sentences are chosen for the summary.

4.3.1.4 Pseudocode for ATSSI

1. Generate the graph from the text input such that nodes will contain the information about
   (a) The position of the word in the sentence
   (b) The position of the sentence in the document
   (c) The POS tag of the word.
2. For all nodes in the graph, if node satisfies the constraint of being lesser than Start Node, the graph is traversed.
3. While traversing the graph, if the path overlap is greater than P, check if the current node is a valid end node and the current sentence is a valid sentence, if it is, add it to the list of candidate summaries, else discard it.
4. For all the neighbours of the current node
   (a) Calculate the redundancy.
   (b) Check if the node can be used to fuse a sentence.
   (c) If yes, then calculate the sentiment of the anchor statements, and choose the connector accordingly from the pre-existing conjunction list. If the node cannot be used to fuse sentences, graph is traversed again.
5. Graph is again traversed from all the neighbours of the current node to find the further nodes of the sentence.
6. The new score is computed and the duplicate sentences are removed from the fused sentences. The resulting fused sentence and its final score are then added to the original list of candidate summaries.
7. Once all paths have been explored, duplicates are removed. The rest of the sentences are sorted in descending order of their path scores. The best S candidates are chosen for the final summary.

4.3.1.5 Dataset Description

Two datasets are used for the evaluation:

- National Institute of Science and Technology (NIST) organizes a conference called Document Understanding Conference (DUC) every year. The first dataset comprises of *50* documents from the DUC *2002*[1] corpus which has been randomly selected. The documents contain about *500* words on an average. The dataset contains about *500* news articles in English along with gold summaries for each article. The gold summaries have also been provided for the corresponding documents and are about *100* words on an average.
- Second dataset [Ganesan *et al.* 2010a] contains *51* documents pertaining to a single query, for example, Amazon Kindle: buttons, Holiday Inn, Chicago: staff, etc. There are about *100* redundant, unordered sentences in the document for every query. There are *4* peer summaries corresponding to each of these *51* documents.

4.3.1.6 Experiments & Results

ROUGE metric is introduced by [Lin 2004] and has been adopted by the DUC and leading conferences on Natural Language Processing. ROUGE calculates the overlap between the candidate summaries and the reference summaries and it has been found that correlation of ROUGE-1 and ROUGE-2 is the most with human summaries ([Lin & Hovy 2003]). ROUGE-N is a recall measure that computes the number of matches between the candidate summaries and the reference summaries. The formula to calculate the ROUGE scores [Lin 2004] is given as:

$$ROUGE-N = \frac{\sum_{S\epsilon(ReferenceSummaries)} \sum_{gram_n \epsilon S} Count_{match}(gram_n)}{\sum_{S\epsilon(ReferenceSummaries)} \sum_{gram_n \epsilon S} Count(gram_n)} \tag{4.7}$$

where *match* is the maximum number of N-grams that occur in the reference summaries and the candidate summary. Count is the number of *n*-grams in the reference summaries.

[1]http://duc.nist.gov/

The precision, recall and F-measure [Lin 2004] is calculated as follows:

$$Precision = \frac{Match(Sentence)}{Match\_Candidate(Sentence)} \tag{4.8}$$

$$Recall = \frac{Match(Sentence)}{Match\_Best\_Candidate(Sentence)} \tag{4.9}$$

$$F - measure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{4.10}$$

where $\mathrm{Match}_{\mathrm{Candidate}}$ is the number of sentences of present in the candidate summary. $\mathrm{Match}_{\mathrm{Best\_Candidate}}$ is the total number of sentences in the best sentences summary. In experiments, ROUGE-1 and ROUGE-2 are used for comparison with Baseline 1 and for comparison with Baseline 2 only ROUGE-1 is used.

4.3.1.6.1 Evaluation & Discussion Results are being compared by two baselines on two different datasets, apart from the comparison with human summaries. $\mathrm{Baseline}_1$ is defined by the algorithm implemented by [Ganesan *et al.* 2010a] whereas $\mathrm{Baseline}_2$ is defined by algorithm mentioned in [Lloret & Palomar 2011]. $\mathrm{Baseline}_1$ and $\mathrm{Baseline}_2$ have been chosen for comparison since they have used graph-based algorithms for summarization and proposed algorithm overcomes the limitations of [Ganesan *et al.* 2010a] and [Lloret & Palomar 2011]. [Ganesan *et al.* 2010a] describe an approach that used directed graphs that use the original sentence word order to generate abstractive summaries. Their technique leverages the graphical form of the input text to reduce redundancy. If their algorithm finds two sentences that are collapsible, they use the connectors already present in one of the sentences to be used as the connector for the collapsed sentence. However, this technique is effective, but it has a drawback that there might be two sentences which are capable of being fused together but can't be fused because of the absence of a pre-existing connector. The proposed approach (ATSSI) does not face this drawback since sentiment analysis is used. [Lloret & Palomar 2011] describes a technique where they have built a directed weighted word graph where each word text represents a node in the graph and the edge contains the adjacency relation between the words. The weight of the edge is determined by using a combination of PageRank value and the frequency of the words. To determine important sentences, first node consists of ten words with highest

TF-IDF score. Sentence correctness is ensured using the basic rules of grammar like the length a sentence should be greater than *3* words, a sentence must contain a verb and should not end in an article or conjunction. A flaw with this methodology is that a lot of important information is lost because of the impositions of grammar on the sentences and the policy of selecting the ten words with highest TF-IDF scores. Furthermore, a lot of redundant sentences will still be present in the summary because the TF-IDF scores will give more importance to them. Proposed approach (ATSSI) does not face the deficiency that is faced by [Lloret & Palomar 2011] because it incorporates the redundancies in graph structure itself.

Figure 4.21 shows that ATSSI has higher precision over the Baseline$_1$, this is because proposed approach has overcome the demerit of the approach as stated by [Ganesan *et al.* 2010a], that it can only connect the sentences if there is a pre-existing connector. Since the dataset used by [Ganesan *et al.* 2010a] already has redundant data with connectors, ATSSI is only showing a marginal difference in precision. Dataset [Ganesan *et al.* 2010a] results in low recall as shown in Figure 4.21 because there is a high presence of redundant information in the dataset which is leading to infusing large number of sentiments in a particular sentence. This is resulting in less number of sentences.

Proposed algorithm outperforms Baseline$_2$ [Ganesan *et al.* 2010a] by *13%* as shown in Figure 4.22, since Baseline$_2$ [Ganesan *et al.* 2010a] describes rigid rules for ensuring sentence correctness and has no provision for fusing sentences. ATSSI outperforms Baseline$_2$ [Ganesan *et al.* 2010a] by a considerable margin of precision since ATSSI incorporates sentiment infusion and a provision for removing redundancy. Recall of ATSSI is also marginally low as that of Baseline$_2$ for similar reasons mentioned for Baseline$_1$. Figure 4.21 represents the F-measure on the dataset by the proposed algorithm and compares it with human summary and Baseline$_1$ whereas Figure 4.22 represents the precision and recall on DUC *2002* dataset with the human-summary and Baseline$_2$ respectively.

Finally, it is worth noting that generating summaries that are purely abstractive in nature is an onerous task, as shown by [Liu & Liu 2009] where F-measure values are in the range *13%* to *18%*.

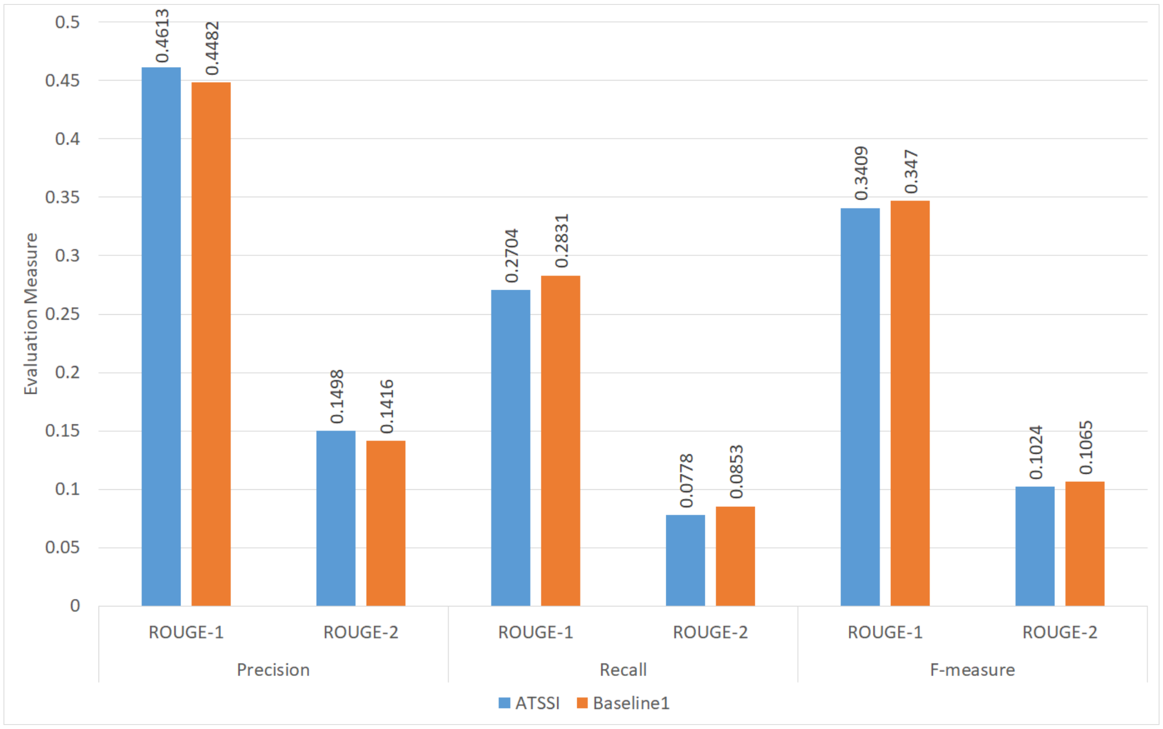


Figure 4.21: Evaluated results on dataset [Ganesan *et al.* 2010a] with human summary and Baseline 1 vs ATSSI

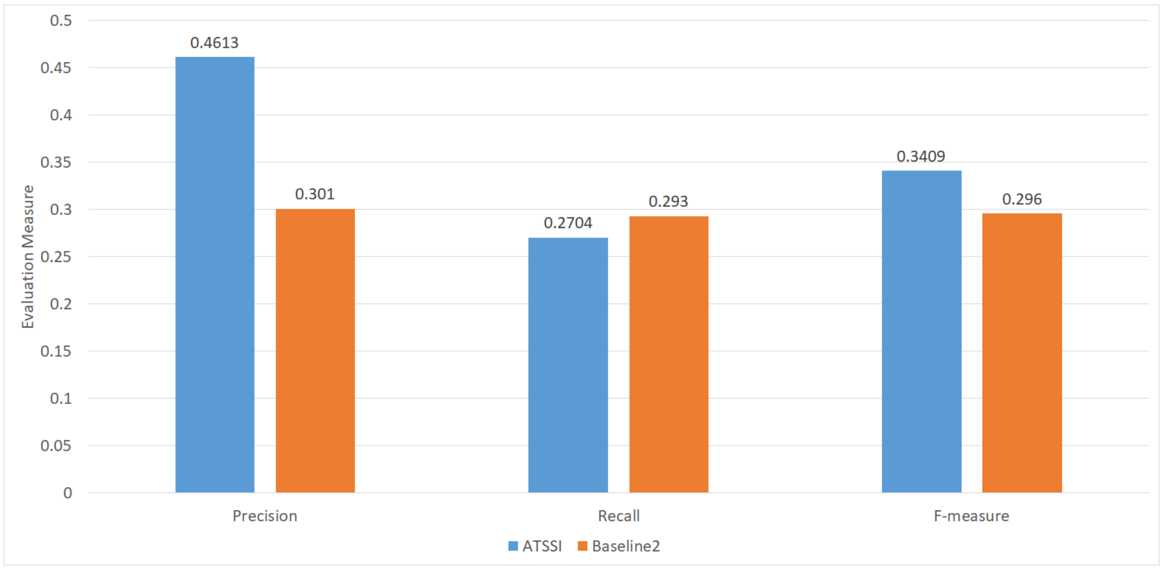


Figure 4.22: Evaluated results on DUC2002 dataset with human summary and Baseline 2 vs ATSSI

4.3.2 Abstractive Text Summarization using Generative Adversarial Networks

In this section, a deep learning approach has been proposed which uses a generative model to generate abstractive summaries from the input datasets. Generative Adversarial Networks (GAN) have been used for caption generation ([Reed *et al.* 2016]), generating text from images, face generation, etc.

A GAN can be decomposed into two adversaries, a discriminator, and a generator. The probability distribution learning problem is posed as a game in between the two, which is solved using the min-max algorithm. The generator performs the task of generating samples and the discriminator is a binary classifier with the task of separating the real samples from the fake ones. GAN(s) was first proposed by [Goodfellow *et al.* 2014] in *2014*. Deep learning holds the promise of building hierarchical models which can represent the probability distribution of data ranging from text to images. However, most of the success in the past has been with models that can map the high dimensional input to a class label. Discriminative models perform better than generative models because of the use of backpropagation and dropout algorithms.

Generative models are difficult to train because of the approximation involved in determining the probabilities and maximum likelihood estimations. The model proposed by [Goodfellow *et al.* 2014] circumvents these limitations. These models are advantageous to use because even noisy data can be represented by these models. In contrast, Markov chains based models require data that can be partitioned easily for the chains to generate a probability distribution. The other advantage for GANs is that the network is updated only by the gradients. Hence, the noise in the input is not propagated forward to the model. However, training GANs are very difficult as they are very unstable and very volatile with the slightest change in the hyper parameters[Arjovsky & Bottou 2017].

The alternative to GANs is Wasserstein GAN (WGAN) introduced by [Arjovsky & Bottou 2017]. There are various benefits of WGAN(s) over GAN(s). One of the advantages is that WGANs can be trained until optimality. WGANs specify the point till the generator is to be trained. WGAN(s) are also more resilient as compared to GANs when the hyperparameters for the network are varied. Despite being used extensively for image generation, use of GANs in natural language processing has not been so widespread. In this section, these models are adapted to tackle text summarization which involves generating a short summary for a longer piece of text.

4.3.2.1 Proposed Approach: Text Generation using Generative Adversarial Networks (TGGAN)

Paraphrase detection algorithm proposed in section 4.1 is used to reduce the redundancy in the text by removing duplicate paraphrases. The output generated without redundant content is then passed as an input to GANs. The building blocks for the GAN are GRU, and Recurrent Neural Networks are used for comparisons. The number of computations in GRU(s) are significantly lesser than Long Short Term Memory Networks. The steps inside a GRU can be broken down into:

- Gate for updating $z_t = \sigma(W_z x_t + U_z h_{t-1})$
- Gate for resetting $r_t = \sigma(W_r x_t + U_r h_{t-1})$
- New Memory $h_t = tanh(r_t.Uh_{t-1} + Wx_t)$
- Hidden State $h_t = (1 - z_t).h_t + z_t.h_{t-1}$

Output of the paraphrase detection (section 4.1) is passed as an input to the generator, as shown in Figure 4.23. Inside the GRU, at every step x_t is provided as input sequence to the network. y_t from the previous step is also given to the network. The network updates the currently hidden state h_t based on the previously hidden state. Embeddings from the text are generated to be passed as the input to the generator. The probability distribution is then calculated by the network over the next element in the sequence. The softmax layer on top of the hidden layer generates discrete output. The output sequence, parameterized by θ_g is denoted as $P_{\theta_g}(y|x)$. The output of the generator is discrete and hence can't be differentiated by the discriminator. TGGAN has an additional summarization function S, which is a function of x, y, and θ_g. S is differentiable and hence can be used by the discriminator. The generated text is passed to the generator for classification. Function S also back-propagates from the discriminator to the generate. Algorithm 4.5 discusses the proposed approach for text summarization using paraphrase detection.

The discriminator maximizes the probability of differentiating correctly between the output of S on the hidden state versus the input embedding. The loss function used is a weighted sum of individual loss functions mentioned in equation 4.11 and 4.12.

$$Loss_{discrimnator} = -1/2E_{(x,y)\ data}[log(D(B(x,y,\theta_g),\theta_d))] - 1/2E_{y\ P_{\theta_g}}[log(1 - D(B(x,y,\theta_g),\theta_d))] \tag{4.11}$$

$$Loss_{generator} = -1/2E_{(x,y)\ data}[log(P_{\theta_g})] \tag{4.12}$$

Once the probability distribution is determined, sentences generated are checked for its correctness using the following grammar rules. However, these rule vary depending upon the language.

1. *. * (/nn) + . * (/vb) + . * (/jj) + .**
2. *. * (/jj) + . * (/to) + . * (/vb) + .**
3. *. * (/rb) + . * (/jj) + . * (/nn) + .**
4. *. * (/rb) + . * (/in) + . * (/nn) + .**

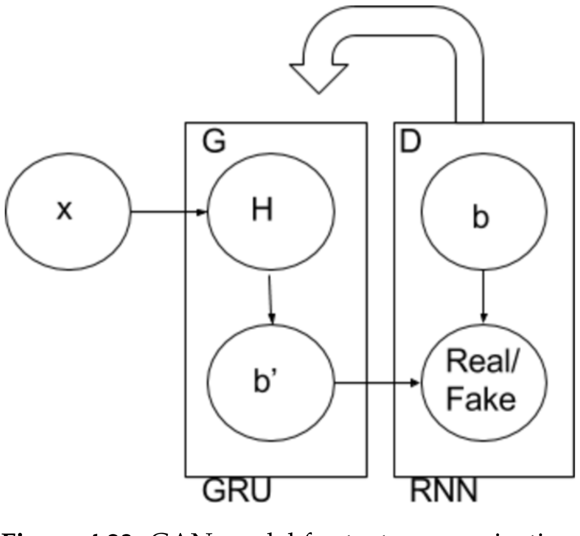


Figure 4.23: GAN model for text summarization

4.3.2.2 Dataset

- Multiling 2015 Dataset: This dataset contains hundred news articles from English, Hindi, Malayalam, etc. amongst the *100* documents, *10* news articles pertaining to

Algorithm 4.5: Text Summarization using paraphrase detection

```
input  : A document D to be summarized
output: An output Summ, the summary
1  Initialization: pooled_S=[];
2  Preprocessing;
3  Set Sentence, which contains the sentences already processed;
4  for Sentence S in D do
5      for S′ in Sen do
6          if S′ and S are paraphrases then
7              flag = true;
8      if flag == false then
9          Add S to Sen;
10 Sen is provided as the input to GRU;
11 Train the GAN, generate Probability distribution P;
12 s = “”;
13 while Summ <required length do
14     while s does not specify grammar rules and len(s) <10 do
15         s.append(predict(s));
16     if s satisfies grammar rules then
17         Add s to Summ;
```

one topic. The *10* documents are combined into one document to be given as input to the approach proposed in this work.

- Opinosis Dataset [Ganesan *et al.* 2010a]: This dataset contains product reviews in English ranging from GPS navigation system, cars, and iPod. There are *51* documents pertaining to these user reviews and each review has *100* sentences on an average.

4.3.2.3 Parameter Settings

In RNN of the proposed model, number of layers is fixed to two. The batch size for RNN and GRU is *50*. The decay rate is *0.9* and learning rate starts at *0.002*. Adam optimizer is used to determine the adaptive learning rate. Training for the generator is paused when the loss is less than *0.5* and for discriminator when the loss is less than *0.3*. The network is trained for *30* epochs. In Wasserstein GANs, weight clipping is done to make the network stable. In this network, gradients are clipped at *5*. One-sided label smoothing is performed to avoid gradient explosion.

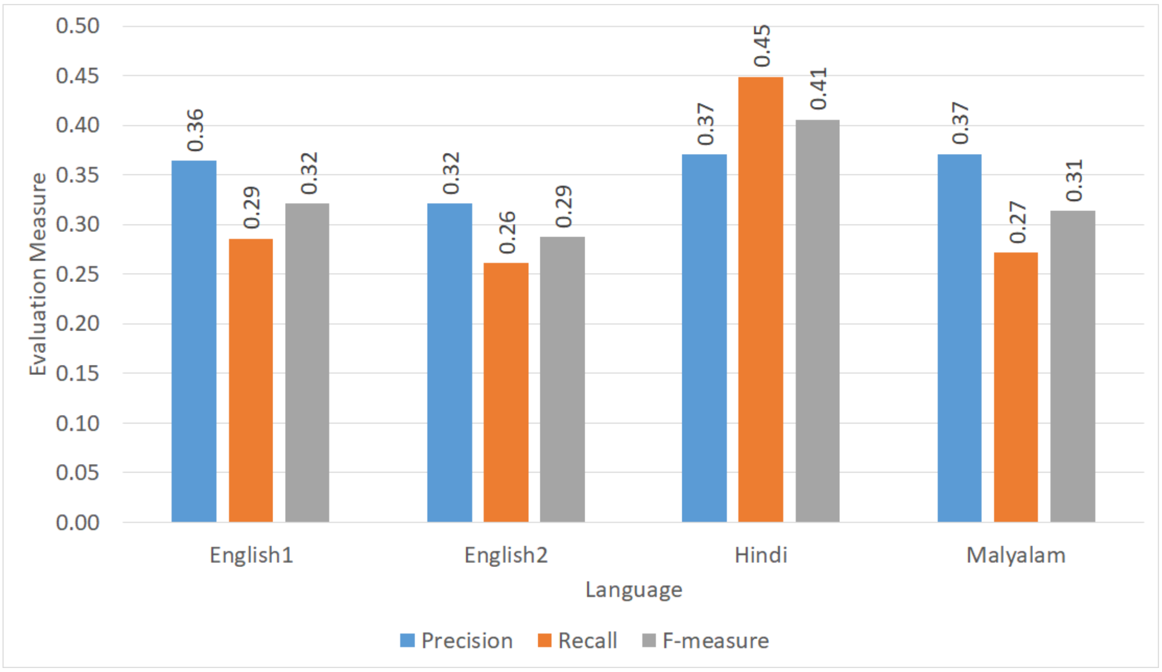
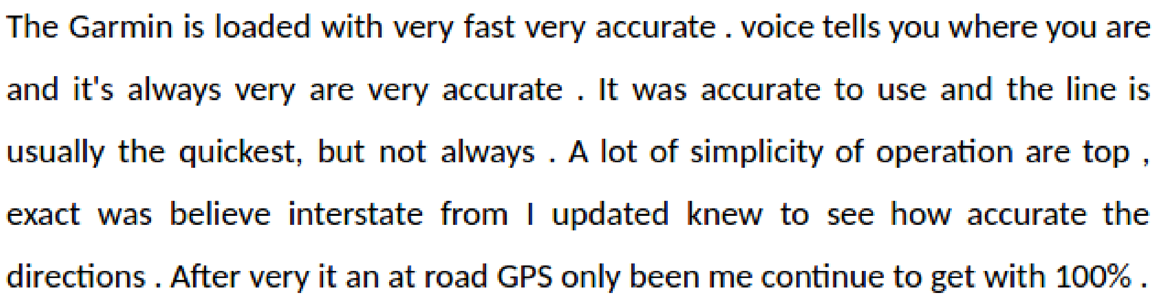
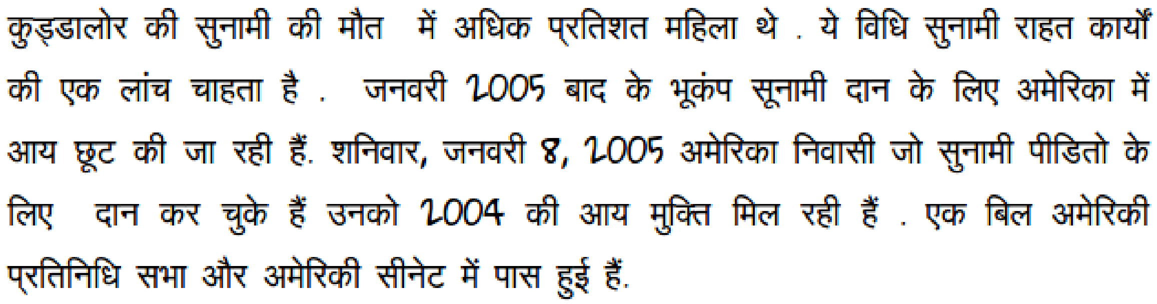


Figure 4.24: Results for Text Generation using Generative Adversarial Networks



The Garmin is loaded with very fast very accurate . voice tells you where you are and it's always very are very accurate . It was accurate to use and the line is usually the quickest, but not always . A lot of simplicity of operation are top , exact was believe interstate from I updated knew to see how accurate the directions . After very it an at road GPS only been me continue to get with 100% .

Figure 4.25: Example summary for English generated by Proposed approach (TGGAN)



कुड्डालोर की सुनामी की मौत में अधिक प्रतिशत महिला थे . ये विधि सुनामी राहत कार्यों की एक लांच चाहता है . जनवरी 2005 बाद के भूकंप सूनामी दान के लिए अमेरिका में आय छूट की जा रही हैं. शनिवार, जनवरी 8, 2005 अमेरिका निवासी जो सुनामी पीडितो के लिए दान कर चुके हैं उनको 2004 की आय मुक्ति मिल रही हैं . एक बिल अमेरिकी प्रतिनिधि सभा और अमेरिकी सीनेट में पास हुई हैं.

Figure 4.26: Example summary for Hindi generated by Proposed approach (TGGAN)

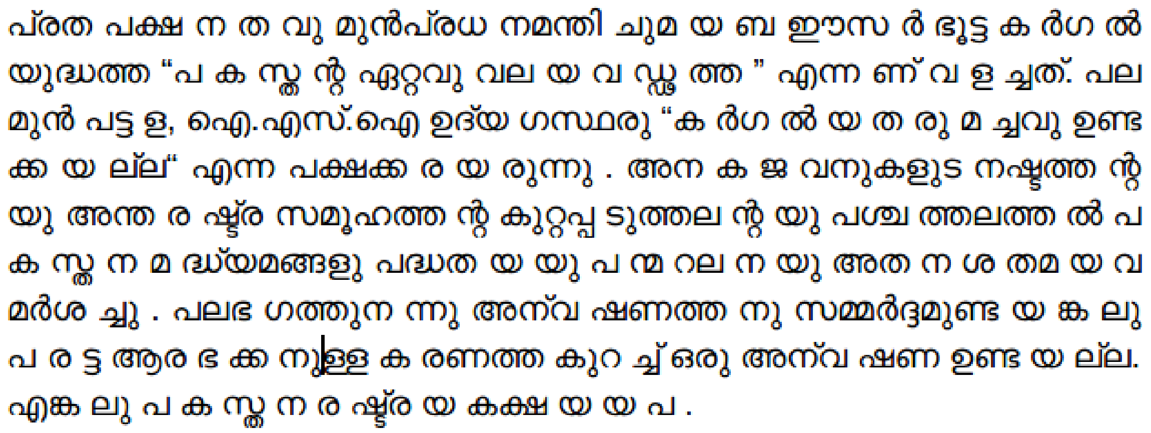

പ്രത പക്ഷ ന ത വു മുൻപ്രധ നമന്തി ചുമ യ ബ ഈസ ർ ഭൂട്ട ക ർഗ ൽ
യുദ്ധത്ത "പ ക സ്ത ന്റ ഏറ്റവു വല യ വ സ്ഥ ത്ത " എന്ന ണ് വ ള ച്ചത്. പല
മുൻ പട്ട ള, ഐ.എസ്.ഐ ഉദ്യ ഗസ്ഥരു "ക ർഗ ൽ യ ത രു മ ച്ചവു ഉണ്ട
ക്ക യ ല്ല" എന്ന പക്ഷക്ക ര യ രുന്നു . അന ക ജ വനുകളുട നഷ്ടത്ത ന്റ
യു അന്ത ര ഷ്ട്ര സമൂഹത്ത ന്റ കുറ്റപ്പ ടുത്തല ന്റ യു പശ്ച ത്തലത്ത ൽ പ
ക സ്ത ന മ ദ്ധ്യമങ്ങളു പദ്ധത യ യു പ ന്മ റല ന യു അത ന ശ തമ യ വ
മർശ ച്ചു . പലഭ ഗത്തുന ന്നു അന്വ ഷണത്ത നു സമ്മർദ്ദമുണ്ട യ ങ്ക ലു
പ ര ട്ട ആര ഭ ക്ക നുള്ള ക രണത്ത കുറ ച്ച് ഒരു അന്വ ഷണ ഉണ്ട യ ല്ല.
എങ്ക ലു പ ക സ്ത ന ര ഷ്ട്ര യ കക്ഷ യ യ പ .

Figure 4.27: Example Summary for Malayalam generated by Proposed approach (TGGAN)

4.3.2.4 Evaluation and Discussion

Results of the Multiling dataset are being referred here as English$_1$ whereas results of Opinosis dataset are being referred as English$_2$. For evaluation of the summaries, ROUGE [Lin 2004] is used. For Opinosis dataset, the result for English summaries is significantly lesser than that of Multiling English summaries as can be seen from Figure 4.24. The reason for this could be that opiniosis dataset contains more redundant sentences and as such removing them from the dataset greatly reduces the input text to the GAN. For Hindi and Malayalam, the precision is higher than that of English. It can be seen from Figure 4.24, the same pattern between English$_1$ and English$_2$ is observed for recall as well. The F-measure score for Hindi is the highest which might be because of the language semantics being in play. For Opinosis dataset, the F-measure score is the lowest (Figure 4.24) amongst all which is a result of low precision and recall. Figure 4.25, 4.26 and 4.27 shows sample summaries generated for English, Hindi, and Malayalam respectively.

4.4 Concluding Remarks

Two different approaches are proposed for each problem (Paraphrase detection, Extractive text summarization, and Abstractive text summarization) targeted in this chapter. A feature vector based approach with three features (POS tags, word stems and Soundex codes) is discussed for paraphrase detection of Hindi language. Levenshtein distance is used to calculate the similarity measure. The proposed system achieved an accuracy of

89.7% and F-measure of *89%* for identification of binary class paraphrase detection using Logistic Regression. For ternary classification, the proposed system gave an accuracy of *71.7%* and F-measure of *71.2%* using Random Forest classifier. In second proposed approach for paraphrase detection, a series of experiments with CNN(s) and RNN(s) are described. Despite little tuning of hyperparameters, a simple CNN with one layer of convolution performs on par with the existing Deep Learning Architectures. LSTM(s) and Bi-LSTM(s) also perform on par with CNN.

The first approach for extractive text summarization outlined present linguistically independent feature for extracting important text and used a voting based approach to classify the sentence. A comparison is made with the baselines using RF, SVM, and NB. The second approach proposed for extractive text summarization using deep learning where training dataset is built using the algorithm of paraphrase detection such that each sentence is classified as a summary sentence or not based on the fact that it matches with the sentence of human-generated gold summary or not. Fully connected CNN is used to train the model. It is observed that a recall of *0.428* is obtained for English whereas a recall of *0.846* is obtained for Malayalam.

Evaluation of proposed algorithm ATSSI, using DUC *2002* dataset[1] and on the dataset used by [Ganesan *et al.* 2010b], outperforms the abstractive summarization algorithms described by [Lloret & Palomar 2011] and [Ganesan *et al.* 2010b]. ATSSI is able to leverage the sentence word order to form coherent sentences, and hence generated summaries are concise and able to communicate the information in conjunction with the ability to remove the redundancy from the input text. No domain knowledge is required for the proposed algorithm (ATSSI) to work and hence it is adaptable to different types of content as well. The second approach proposed for abstractive text summarization, a generative adversarial network is used to perform multilingual text summarization. The results are at par with existing deep learning frameworks for summarization. Improvements can be made in terms of hyper-parameter tuning and size of dataset. The input used to train the network needs to be larger to achieve better results. The current work uses significantly lesser data to train the network than the other text generation models.

[1]http://duc.nist.gov/data.html

Chapter 5

Spam Detection in Reviews

Opinions influence almost all decisions of humans. The day-to-day choices that we make in our lives are based on how our peers and friends perceive the world. Be it an organization or an individual, opinions are an integral part of the decision making process. The study of emotions, opinions, choices, sentiments and behavioural patterns is known as opinion mining, also called sentiment analysis. With the growth of social media, organizations and individuals depend highly on the media content for decision-making. These online reviews help customers, companies, and vendors to make decisions regarding quality of products and services. Social media allows an individual to post any feedback, reviews or comments regarding a product, service or organization anonymously. Due to this anonymity, people with hidden motives or malicious intentions post deceptive opinions or feedback to misguide or wrongly influence people on social media. Such activities are called opinion spamming [Jindal & Liu 2008]. Such content on social media falls into the category of opinion spam, and the people are called as opinion spammers. It is thus essential to address the problem of review spam on social media.The need of the hour is to have a technique capable of analyzing the truthfulness of reviews to assist with the decision-making process or for marketing intelligence.

Product reviews are a progressive way of generating online consumer content as they offer valuable insight into user preferences and requirements. The World Wide Web contains a vast amount of user reviews and opinions on various products expressed in newsgroups and sites. As a result, opinion mining has recieved increasing attention over the last few decades [Jindal & Liu 2008]. The e-commerce websites have made it a practice to let con-

sumers voice their opinions and publish reviews for various products on their websites and mobile apps. These reviews give customers the analytical information to broaden their scope with more competitors who are providing the same product/services [Algur *et al.* 2011]. Product manufacturers get to know consumer preferences/interests, as well as the positive/negative attributes of their products and that of the competitors and hence, can take necessary action by which profits could be maximized. [Dave *et al.* 2003].
Several researchers have studied the problem of deceptive spam reviews [Feng *et al.* 2012b], [Li *et al.* 2014]. Various review filtering systems have been developed and are being used by companies such as Yelp and Dianping to detect low quality and fake reviews on their product pages. These systems help alleviate the negative impact of fake reviews. Many spammers work collectively to promote or demote a set of target products so that they can work without leaving a trail [Li *et al.* 2015]. Experiments of [Ott *et al.* 2011] suggested that an average accuracy of detecting spam by three human judges is *57.33%* and hence, the research for detection of deceptive opinion spam is necessary and meaningful. The problem of spam is troublesome and poses a security threat as well. However, it is challenging to filter out a spam review or capture spammer behaviour, even if done manually. The objective of this chapter is to distinguish whether a review is a spam or truth (not spam). The task can be transformed into a binary class classification problem. In traditional machine learning techniques, feature engineering is essential. However, the inherent law of data from a semantic perspective can hardly be learned from it. Deep learning refers to a set of algorithms that attempt to learn in multiple levels, corresponding to different levels of abstraction. It is nothing but a very large or deep neural network which automatically learns feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features. Deep learning algorithms have performed well on various natural language processing tasks such as paraphrase detection [Huang 2011b], POS tagging [Popov 2016], language modelling [Sutskever *et al.* 2014] and sentiment analysis [Socher *et al.* 2013]. This chapter presents an experimental study and its analysis on variants of sequential models based on deep network architecture. It also compares the proposed approach, and its variants explained in section 5.1 to that of the baseline approach (discussed in section 5.2) based on traditional machine learning techniques to analyze the effectiveness of the proposed approach.

5.1 Composite Sequential Modeling for Identifying Fake Reviews

Proposed approach composite sequential modeling for identifying fake reviews, comprises of multiple phases as shown in Figure 5.1. In pre-processing phase, the raw reviews are taken and converted into lower case. The vocabulary of the entire data set is extracted from the reviews, and overall frequency of each word is calculated. A sorted dictionary of all the words along with their frequencies is generated, and the most frequent word is labelled *1*, the second most frequent word is labelled *2* and so on. Words in the raw reviews are replaced by generated labels correspondingly, and in this way, each review is encoded as a sequence of word indexes (integers). Custom vectorization is used instead of pretrained vectors, to create context specific word embedding.

In sequential models phase, different sequential models are generated for classification

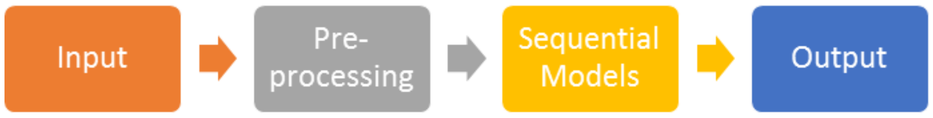


Figure 5.1: Block Diagram for Proposed Approach

of the reviews using architectures explained in section 5.1.1, 5.1.2, 5.1.3. The first layer of each model is an embedding layer, which converts positive integers (indexes) into dense vectors of fixed size. Word embeddings help in associating a numeric vector to every word in a dictionary. Ideally, words with similar semantic meaning are closer in the embedding space. The embedding layer converts each word encoding (index) to a vector in embedding space. The vectors are initialized randomly and then optimized iteratively. The input parameters for embedding layers are input_dim, output_dim and dropout. The input dimension is the size of the vocabulary (*20,000* for all the models implemented), and the output dimension is the dimension of the dense embedding vectors (*512* for all the models). Dropout is varied between *0* and *1* (*0.2* for this specific model), which corresponds to the fraction of the embeddings to drop.

5.1.1 Parameter setting for Recurrent Neural Network (RNN)

The parameters set for the RNN layer of the proposed model are dimensionality of the output layer (*256* for this layer), *tanh* as the activation function and the dropout (fraction of units to drop for the linear transformation of the inputs) as well as the recurrent_dropout (fraction of units to drop for the linear transformation of the recurrent state) set at *0.2*.

5.1.2 Parameter setting for Long Short Term Memory (LSTM)

The parameters of the LSTM layer are the dimensionality of the output layer (kept at *256*), activation function (*tanh*), the recurrent_activation function (*hard_sigmoid*), dropout as well as recurrent_dropout (kept at *0.2*).

5.1.3 Parameter setting for Convolutional Neural Network (CNN)

While building CNN architecture, some of the parameters to choose from are input representations, i.e. embedding layer, number and size of convolution filters, pooling layer (*max or average*) and activation functions (*tanh or ReLU*). The convolution layer is the primary building block of a CNN which comprises of a set of independent filters. The pooling layer progressively reduces the spatial size of the representation to reduce the number of parameters and computation in the network and operates on each feature map independently. After the activation function layer, CNN usually has a fully connected layer at the end in which neurons in the layer have full connections to all activation in the previous layer. CNN(s) are usually very fast and give reasonable results for NLP task. *1D Convolution* layer is used, which applies the convolution operator for filtering neighbourhoods of one-dimensional input. The number of convolution kernels or dimensionality of output is set as *250*, the kernel_size or the length of the *1D Convolution* window is kept at *3* with the activation function *ReLu* and a stride of *1*. There was no padding applied to the input. The next layer is a *GlobalMaxPooling1D* layer which does the max pooling operation. A dropout layer with value *0.2* is added to prevent over-fitting.
In each of the model discussed in section 5.1.3, 5.1.2, 5.1.1, the penultimate layer is a *Dense* layer, which is an entirely connected neural network with the output as the corresponding label of the review. The last layer of each model is an activation layer that applies an acti-

vation function (*'Sigmoid'* in the proposed models) to the output. After generation of the models, they are compiled using an optimizer (*'adam'*, for this model) and a loss function (*binary_crossentropy* for the implemented model) with an objective that the model tries to minimize. The optimizer *'adam'* uses *0.001* as the default learning rate. After compilation, the model is trained in a batch size of *32*. Finally, models are evaluated, batch by batch for the loss and model accuracy.

5.2 Baseline Approach

Baseline approach is based on traditional machine learning techniques for comparing the results obtained using the proposed approach. One of the key ingredients for traditional machine learning techniques to work is the extraction of manually engineered features, by which the model tries to learn and predict class labels. [Chen *et al.* 2015] uses shallow syntactic features (Bag of words, POS-n-gram, punctuation, hotel name), discourse parsing features and sentiment features to build the classification model. The baseline model chosen to compare proposed work implements the shallow features and sentiment features along with some more additional features such as Bag of uni-grams and bi-grams for more contextual information, length of review and subjectivity of the review. Length of a review and subjectivity are chosen based on some heuristics which suggest that truthful reviews are usually long and descriptive as compared to spam ones which are comparatively more vaguer [Li *et al.* 2014].

5.2.1 Features

1. Bag of Words *(F_1)*:

   CountVectorizer [Pedregosa *et al.* 2011b] is used to convert each review into its corresponding vector for frequencies of each word in the review.

2. Punctuation *(F_2)*:

   A binary feature to indicate punctuation ('?' or '!') in the review.

3. POS-n-gram *(F_3)*:

   POS uni-gram and POS bi-gram tags are used as they help maintain the structure of the sentence.

4. Bag Of Uni-grams and Bi-grams *(F_4)*:

   Instead of the bag of words, the bag of uni-grams and bi-grams are considered as features. CountVectorizer [Pedregosa *et al.* 2011b] is used for generating the bag of uni-grams and bi-grams.

5. Hotel_Name *(F_5)*:

   This feature indicates whether *Hotel_Name* appears in the first line of the review.

6. Length of the Review *(F_6)*:

   An integer indicating the length (number of words) of each review.

7. Subjectivity *(F_7)*:

   TextBlob [Pedregosa *et al.* 2011b] is used to calculate subjectivity of each review. It returns a float value *[0.0, 1.0]*. *0.0* means very objective and *1.0* means very subjective.

8. Sentiment Words *(F_8)*:

   Three separate features corresponding to sentiment words are extracted : positive sentiment words, negative sentiment words and total sentiment words. The average number of sentiment words used in the review is considered.

9. Polarity of the review *(F_9)*:

   The range of the polarity is *[0.0, 1.0]* where *0.0* means the review is entirely negative and *1.0* means highly positive review.

The feature vector is generated using features (*F_1 to F_9*) extracted. Classification techniques such as Logistic Regression, Naive Bayes and Support Vector Machine are used to build the model.

5.3 Experiments and Results

In this section, data set used for experiments is described along with the results obtained. The baseline approach is used to evaluate and compare the results obtained by the proposed approach. All the proposed sequential models, as well as the model for the baseline approach, are coded in python using machine learning libraries such as scikit-learn [Pedregosa *et al.* 2011a] and keras [Chollet *et al.* 2015].

5.3.1 Data Set Description

The data set used for experimentation is the Deceptive Opinion Spam Corpus v1.4 [Ott *et al.* 2011]. The corpus contains *1600* truthful and deceptive hotel reviews of *20* Chicago hotels. The data set consists of the review, polarity of review along with the label of the review (*0* for not spam and *1* for spam). There are *400* deceptive and *400* truthful reviews of positive polarity. Similarly, there are *400* deceptive and *400* truthful reviews of negative polarity. The best state of art accuracy for the data set "Deceptive Opinion Spam Corpus v1.4 is *89.5%* which is implemented by [Chen *et al.* 2015]. Th best model combines Bag of Words, syntactic features and discourse parsing features.

5.3.2 Evaluation & Discussion

This section discusses the evaluation of the proposed approach and its variants. Dataset described in Section 5.3.1 has been split into *80:20* ratio for training and testing respectively.

5.3.2.1 Results for Baseline Approach

Features play an important role in traditional machine learning techniques and hence, individual features are evaluated using Logistic Regression model. It can be observed from Figure 5.2, the bag of words feature yields the maximum accuracy of *86.25%*, followed by *Hotel_Name (64.58%)*. The rest of the features gives accuracy in the range of *45-55%* as depicted in Figure 5.2. F-measure for the bag of words is *85.96%*. *Hotel_Name* feature gave F-measure of *63.52%*. Rest of the features yields a result in the interval of *43-60%* for F-measure. The similar trend is observed for precision. However, a slightly different trend observed for recall values as shown in Figure 5.2. The maximum recall is obtained for the bag of words *(87.83%)*. Positive polarity feature yielded the second-best value of recall *(82.61%)* followed by the length of review *(73.04%)*. The rest of the features are in the interval *40-65%* for recall. Overall, the bag of words features obtained the best results using Logistic Regression. This may be because the data set is engineered for spam detection and contains words that majorly contribute towards the improvement of the model. The result for the combined feature vector that is generated using the individual features is evaluated using three different classifiers (Support Vector Machine, Naive Bayes, and Logistic Regression) as shown in Figure 5.3. Logistic Regression gives the best results concerning accuracy *(87%)*, F-measure *(86.58%)*, precision *(86.2%)* and recall *(86.96%)*, followed closely by Naive Bayes which gave an accuracy and F-measure of *73%*, precision *70%* and recall *77%*. The results obtained by logistic regression are better than the others because spam detection is a binary classification problem. SVM(s) are usually more useful in the case of non-linear classification since they use kernel trick. The results range in *53-56%* using SVM.

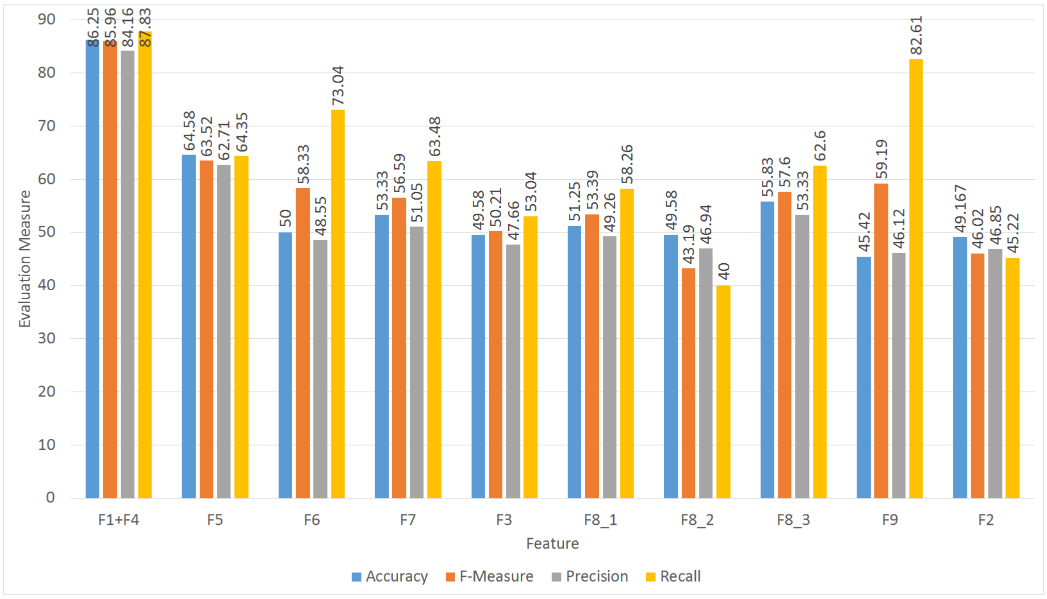


Figure 5.2: Results obtained at Feature Level

5.3.2.2 Results for Proposed Approach

The accuracy obtained for different variant sequential models based on LSTM, RNN, and CNN are shown in Table 5.1, 5.2 and 5.3. Each model is evaluated using different number of layers as a varying parameter.

Table 5.1, 5.2 and 5.3 discuss results obtained for LSTM, RNN, and CNN respectively for *1, 2 & 3* layers. In all the three cases, the best results are obtained in the case of a single

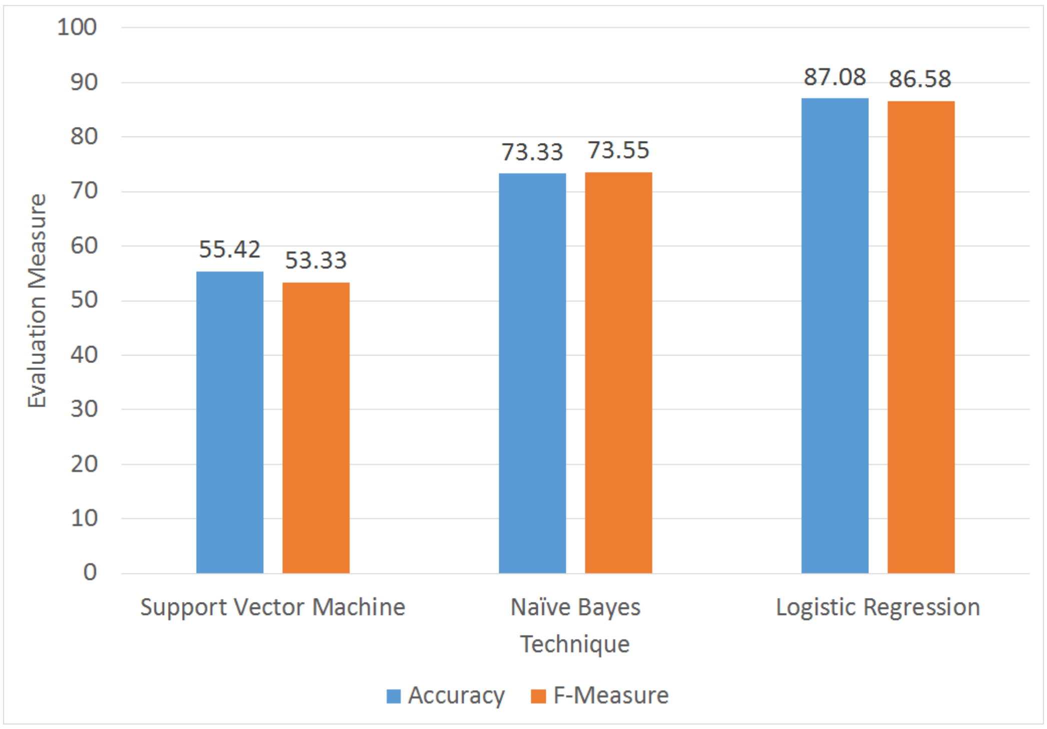


Figure 5.3: Comparison of different techniques

layer model as shown in Figure 5.4. CNN gave an accuracy of *81%*, and LSTM gave *80%* accuracy when a single layer is used whereas RNN gave an accuracy of *59%* for a single layer. LSTM performs better than RNN because it considers long-term dependencies as well. CNN helps in extracting position invariant features whereas RNN and LSTM are useful for modelling units in sequence. Since the task at hand is to detect spam reviews, pattern detection is essential. LSTM preserves long-term structure whereas CNN is good at classification. Hence, CNN gives slightly better results as compared to LSTM. The results show that increasing the number of layers of a particular model reduces the accuracy of the model in case of spam detection.

While implementing a two-layer model where each layer is either RNN, LSTM or CNN, the best results are obtained in the model where the combination of LSTM and CNN is implemented. Table 5.4 shows the results for two-layered models, where *1* represents the first layer, *2* represents the second layer, and *0* represents no layer. It can be observed that change in the placement of layer in case of LSTM and CNN model affected the results. The accuracy of the model with the first layer as LSTM and the second layer as CNN is *76.25%* whereas it is *76.67%* when the first layer is CNN layer and the second layer is LSTM. The

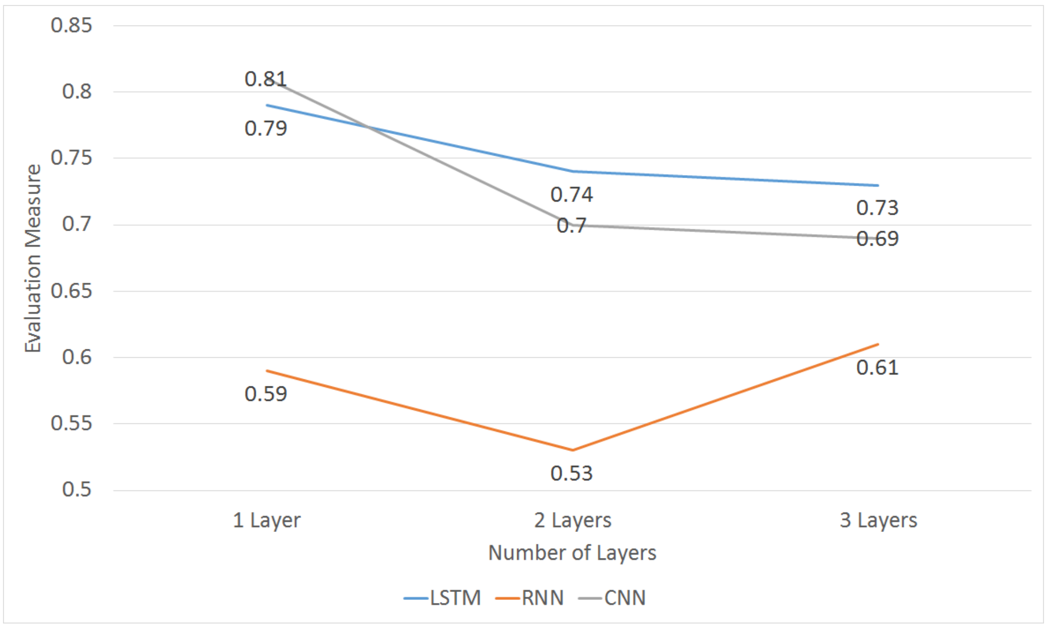


Figure 5.4: Comparison of layered models of different architecture

precision, recall, and F-measure for LSTM-CNN network is *75.65%* when CNN is the first layer whereas precision is *71.32%*, recall *84.35%* and F-measure *77.29%* when LSTM is the first layer. The overall accuracy of the model is comparatively less when a layer of RNN is introduced in the model because RNN is a less efficient deep learning algorithm as compared to LSTM. The least accuracy is obtained in the combination of RNN and LSTM (*69%* and *74%*). CNN is a better network architecture for text classification as discussed above and hence, poor results are obtained when CNN layer is missing in the architecture. When the first layer is LSTM, better results are obtained with precision of *71%*, recall of *76.52%* and F-measure of *73.64%*. This is due to the fact that LSTM performs better than RNN because of long term dependencies. As second layer takes input from the first layer in a neural network, the more efficient the first layer is, better the feature extraction and hence, improved results of the overall network.

Table 5.1: Results for different model of LSTM

Model	Accuracy	Precision	Recall	F-Measure
1 Layer	0.79	0.77	0.8	0.78
2 Layers	0.74	0.71	0.79	0.75
3 Layers	0.73	0.68	0.87	0.76

In case of three-layered model variant, different combinations are build and evaluated

Table 5.2: Results for different model of RNN

Model	Accuracy	Precision	Recall	F-Measure
1 Layer	0.59	0.57	0.62	0.59
2 Layers	0.53	0.5	0.59	0.54
3 Layers	0.61	0.58	0.63	0.6

Table 5.3: Results for different model of CNN

Model	Accuracy	Precision	Recall	F-Measure
1 Layer	0.81	0.78	0.84	0.81
2 Layers	0.7	0.66	0.8	0.72
3 Layers	0.69	0.65	0.76	0.7

as shown in Table 5.5, where *1st* layer represents that the particular model is used as a *1st* layer of the sequential model. *2nd* layer and *3rd* layer depict the respective positions of a layer in the model. The best result is obtained when CNN is used as the first layer followed by RNN and LSTM *(78.75%)*. Further, an accuracy of *74.58%* is obtained when RNN is used as the first layer, followed by LSTM and CNN. A common trend is observed that best results are obtained when CNN is kept as a first layer. This is because CNN performs better feature extraction than LSTM or RNN. Moreover, worst accuracy is obtained when the first layer is LSTM, followed by RNN and CNN layer *(57.5%)*. Output dimensions of embedding layer are also varied to evaluate the effect on accuracy. Best results are obtained in the proposed approach when the output dimensions are *512 (81.25%)*. As the number of dimensions increases in the embedding layer, the accuracy of the neural network increases, as shown in Table 5.6. More the dimensions, better is the representation of the reviews in vector space, and hence the accuracy increases with an increase in the number of dimensions. However, change in accuracy became near constant after *512* dimensions. Figure 5.5 shows the ROC curve for the best results obtained in the proposed approach (*81.25%* accuracy for single-layer CNN model). The AUC or Area under the ROC Curve is shown in Figure 5.5 is *0.8137*.

5.3.2.3 Comparison of Proposed Approach with Baseline Approach

The baseline approach with the variant of Logistic Regression as machine learning technique gave a maximum accuracy of *87%* for the hotel reviews dataset whereas proposed

Table 5.4: Results for two-layered model

LSTM	SimpleRNN	CNN	Accuracy	Precision	Recall	F-Measure
0	1	2	0.75	0.74	0.76	0.75
0	2	1	0.75	0.71	0.82	0.76
1	0	2	0.76	0.71	0.84	0.77
1	2	0	0.74	0.71	0.77	0.74
2	0	1	0.77	0.76	0.76	0.76
2	1	0	0.69	0.68	0.68	0.68

Table 5.5: Results for three-layered model

LSTM	RNN	CNN	Accuracy	Precision	Recall	F-Measure
1st Layer	2nd Layer	3rd Layer	0.575	0.5436	0.7044	0.6136
1st Layer	3rd Layer	2nd Layer	0.5792	0.5603	0.5652	0.5627
2nd Layer	1st Layer	3rd Layer	0.7167	0.688	0.7478	0.7167
2nd Layer	3rd Layer	1st Layer	0.7333	0.6946	0.7913	0.7398
3rd Layer	1st Layer	2nd Layer	0.5917	0.5714	0.5913	0.5812
3rd Layer	2nd Layer	1st Layer	0.7875	0.75	0.8348	0.7901

Table 5.6: Result for different model of CNN

Dimensions	Accuracy	Precision	Recall	F-Measure
512	0.8125	0.7822	0.8435	0.8117
256	0.7917	0.748	0.8522	0.7967
128	0.7917	0.77	0.7913	0.7844
64	0.7792	0.7384	0.8347	0.7837
32	0.7625	0.7101	0.8522	0.7747
16	0.7083	0.6552	0.8261	0.7307

approach gave a maximum accuracy of *81.25%* which is comparable to traditional machine learning algorithms, if not better. Deep learning algorithms give much better results with a huge amount of data. Since the data considered for all the above-obtained results is just *1600* reviews, which is very less for a deep learning model, the results obtained by proposed approach didn't surpass the baseline approach.

5.4 Concluding Remarks

The proposed approach suggests that deep learning sequential models perform comparably and are well suited to address the problem of review spam detection. A single layer LSTM, CNN or RNN model outperforms multi-layer sequential models of LSTM, CNN or RNN. Single-layer LSTM model and single-layer CNN model give *79%* and *81.25%*

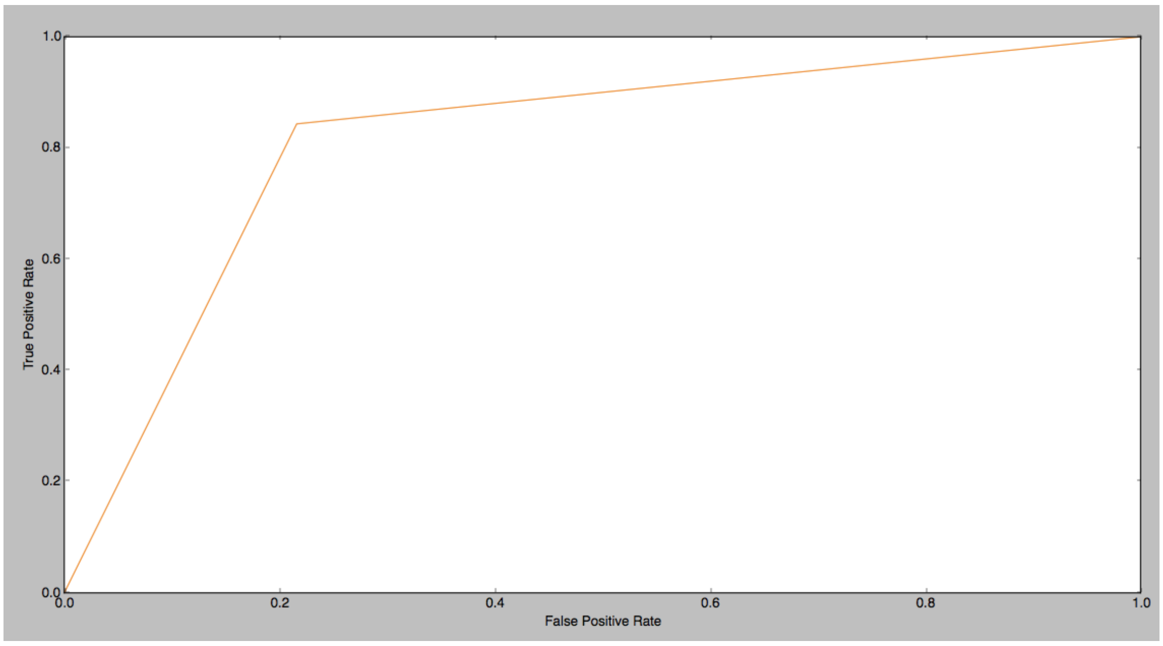


Figure 5.5: ROC Curve for single-layer CNN

accuracy respectively whereas single-layer RNN model gives an accuracy of *59%*. When a combination of LSTM, CNN or RNN is used in a two-layered sequential model, the CNN-LSTM combination gives the best results (*77%* accuracy). An accuracy of *78.75%* was obtained when the first layer of a three-layered sequential model implemented CNN, second layer RNN followed by LSTM in the last layer. For the CNN model, results have improved with an increase in the number of output dimensions of the embedding layer in the deep network model. Larger networks having more number of dimensions help capture information better and hence yield better results. An accuracy of *81.25%* was obtained when the number of output dimensions of the embedding layer was *512*. A comparison is drawn between traditional machine learning algorithms and deep learning algorithms for spam review classification. Baseline approach resulted in an accuracy of *87%* using Logistic Regression. The results show that traditional machine learning model gives a higher accuracy for small sized data. However, the deep learning models are not far behind. With more massive datasets, deep learning techniques are likely to surpass traditional machine learning algorithms. In future work, the aim is to implement these sequential deep learning models on more massive datasets and then compare the performance.

Chapter 6

Sarcasm Detection

“Sarcasm is the use of words that mean the opposite of what you really want to say especially to insult someone, to show irritation, or to be funny” as indicated by [Merriam-Webster 1983]. Automatic sarcasm detection refers to computational ways to detect sarcasm in text. A computer needs to make sense of what you implied inverse of what you just said to identify Sarcasm. The onset of Web applications, such as small-scale blogging sites, forums and social networking sites have given clients a platform to express their reviews, comments, recommendations, ratings and feedback. These applications have given the user a stage to express anything they feel about any politician or product or, people or event. This data if mined can be of great use to companies, politicians, service providers, social psychologists, analysts and researchers. Usually, companies might want to know if a user liked their product or not. Sentiment analysis tools perform this task for a company, but it may provide the company with wrong information, if it cannot analyze sarcasm in the review. Hence, sarcasm is a challenge for sentiment analysis tools. This motivates to work on sarcasm detection, which can be used in conjunction with sentiment analysis tool to give the company a correct idea about the response from the users for its product. If a system can reliably identify sarcasm and irony in the text to infer the actual communication intention of the author then it can improve the performance of many natural language processing systems including summarization, sentiment analysis, opinion mining and advertising.

Sarcasm is defined as the use of remarks that apparently mean the opposite of what they say, made to hurt someone’s feelings or to criticize something humorously. Sentiment analysis and related fields of study are continually gaining popularity as it captures the human sentiment underlying in their opinions, reviews for various purposes like opinion mining for products and polling for a global phenomenon. This creates a need for correctly determining the hidden sentiment in the text. Thus, sarcasm detection is a crucial key in this process due to its property of showcasing a false positive sentiment behind a true negative sentiment and vice-versa.

In the speech, sarcasm is quite indicative as it involves tone, body language and gestures, which makes its detection relatively easy. However, sarcasm in text is restrictive and cannot encompass such a range of auxiliary features. Thus, sarcasm detection is challenging task in the field of NLP. This is apparent from the results reported by the survey [Joshi *et al.* 2016a].

Often, knowledge of the universal truth on a given topic is insufficient to detect sarcasm, since the text may refer to some specific events to express sarcasm. The same sentence can be sarcastic or not sarcastic depending on the given context. For example, the sentence "You have been working hard, I see." is difficult to be classified as sarcastic or not sarcastic without provision of an explicit context. Apart from this, the use of phrases "working hard" and "I see" express semantic contrast and points to the presence of sarcasm. Thus, quite often, one needs a considerable amount of facts and contextual knowledge to draw such a conclusion. This makes it quite difficult for even humans to sense sarcasm.

The high importance of sarcasm detection has motivated researchers to work on the problem of sarcasm detection. Some earliest work in this field is done by [Tepperman *et al.* 2006]. He dealt with speech and text-related features. Sarcasm detection from text now has been extended to tweets and reviews. Researchers have used several techniques for sarcasm detection such as rule-based, supervised and semi-supervised. This synergy has resulted in interesting innovations for automatic sarcasm detection. Pattern based features [Tsur *et al.* 2010] are extracted from the text in conjunction with lexical features for identifying sarcasm in text.

This chapter discusses three proposed solutions for sarcasm detection. The first approach presented in section 6.1 models the problem of sarcasm detection as a classification task and uses traditional machine learning techniques. Second approach presented in section 6.2 is based on code mixed data and uses deep learning technique to identify the sarcasm in code mixed text of English and Indian languages.

6.1 Sarcasm Detection using Machine Learning Techniques

In this section, an approach based on traditional machine learning technique is explained. Manually engineered features are elaborated in section 6.1.3. Further, different variations and combinations of these features are experimented to identify the best combination of features in section 6.1.6. The approach observes the results by combining existing features in state of the art and how they effect the overall results of sarcasm detection.

6.1.1 Dataset Description

The dataset used for training consists of *1,254* Amazon reviews out of which *437* are ironic, and *817* are non-ironic. It has been acquired by [Filatova 2012] using the crowdsourcing platform amazon mechanical turk. The process of acquiring data included following steps. In the first step, employers are asked to find pairs of reviews on the same product so that one of the reviews is sarcastic, but the other one is not. After that they have to submit the ID of both reviews, and in the case of an ironic review, they have to provide the fragment conveying the irony. In the second step, five additional workers annotated each collected review. Annotated review remained in the corpus if three of the five new annotators concurred with the initial category, i.e., ironic or non-ironic. This dataset is most suitable to the problem statement defined above because each review in this corpus is at least three sentences long.

6.1.2 Data Pre-processing

In pre-processing step, review is tokenized into sentences and sentences are tokenized into words. Treebank Word Tokenizer was used to tokenize sentences to words and Punkt Tokenizer was used to tokenize reviews into sentences. These tokenizers are present in NLTK [Bird & Loper 2004]. With each word, a POS tag is attached. The sentiment is also attached to each word which indicated the word's polarity which can be positive, negative or neutral. Polarity lexicon by [Hu & Liu 2004] which consists of about *6800* words is used to determine the polarity of words. A sentiment score is assigned to each sentence using python library [Li *et al.* 2017b]. Therefore, each review is considered as an object with attributes like words, bi-grams and sentences. Each word has polarity and POS tag. Each sentence has sentiment score associated with it.

6.1.3 Feature Extraction

Following features are extracted to train the model:

1. Star Rating: With every review, a star rating is attached which signifies how satisfied is the user with the product. The rating is between *1* and *5*. Dataset already contains the star rating for each review.

2. Star Polarity Discrepancy: If the rating given by the user is high, i.e. *4* or *5* but the sentiment of the review is negative or vice versa. *1* and *0* is appended for the presence and absence of this feature respectively.

3. Bag of Words: A dictionary is built which contains all the words present in reviews which are used for training. A feature vector for each review is then built using this dictionary. For all the words present in the dictionary, if that word is present in a given review, *1* is added to feature vector else *0* is added.

4. Bag of bi-grams: A dictionary is built which consists of all the bi-grams present in reviews which are used for training. A feature vector for each review is then built using this dictionary. For all the bi-grams present in the dictionary if that bi-gram is present in a given review *1* is added to feature vector else 0 is added.

5. Hyperbole: It is an extreme exaggeration used to make a point. Presence of it in a sentence indicates that the sentence can be sarcastic. Therefore if a sequence of three positive or three negative words was found in a sentence of a review, then *1* is added to feature vector else 0 is added.

6. Scare Quotes: If two consecutive adjectives or nouns or adverbs having a positive or negative polarity occurs in quotations in a sentence then *1* is added to feature vector else *0* is added. It is also considered as exaggeration and an indicator of sarcasm.

7. Ellipsis plus Punctuation: When an ellipsis is followed by two or more exclamation marks or question marks or combination of them in a review then 1 is added to feature vector else 0 is added.

8. N-gram plus punctuation: If a span of up-to four positive or four negative words occur in a row followed by punctuation symbols like an exclamation mark or question mark or combination of both then, 1 is added to feature vector else 0 is added.

9. Emoticons: To extract this feature, a list of emoticons is created. Three annotators are asked to label these emoticons as positive, negative or sarcastic. Final label is decided based on the majority count. If an emoticon expressing positive sentiment occurred in a sentence but the sentiment of the sentence is negative or vice versa then *1* is added to feature vector else *0* is added. If sentence contained any sarcastic emoticon, then *1* is added indicating the presence of sarcasm.

10. Interjections: For extracting this feature, a list containing *155* interjections is built. Three annotators are asked to label these interjections as positive, negative or neutral. Final label is decided based on the majority count. If an interjection expressing positive sentiment occurred in a sentence but the sentiment of the sentence was negative or vice versa then *1* is added indicating the presence of sarcasm.

11. Capitalization: If a word is written in upper case in a sentence then there is a high probability that sentence can be sarcastic and hence the number of words present in the upper case in a review are taken as a feature. The letter *'I'* is not considered as upper case word as *'I'* is always capitalized.

12. Pattern based features: Words whose corpus frequency is less than 2000 words per million are considered as low frequency words, also called as Content Word (CW). Remaining words are considered as high frequency words. Punctuation symbols are also considered as high frequency words. In the review, a content word is replaced by 'CW'. After this classification, patterns are extracted from reviews where each pattern must begin and end with a high frequency word, and there must be one content word in between them. Patterns of length *3, 4, 5, 6* and *7* are extracted. Each review is then matched with all the patterns, and thus a feature vector is made for

each review. Pattern matching assigns a score to each review using the following:

1 : (Exact match) all the pattern components match

alpha : (Sparse match) same as an exact match, but additional non matching words can be inserted between pattern components.

gamma * n/N : (Incomplete match) only n *>1* of *N* pattern components appears in the sentence, while some non-matching words can be inserted in-between.

0 : No match

alpha and gamma is taken as *0.1*. As the patterns are relatively long, exact matches are uncommon, and taking advantage of partial matches allows us to reduce the sparsity of the feature vectors significantly. For sentence "Brisbat apparently does not care much about product quality or customer support", the value for "Brisbat CW does not" would be 1 (exact match); for "Brisbat CW not" would be 0.1 (sparse match due to insertion of 'does'); and for "Brisbat CW CW does not" would be $0.14/5 = 0.08$

6.1.4 Classification

Existing approaches for sarcasm detection have mostly used following traditional machine learning techniques and hence, these techniques are used to observe the result.

1. Support Vector Machine: It is a supervised learning technique for classification. As number of dimensions or size of the feature vector is significant in the problem of the sarcasm classification proposed algorithm has been used as it is effective in high dimensional spaces. It is additionally, memory efficient as it uses a subset of training points in the decision function.

2. Logistic Regression: It is a linear model for classification. As the proposed model contains two classes, binary logistic model is used which estimates the probability of a binary response based on one or more features.

3. Naive Bayes: Every pair of the feature is assumed to be independent in this technique. This is the primary reason for lower results than that of other two classification techniques as shown in section 6.1.6

6.1.5 Algorithm: Sarcasm Detection using Machine learning Techniques

The dataset contains a file for each review, which contains the review text and stars given by the user to the product. For each review, a review object is created which has properties like star rating, text, polarity and array of sentences. Each sentence has attributes like sentiment and array of words. Each word has attributes like polarity and POS tag. Bag of words dictionary is formed which store all the distinct words present in the reviews. Moreover, a bag of the bi-grams dictionary is formed which stores all the distinct bi-grams present in the reviews. All the features mentioned in section 6.1.3 are extracted, and their return value is appended to the array for each review. The classifier takes the feature vector and true label of respective reviews to build a model. This model is then used to classify other reviews. Algorithm 6.1 discusses the details of the approach.

Algorithm 6.1: Algorithm for identifying Sarcasm using Machine Learning Technique

```
input  : training array $T$, test array $Test$
output: Array of 0's and 1's where 0 means non-sarcastic and 1
        means sarcastic
1  Initialization label=[],feature=[], true_label=[], test_feature=[],
   feature_name=[list of features mentioned in section 6.1.3];
2  for $i = 0$ to $T.length$ do
3      label.append(T[i].label);
4      feature_list=[];
5      for $j = 0$ to feature_name.length do
6          feature_list.append(extract(T[i],feature_name[j]));
7      feature.append(feature_list);
8  for $i = 0$ to $Test.length$ do
9      true_label.append(T[i].label);
10     feature_list=[];
11     for $j = 0$ to feature_name.length do
12         feature_list.append(extract(T[i],feature_name[j]));
13     test_feature.append(feature_list);
14 clf=Classfier();
15 clf.fit(feature,label);
16 prediction=clf.predict(test_feature);
17 return prediction;
```

6.1.6 Experiments & Results

For training *375* sarcastic and *375* non sarcastic reviews are used. They are randomly picked from a corpus of *817* non sarcastic reviews and *441* sarcastic reviews. To remove the bias, an equal number of sarcastic and non-sarcastic sentences are taken for training. Results are obtained by testing on *66* sarcastic and *66* non-sarcastic reviews. F-measure achieved using different classifiers on different features has been shown in Figure 6.1. It can be observed that highest F-measure of *0.8* is given by bag of words feature using logistic regression. The reason for this could be the fact that corpus is specially built for sarcasm detection and contains words that are oftenly used for sarcasm. Figure 6.2 shows the F-measure achieved using different classifiers on various combination of features. Highest F-measure of *0.84* is given by all the features together excluding the star rate and pattern feature by logistic regression. The results given by naive bayes is lower compared to logistic regression and linear SVM because naive bayes has a higher bias but lower variance. In case of bag of words or combination of features, the feature set is large and sparse. Naive bayes technique counts the features that are correlated with each other because it assumes that each $p(x|y)$ event is independent when they are not. Logistic regression does a better job by naturally splitting the difference among these correlated features.

Table 6.1: Result comparison with baseline [Buschmeier *et al.* 2014]

	F-measure	F-measure (Baseline)
Interjections	0.62	0.01
Emoticons	0.47	0
NegNGP	0.33	0.01
Ellipsis+Punc	0.33	0.01
POSSQuotes	0.5	0.02
NegHyper	0.35	0
Imbalance	0.57	0.05
StarRates	0.39	0.72
BOW	0.8	0.69

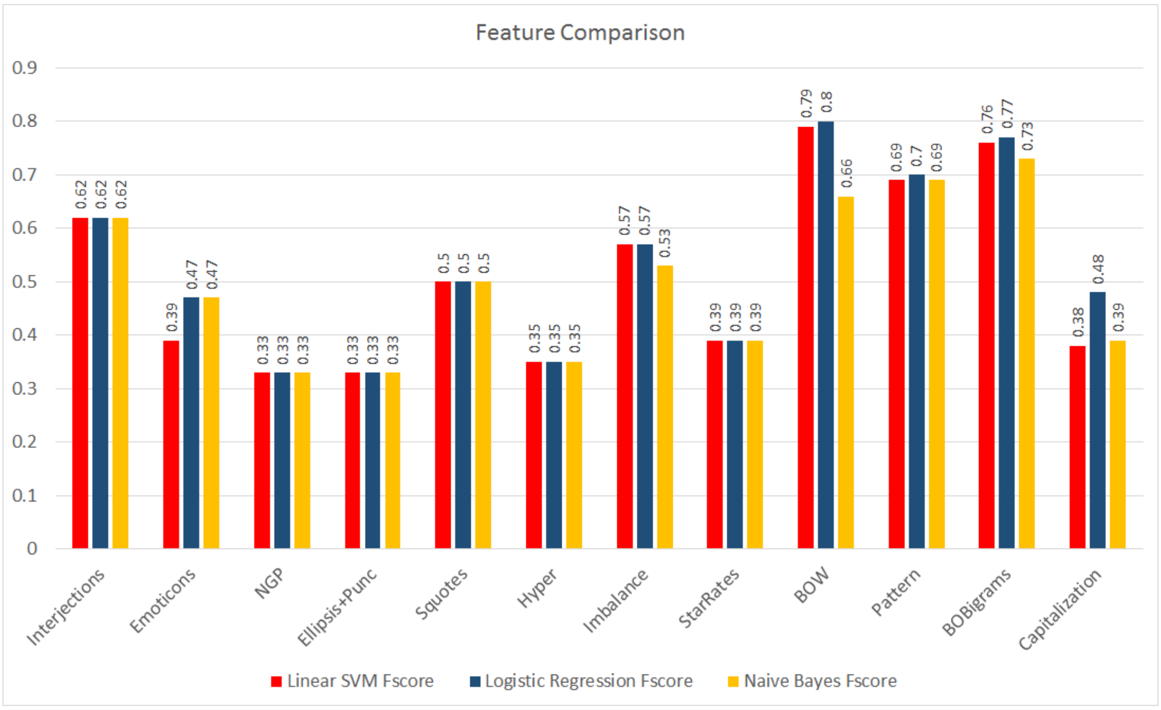


Figure 6.1: Feature comparison of different classification Techniques

6.1.6.1 Performance Analysis

Apart from *bag of words* feature other features like *bag of bi-grams* and *patterns* also gives F-measure of *0.77* and *0.7* respectively which is comparable to the best F-measure achieved. These results show that there is a specific vocabulary associated with sarcastic sentences and a pattern exists which could help in identifying sarcasm. F-measure of *0.62* is achieved by interjection feature which indicates that customers or users use interjections to express sarcasm. Star imbalance gave the F-measure of *0.57* which indicates that other information about a product like star rating can be useful in identifying sarcastic sentences. Other features resulted F-measure value less than *0.5*. F-measure is also calculated for individual features. Features which gave a high F-measure were chosen and then used a combination of them to see if the performance of classifier increases. On combining the bag of words, bag of bi-grams and interjections, F-measure increased to *0.82*. When a combination of all the features is used, F-measure achieved is *0.7*. This showed that some features need to be removed, to get better performance. On removing pattern based feature and star rating feature, highest F-measure of *0.84* is achieved. Table 6.1 shows the comparison between results achieved by proposed approach and [Buschmeier *et al.* 2014].

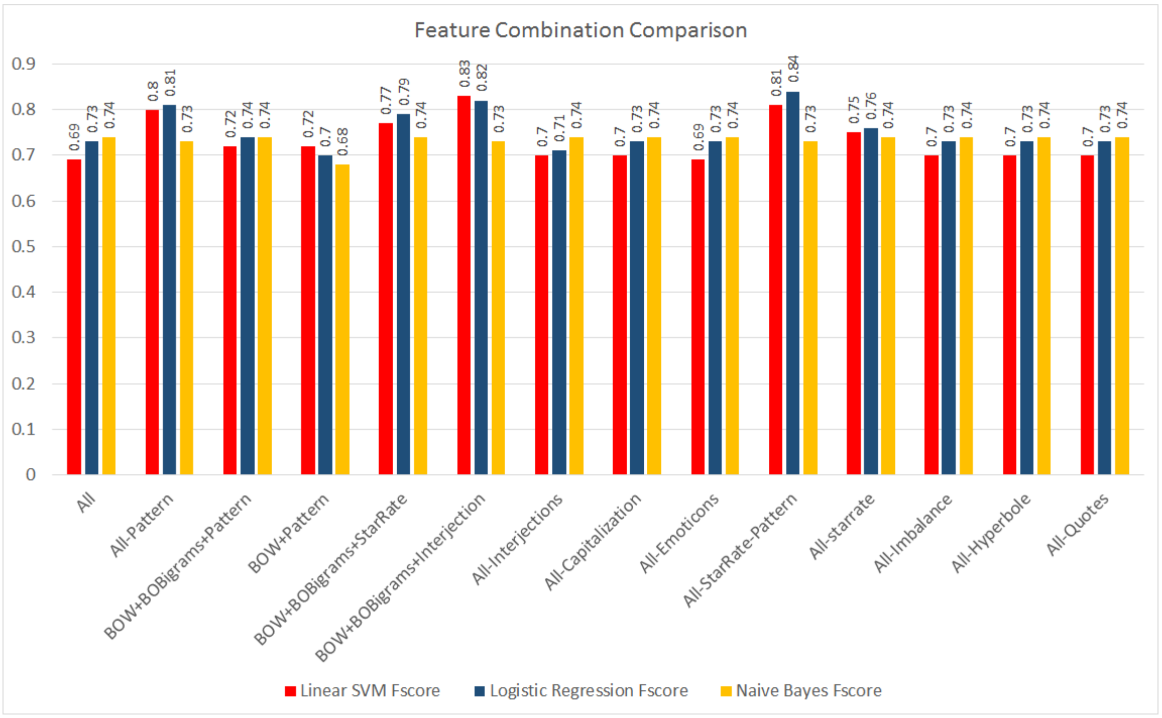


Figure 6.2: Comparison of different feature combination using various classifiers

The work in [Buschmeier *et al.* 2014] achieved the best F-measure of 0.74 on a combination of all features including star rating. However, their combination of all features did not include the features such as patterns, bag of bi-grams, capitalization and interjections which are included in proposed approach.

6.2 Sarcasm Detection for Monolingual & Code Mix Text

Substantial research exists for detection of sarcasm in languages, particularly English, while the native languages such as Indian languages have been neglected. This ignorance directly arises from the lack of annotated corpus in the respective language or from the fact that it is not being used particularly for digital communication. However, with the advent of social media, these native languages are being used more often in a mixed fashion with English resulting in a code-mix genre of language. The approach proposed in this section primarily aims to identify sarcasm in non-monolingual, code-mix language which comprises of more than one language. To facilitate this, a code-mixed, bilingual corpus of English-Hindi (En-Hi) languages has been created as discussed in section 6.2.1. Additionally, techniques for sarcasm detection in monolingual languages are also explored.

Existing research on sarcasm detection have primarily focused on machine learning techniques with feature engineering performed using *n*-grams based approach [Tsur *et al.* 2010], unsupervised pattern mining approach [Maynard & Greenwood 2014], uni-grams and emoticons based approach [Carvalho *et al.* 2009] amongst others.

Instead, proposed approach uses a deep learning technique that learns sarcasm features automatically from a corpus using a deep neural network. This methodology uses unsupervised training of the corpus to propel the supervised classification task for sarcasm detection. This classification problem is binary with the text being categorized as sarcastic or non-sarcastic. The main characterstics of the proposed work are as follows:

1. The proposed work detects sarcasm in code-mix genre of language, which is evolving as the primary language being used by masses for day-to-day communication.

2. A variety of neural net architecture for the classification model are explored to determine the best performing model for the task of sarcasm detection.

3. The use of pre-trained models for feature extraction is another attribute of this work. In computer vision, for classification task, images are used as direct input to the network (in the form of pixels). Such a procedure cannot work in the context of natural language processing. Hence, pre-trained models are used in the proposed work.

6.2.1 Dataset Description

Code Mixing is a very common phenomenon in social media text. However, to detect such data in social media has been a current field of research for quite some time now. Researches have worked on the generation of such text to aid the training of automated tools to analyze code mix text [Gupta *et al.* 2012] [Bhatia & Ritchie 2016] [Prabhu *et al.* 2016] [Banerjee *et al.* 2016c]. [Bali *et al.* 2014] worked on Hindi English language pair and had an interesting observation where they found out, most of the code mixing is occuring for the words tagged as nouns and verbs. However, there were certain exceptions to that. Similarly, few other works in literature such as work by [Gupta *et al.* 2016b], [Choudhury *et al.* 2007] and [Gupta *et al.* 2012] suggest the same. Hence, data for code mix sarcasm detection was synthesized considering the exceptions, and appropriate nouns and verbs were translated. Moreover, the synthetic data created was checked by human annotators to verify if the sentence seemed appropriate or not. In case of disagreement, the sentence was removed. This was important because there was no such data that existed beforehand in bulk for the training purpose of deep net models. Since deep learning approach for classification learns input features on its own, it demands a relatively large dataset. Hence, for the creation of the code-mixed dataset, most of the English(En) based annotated corpus are used. The corpora used are as follows:

- ironicQuotes.source[1]: A dataset collected by Antonio Reyes, it consists of 1,002 sarcastic quotes in English.
- Sarcasm Corpus v1 [Oraby *et al.* 2016]: It is a subset of the Internet Argument Corpus (IAC), which is a corpus for research in the political debate on internet forums, including response text from quote-response pairs annotated for sarcasm. The dataset consists of 1,993 opinions.
- Amazon Sarcasm Corpus: The dataset consists of *1,254* Amazon reviews, of which *437* are ironic, and *817* are non-ironic. It has been collected by [Filatova 2012] using the crowd-sourcing platform Amazon Mechanical Turk.
- Reddit[2]: The dataset is a collection of television sub-reddit comments by users accumulating to *2,688* comments.
- Sarcasm Corpus V2 [Oraby *et al.* 2016]: Sarcasm corpus V2 is a subset of the IAC, including response text from quote-response pairs annotated for sarcasm. The quote-response pairs are combined to form dataset sample as their combination captures the sarcastic context.

In total, there are *11,609* samples for training out of which *4,851* are sarcastic and *6,758* are non-sarcastic samples, corresponding to the vocabulary size of *51,961*. Figure 6.3 describes the distribution of different data used for evaluation.
The complexity in classification can be judged by the data visualization shown in Figure 6.4. As seen, the data points are quite overlapping which points to the degree of difficulty

[1]http://users.dsic.upv.es/grupos/nle/resources/ironicQuotes.source
[2]http://www.parrotanalytics.com/wp-content/uploads/2015/11/Sarcasm-Detection-in-Reddit-Comments.pdf

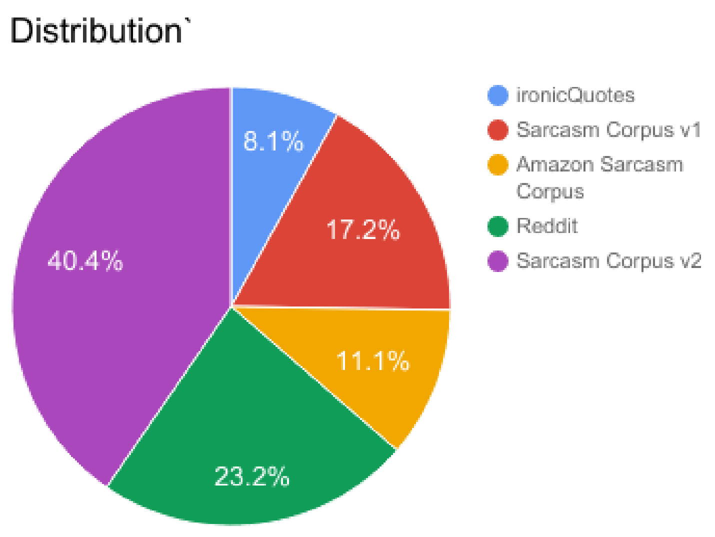


Figure 6.3: Data Distribution

in classifying the data points. The vector for a given data point is calculated as the cumulative sum of embeddings of its constituent words.

6.2.2 Preprocessing

Preprocessing step revolves around word-level tokenization to conserve only those words in a given sample which contributes to sarcasm detection. A custom tokenizer removes the URL(s), HTML tags while preserving the emoticons and interjection cues. The code-mixed English-Hindi (En-Hi) and Hindi (Hi) datasets are created via translation, benefited with automated word correction but suffered from a lot of other random disturbances like the inclusion of non-translatable words, which are manually handled. The tokenizer also features an English language specific contractions replacer which expands a given contraction (e.g. isn't $\rightarrow$ is not), for better word-level tokenization.

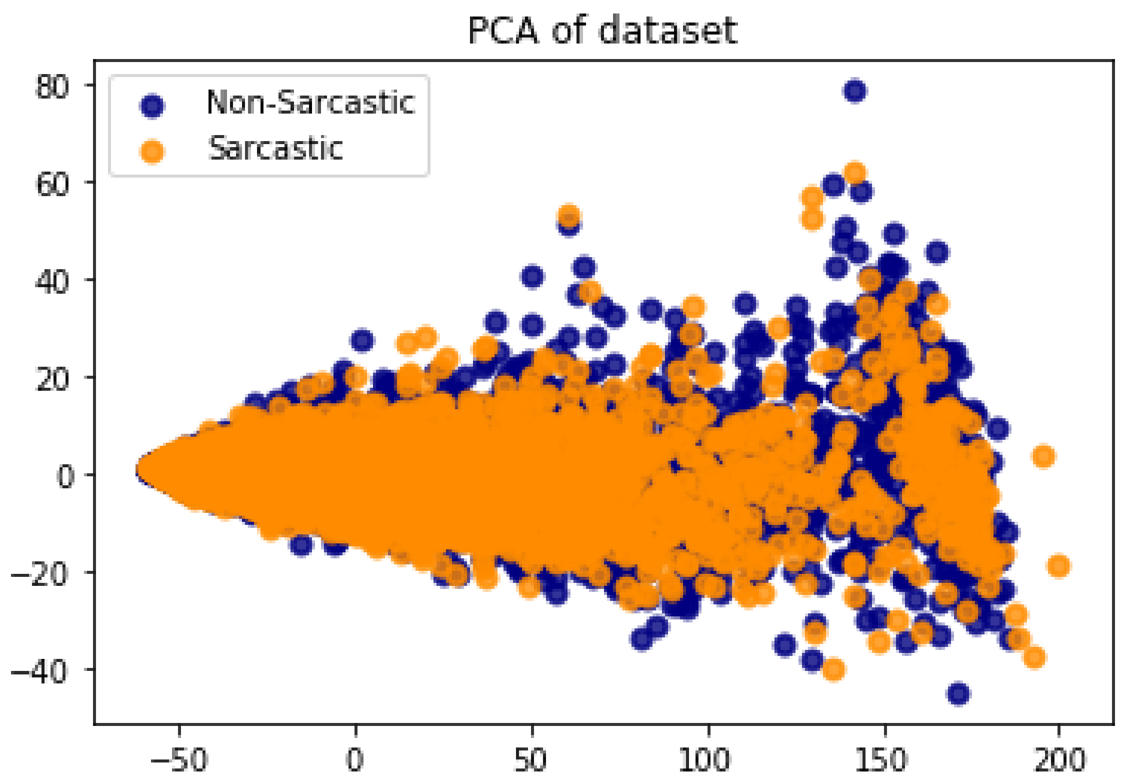


Figure 6.4: Data Visualization

6.2.3 Approach for Sarcasm detection in Monolingual & Code mixed environment

Most of the previous works have focused on sarcasm detection in corpora (majorly tweets) which are monolingual (primarily English). However, with the prevalent code-mix text on social media it has become essential to handle the code mixing. The proposed work targets bilingual code mix text. The framework for detecting sarcasm in both of these variants (Monolingual and Bilingual) have been proposed as follows:

6.2.3.1 Monolingual Text

Significant works have used a machine learning, feature-engineering based approach for the task in the monolingual text. Existing works based on deep learning technique [Amir *et al.* 2016] used twitter-based corpus and model the task based on CNN(s). Broad methodology details of the proposed approach are as follows:

Word-Embedding Representation:

Two variants of monolingual datasets are used: original English dataset and sentence-level translated Hindi dataset. For both the datasets, word2vec [Mikolov *et al.* 2013a] based vectors are used. These vectors are trained on Wikipedia dumps of respective language and have a dimensionality of *300*. Learnable representations of these embeddings are used during training. This is because Wikipedia uses very formal descriptive language which does not encompass the informal peculiarities brought in by social-media based dataset. From the learnable word embeddings, it is meant that the embeddings will be updated during training to include these subtle differences. Word embeddings are known to very well capture the syntactic constructs and semantic features of the language of the corpus. Each sentence of dataset is extended to a one dimensional vector of length n, where n is the maximum number of words amongst all sentences in the dataset. Dataset created has value of n equal to 300. Embeddings are used as a feature vector to the network.

Figure 6.5 depicts a portion of visualization of the created embeddings. It can be observed that the local cluster has verbs in majority, and verbs with the similar sentiment are closer. (For example: agree, disagree and mean, understand, realize clusters).

Neural Network Schema:

The sequential model is the combination of CNN, RNN and LSTM followed by a dense (Fully connected) layer with the sigmoidal output. The network can have any layer repeated while eliminating others to create a homogeneous network or include other layers to make the network heterogeneous. The activation (α) in the network is set to be the *ReLu* activation function due to its simplicity and efficient functionality to bring non-linearity in training. As the classification is binary, sigmoidal or, softmax function can be used to generate an output of the network. The sigmoidal function is used to map output between a range of *0* and *1*. Table 6.2 shows the training parameters of each layer used in this approach. The maximum depth of neural network reached upto three layers.

LSTM is known to capture long-term dependencies and is expected to capture better sarcastic context than RNN network which has a relatively low sightedness w.r.t. the length of the dataset. CNN network is known to be the best-performing networks [Poria *et al.* 2016a] when it comes to NLP tasks. With increasing depth of the network, it evolves over the features learned from previous layers to learn higher dimensional features of the

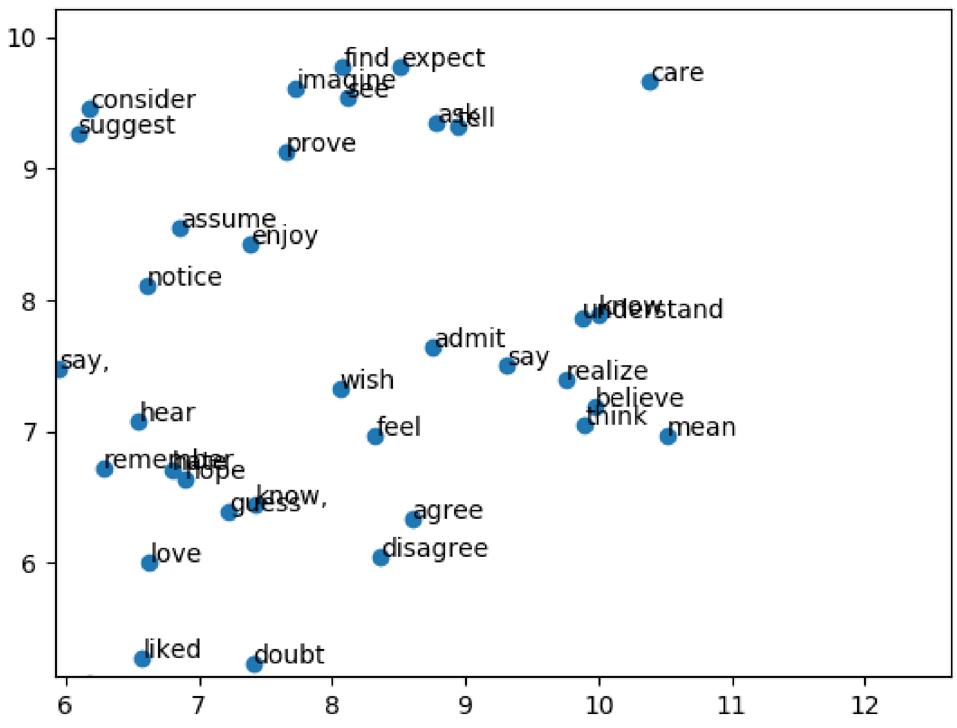


Figure 6.5: t-SNE Visualization of Word Embeddings

dataset. Thus, proposed approach uses CNN based models superseded by LSTM based model which is superseded by RNN based models. For LSTM and RNN models, the effect of increasing depth is decided under the evaluation section 6.2.4. The overall architecture of the monolingual model is described in Figure 6.6. Proposed work exploits the neural network component for performance enhancement, with little attention to the creation of embeddings.

6.2.3.2 Bilingual Text

6.2.3.2.1 Hybrid Model

Since, task deals with code-mix corpus (bilingual predominantly), directly using the above presented methodology for monolingual text is inappropriate and calls for a hybrid approach which encapsulates the grammatical construct of both the constituent monolingual languages in the corpus as well as their mixture. Thus, a unique approach is proposed

Table 6.2: Network Layer configurations. For a neural network layer *X (CNN/LSTM/RNN)* and number *i (1/2/3)*, the notation X_i means the layer *X* at i^{th} depth in the model

CNN Layer		Kernel size	Feature Map
	CNN 1	5	128
	CNN 2/3	5	64
Max-Pool Layer	Kernel Size		
	3/5		
Dense Layer	Nodes		
	1/128		
LSTM Layer		Units	
	LSTM 1	100	
	LSTM 2/3	50	
RNN Layer		Units	
	RNN 1	100	
	RNN 2/3	50	

namely, Hybrid Multi-Model Weighting (HMMW) which uses the combination of efficient monolingual deep learning model for code-mix text and its monolingual elements. The components of the model described in Figure 6.7 are as follows:

Word-Embedding Representation:

Since the corpus is mixed, use of pre-trained embeddings on the sizable monolingual corpus (like Google news) won't work. Instead, the corpus is pre-trained locally on Word2vec model with translated code-mix dataset as input. The context window for creating word embeddings is set to *10*. As for monolingual English (En) and Hindi (Hi) data sets, embeddings discussed earlier are used. Again the embeddings are non-static during training and learn to include the sarcastic nuances present in social media data.

Neural Network Schema:

The neural network used is a combination of CNN, RNN and LSTM neural layers, which came out to be the best performing network of variants described in the evaluation section 6.2.4.2. Hence, three different neural networks which adapted best to a given dataset are used in HMMW model.

Aggregator:

The output from each of the three models is combined according to the weight given to each of the model. The final output is calculated w.r.t. to following equation, where w_i

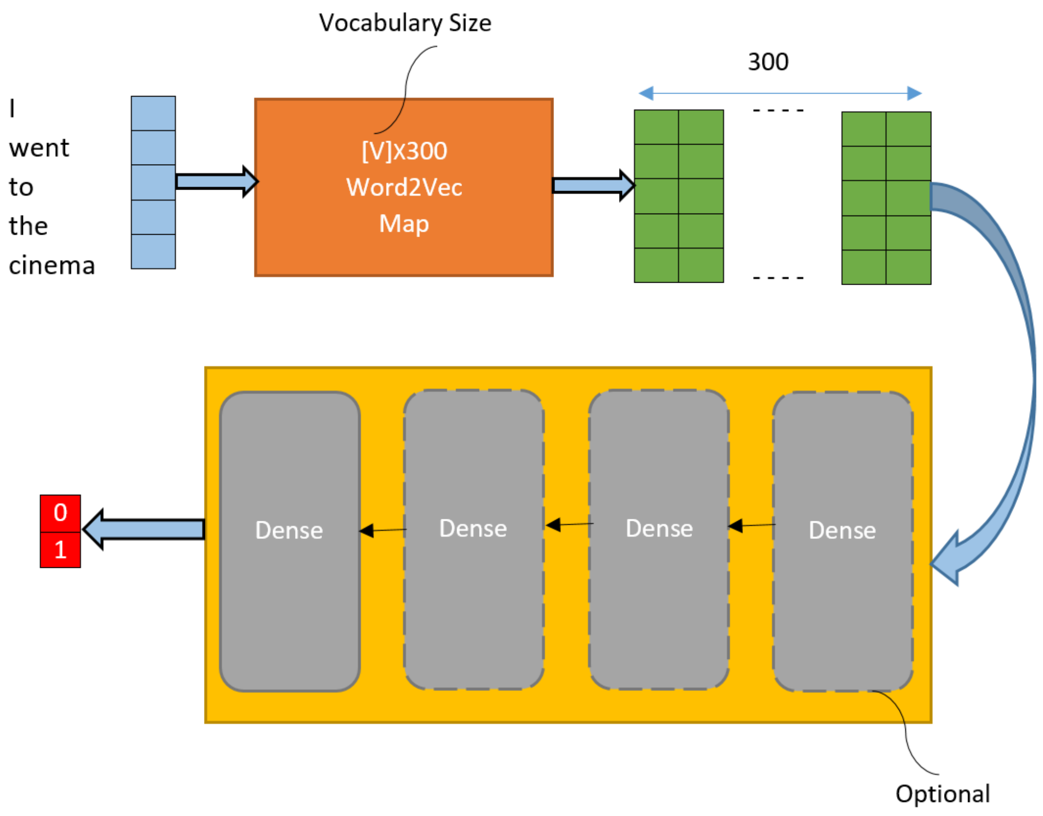


Figure 6.6: Neural Network Architecture

denotes the weight of the i$^{\text{th}}$ sub-model.

$$Y = round \frac{w_1 * y_1 + w_2 * y_2 + w_3 * y_3}{w_1 + w_2 + w_3} \tag{6.1}$$

6.2.3.2.2 Bilingual Embeddings

Another approach proposed uses cross-lingual word-embeddings, which are implicitly created during training in the task of machine translation. However, bilingual embeddings are created explicitly. Previous approaches like monolingual adaptation, bootstrapped the learning of target language (Hi), which is resource-scarce, from well-trained representations of the source language (En), which is resource-abundant. Bilingual mapping independently learns monolingual representations of each language and then learns a mapping between them. Bilingual mapping has been used in other applications as well, and one such work is by [Luong *et al.* 2015]. The main idea is to jointly learn word representations of both languages, a technique known as bilingual training rather than

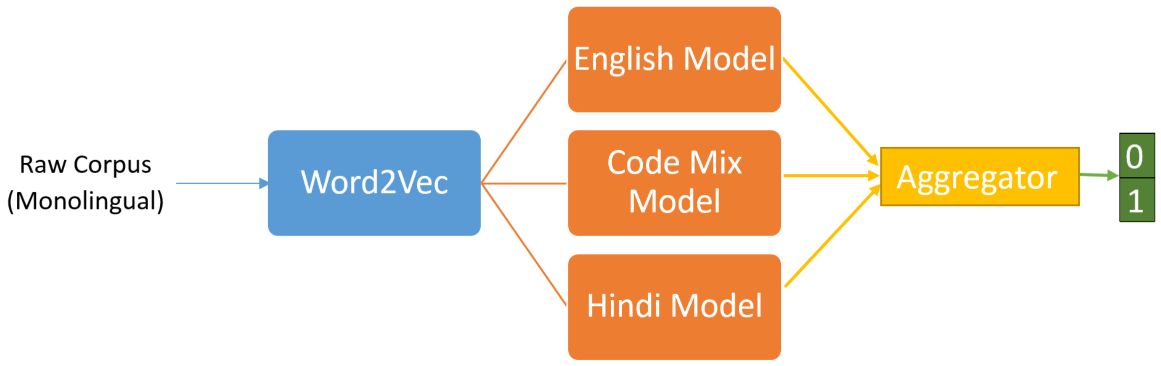


Figure 6.7: Hybrid Multi Model Weighting

fixing pre-trained representations on either source or target language side. The procedure described in Figure 6.8 is approximately the same as that of monolingual methodology, with the modification of cross-lingual embeddings being used at the first stage of the methodology.

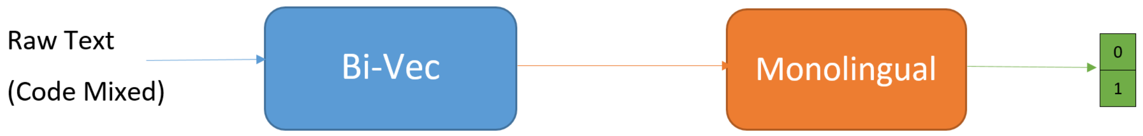


Figure 6.8: Model based on bilingual embeddings

6.2.4 Experiment(s) and Result(s)

The corpus of total samples *11,609* is evaluated under the schema of neural networks described above. These models are composed of basic layers of CNN, RNN and LSTM. The corpus is split in *80:20* ratio of training and testing of sizes *9,288* and *2,321* respectively. The training is done in a batch of *128* samples, for four epochs. Word embeddings are consistently used as input to the model. The models are evaluated on the test partition for accuracy, precision, recall and F-measure as metrics.

6.2.4.1 Monolingual Text

For the monolingual text classification task for sarcasm detection, evaluation is carried for both the original English and translated Hindi data sets.

English dataset

The overall results show that a pre-trained CNN with a single layer (1 x CNN) gives the best performance with an accuracy of *74.30%* and F-measure of *67.60%* as indicated by Table 6.3. Contradicting the hypothesis of learning evolving features with increasing depth, adding more layers of CNN to the network degrades its performance as shown in Figure 6.10. As for LSTM based model, it can be concluded from Figure 6.11, the results do not meet the proposed hypothesis, that long-term dependencies are very well captured, as it resulted in low accuracy and F-measure than CNN based models. However, LSTM based models show better adaptation to the task with an increase in the depth of the model. Moreover, the (2 x LSTM) model gives a high recall of *73.68%*. LSTM models perform better than RNN models as expected. RNN models show inconsistent results with increasing layers in the network as depicted in Figure 6.12. The performance of 1 X CNN model is better than other combinations as shown in Figure 6.13. The dependency of a word in monolingual text is lesser than that of multi-lingual because of less complex behavior of monolingual data. It tends to extract more important features in this case and much context is not needed in terms of longer dependencies. An inspection of the results obtained by combinational models in Figure 6.9 shows that including a CNN layer significantly improves the performance of the network.

Hindi Dataset

As the analysis of English dataset suggested, (1 x CNN) model is the best performing model, Hindi dataset is evaluated under this model for accuracy, precision, recall and F-measure as metrics. The results presented in Table 6.4 show the consistent and improved performance over its English contemporary. As shown in Figure 6.14, the accuracy achieved is *77.36%*, which exceeded the accuracy obtained over English dataset for same (1 x CNN) model by *3.06%*. This points out to the fact that embeddings for the Hindi language are capturing the context of sarcasm detection better than the English.

6.2.4.2 Bilingual Text

For the bilingual text classification task, model variants are trained over the code-mixed dataset. An analysis in Table 6.5 shows that the best performing model for monolingual text, (1 x CNN) model, is outperformed by the combinational model (CNN+LSTM+RNN),

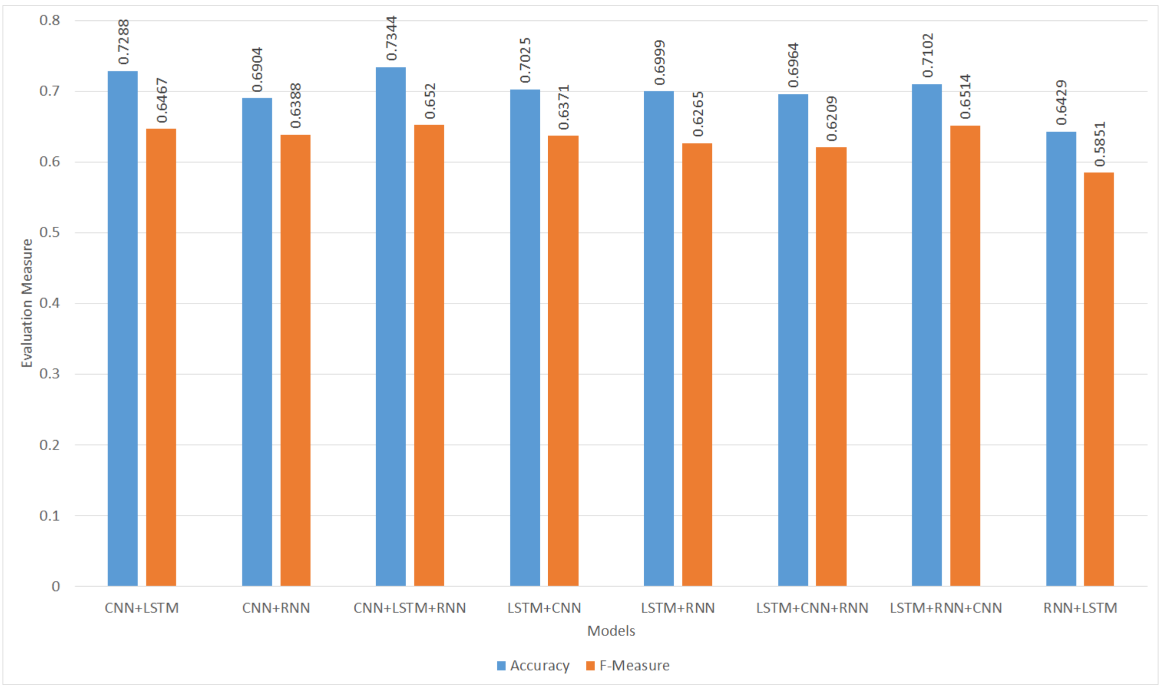


Figure 6.9: Monolingual Heterogeneous Models

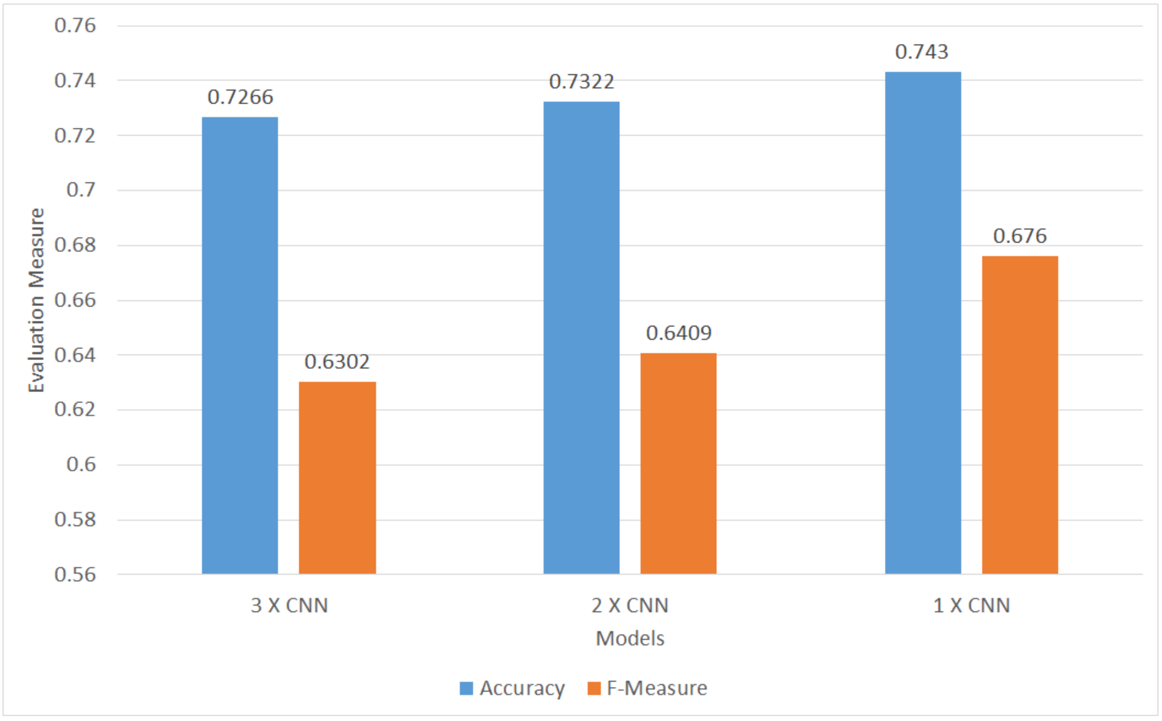


Figure 6.10: Monolingual CNN Models

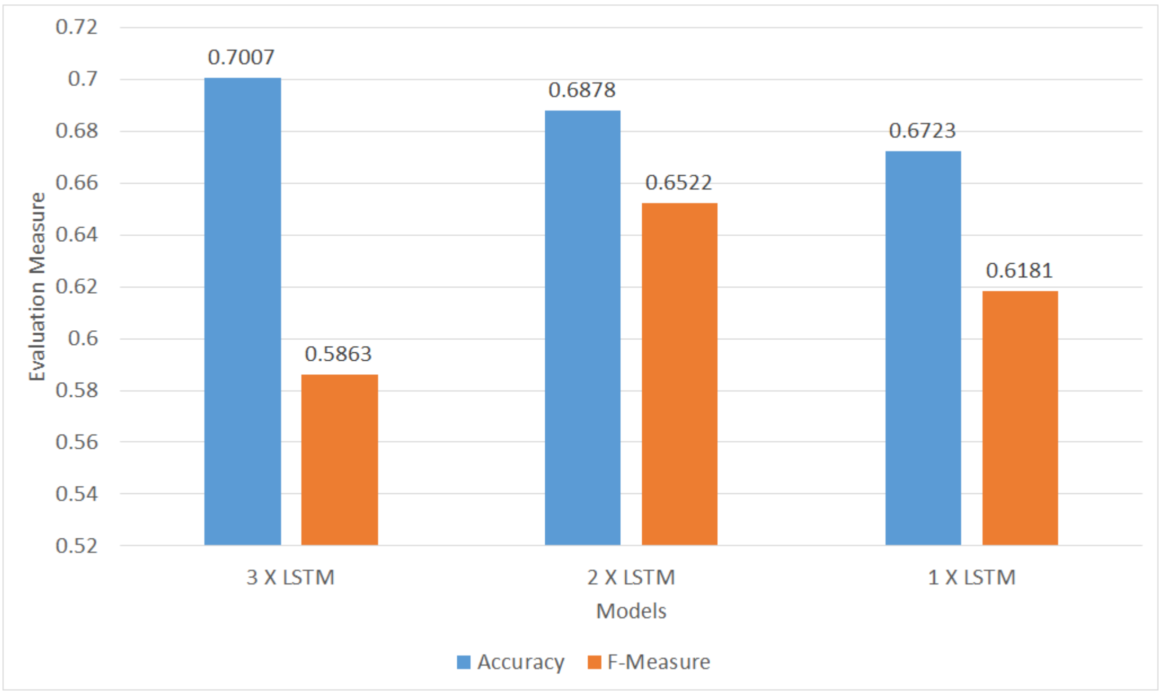


Figure 6.11: Monolingual LSTM Models

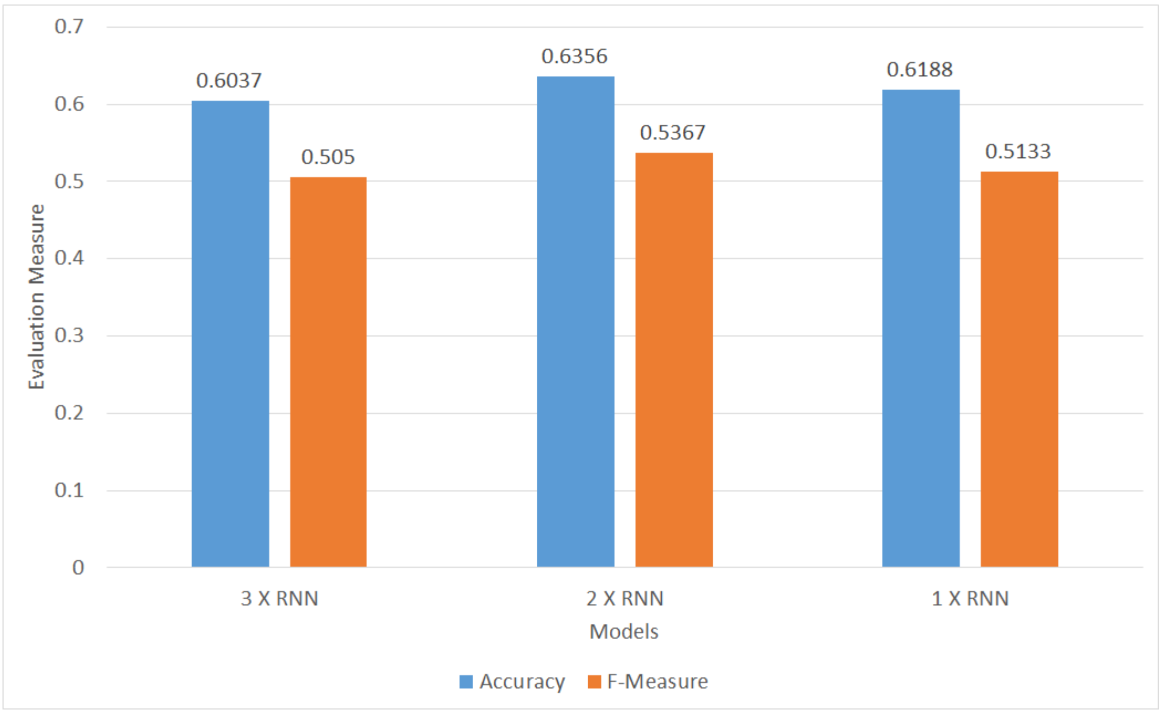


Figure 6.12: Monolingual RNN Models

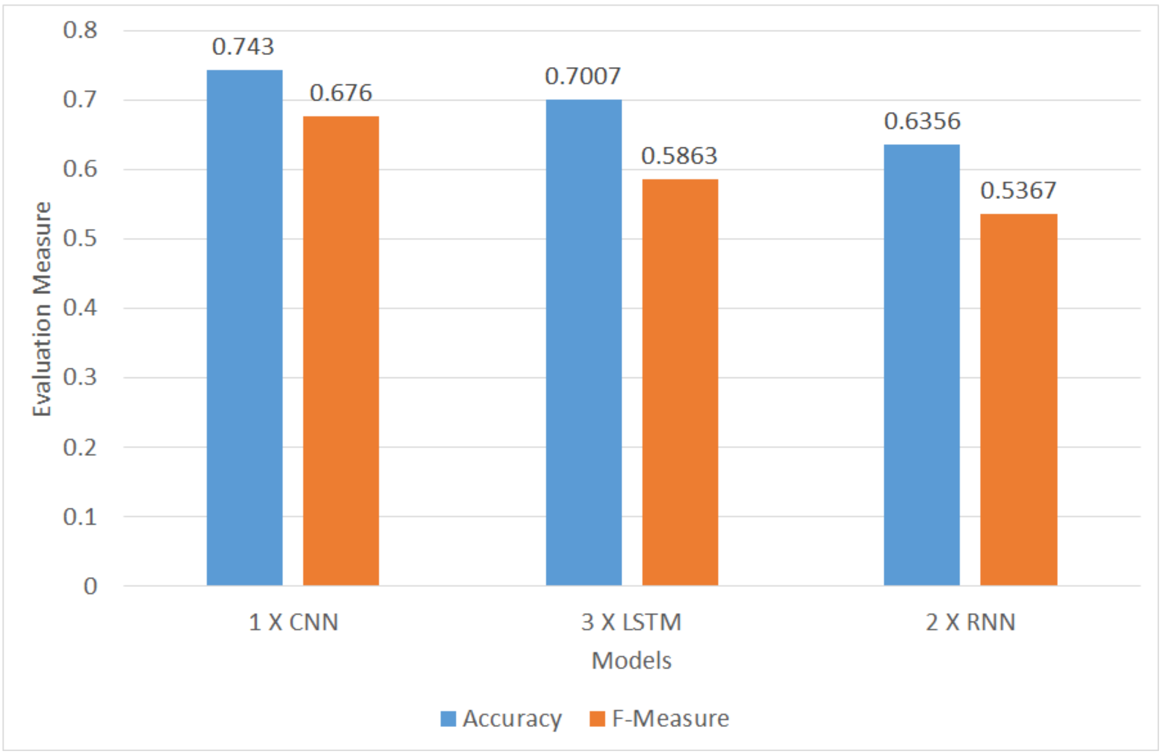


Figure 6.13: Comparison among best models of different architecture for Monolingual text

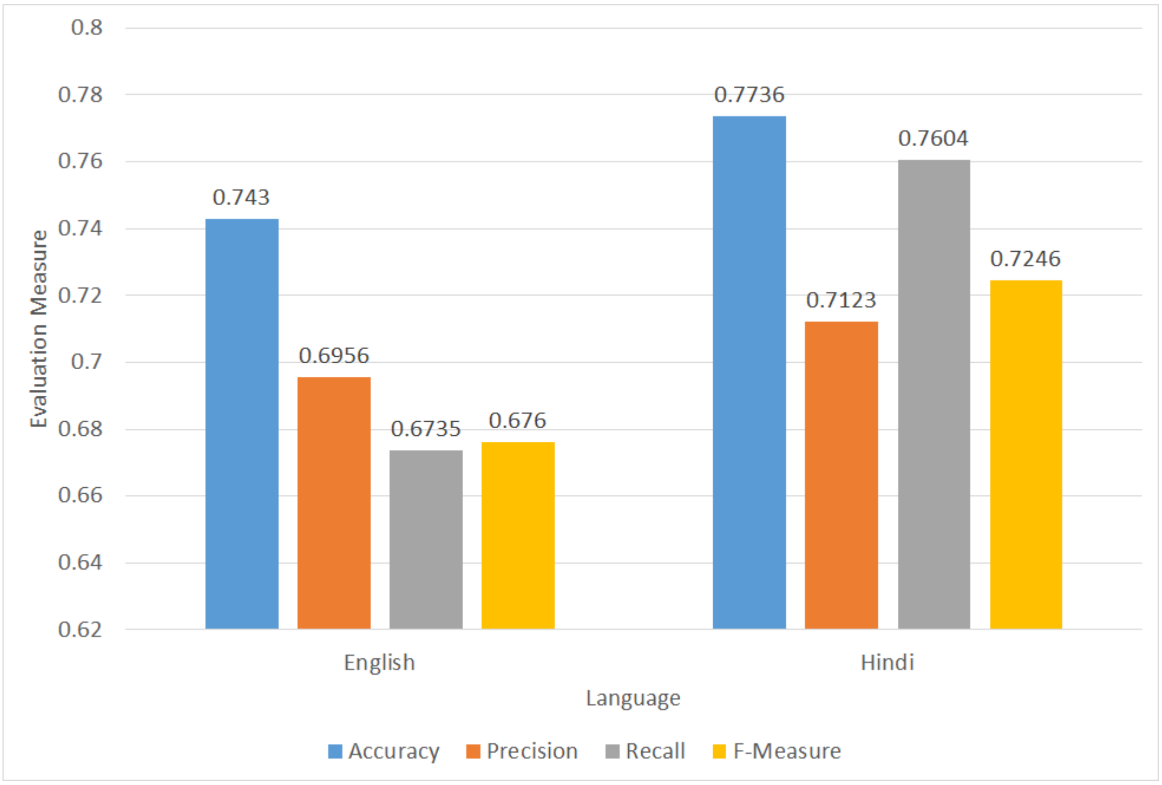


Figure 6.14: Comparison among 1 layered CNN models of Hindi and English

Table 6.3: Peformance of different models (*"i x layer"* model denotes *i* times repeated stack of the respective layer) for Monolingual (En) dataset

Model	Accuracy	Precision	Recall	F-measure
3 x CNN	72.66%	69.94%	59.36%	63.02%
2 x CNN	73.22%	71.41%	60.00%	64.09%
1 x CNN	74.30%	69.56%	67.35%	67.60%
3 x LSTM	70.07%	66.44%	54.17%	58.63%
2 x LSTM	68.78%	59.75%	73.68%	65.22%
1 x LSTM	67.23%	59.08%	67.23%	61.81%
3 x RNN	60.37%	51.73%	51.86%	50.50%
2 x RNN	63.56%	56.02%	53.49%	53.67%
1 x RNN	61.88%	53.03%	51.26%	51.33%
CNN + LSTM	72.88%	69.50%	62.31%	64.67%
CNN + RNN	69.04%	60.76%	60.76%	63.88%
CNN + LSTM + RNN	73.44%	69.97%	63.20%	65.20%
LSTM + CNN	70.25%	63.60%	66.23%	63.71%
LSTM + RNN	69.99%	63.13%	64.29%	62.65%
LSTM + CNN + RNN	69.64%	63.15%	63.30%	62.09%
LSTM + RNN + CNN	71.02%	64.03%	68.45%	65.14%
RNN + LSTM	64.29%	55.88%	63.35%	58.51%

Table 6.4: Performance measure on Monolingual (Hi) dataset

Accuracy	Precision	Recall	F-measure
77.36%	71.23%	76.04%	72.46%

with an accuracy difference of *1.25%*. This is because CNN extracts best of the features and LSTM help in remembering the long term dependencies so that context can be taken into account. In case of code mix when code switching occurs it becomes important to remember the context of data and hence LSTM helped in the same. Overall analysis shows that (CNN+LSTM) model performs best when precision is considered as evaluation metric and (1 x CNN) model with on-the-fly (only Learnable) training of word embeddings, performs best for both recall and F-measure metrics. This is justified by the fact that pre-trained embeddings update themselves while using training data whereas learnable embeddings entirely mend their parameters according to the train dataset giving a higher recall. However, for LSTM based models, increasing the depth of the architecture improves the overall performance of the model as shown in Figure 6.17. This is in contrast with RNN based models which shows a decrease in performance with increasing depth, as depicted in Figure 6.18. Moreover, Figure 6.16 models just based on CNN show the same trend as the RNN based models. Analysis of combinational models suggests that a CNN layer is the main ingredient to improve overall performance for the task of sarcasm detection, as shown in Figure 6.15. Further comparison between the best local performers over all metrics for the three pure classes of CNN, LSTM and RNN which are 1 x CNN, 3 x LSTM and 1 x RNN respectively shows an almost linear decrease in performance over all metrics in decreasing order stated in Figure 6.19.

Table 6.5: Bilingual (Hi-En) results. A *"i x layer"* model denotes *i* times repeated stack of the respective layer. Static and learnable represents that only static embeddings and on-the-fly embeddings are used respectively

Model	Accuracy	Precision	Recall	F-measure
3 x CNN	69.34%	62.18%	64.29%	62.26%
3 x CNN (Learnable Embeddings)	69.17%	61.32%	67.75%	63.24%
3 x CNN (Static Embeddings)	62.27%	55.19%	39.73%	45.19%
2 x CNN	70.29%	69.08%	49.19%	56.48%
2 x CNN (Learnable Embeddings)	68.56%	60.15%	66.14%	62.16%
1 x CNN	71.28%	65.18%	63.48%	63.48%
1 x CNN (Learnable Embeddings)	70.16%	62.03%	70.89%	65.10%
3 x LSTM	69.69%	65.57%	55.16%	58.88%
2 x LSTM	69.21%	66.57%	48.23%	54.91%
1 x LSTM	68.18%	61.22%	61.35%	60.17%
3 x RNN	58.47%	48.79%	48.99%	48.03%
2 x RNN	59.59%	50.40%	49.57%	49.12%
1 x RNN	66.93%	61.71%	47.78%	52.60%
CNN + LSTM	70.16%	69.13%	49.85%	56.59%
CNN + RNN	70.63%	65.07%	59.87%	61.30%
CNN + LSTM + RNN	71.67%	65.44%	65.04%	64.40%
LSTM + CNN	70.16%	63.32%	63.52%	62.62%
LSTM + RNN	67.87%	59.83%	61.62%	59.79%
LSTM + CNN + RNN	69.25%	64.70%	52.27%	56.97%
LSTM + RNN + CNN	67.40%	60.24%	57.50%	57.86%
RNN + LSTM	61.54%	52.87%	47.03%	48.74%

HMMW Model:

With the code-mix (bilingual) model being (CNN+LSTM+RNN) network and the English model, Hindi model being (1 x CNN) model, the HMMW parallel schema achieved an accuracy of 76.84% on the test partition of the dataset. Each model is weighted equally.

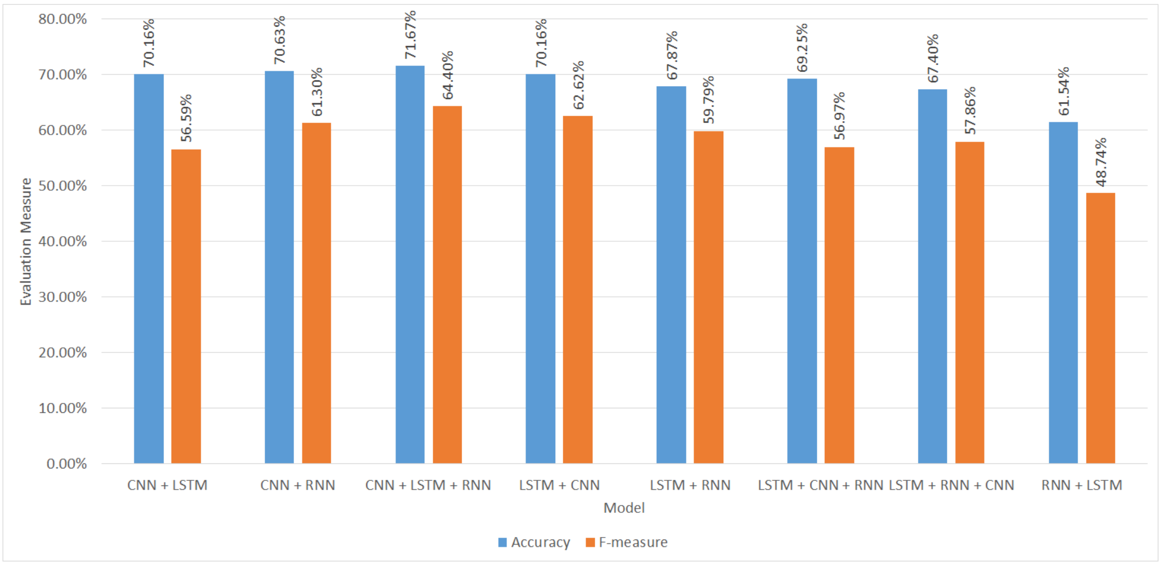


Figure 6.15: Bilingual Heterogeneous Model

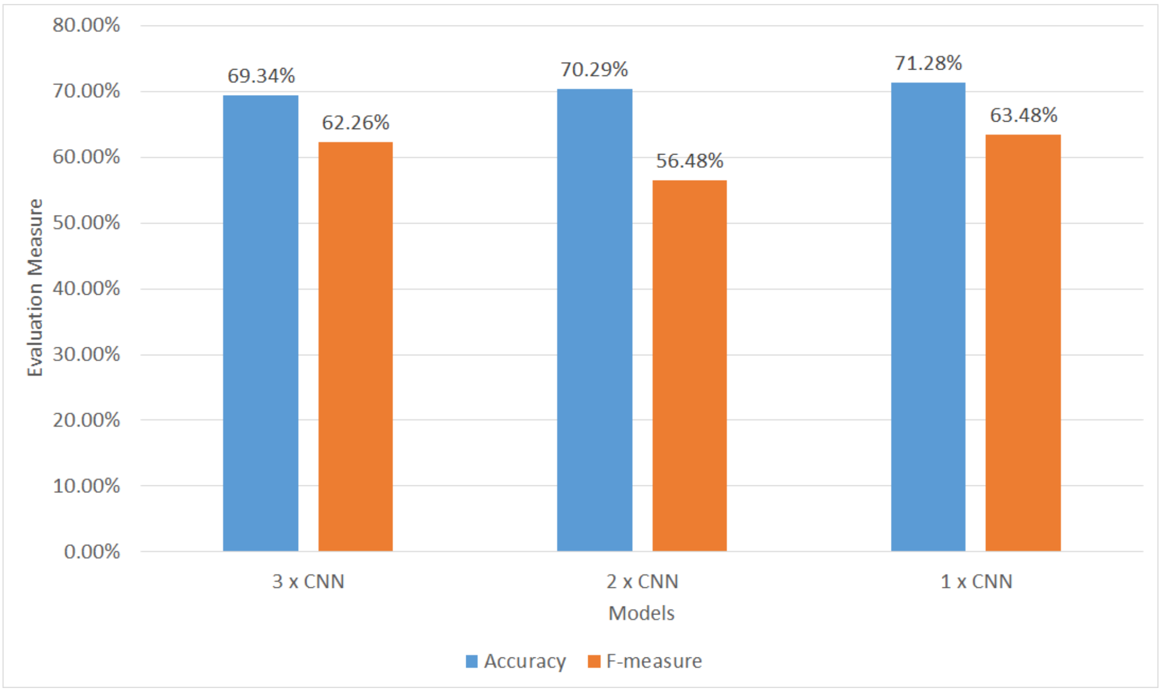


Figure 6.16: Bilingual CNN Model

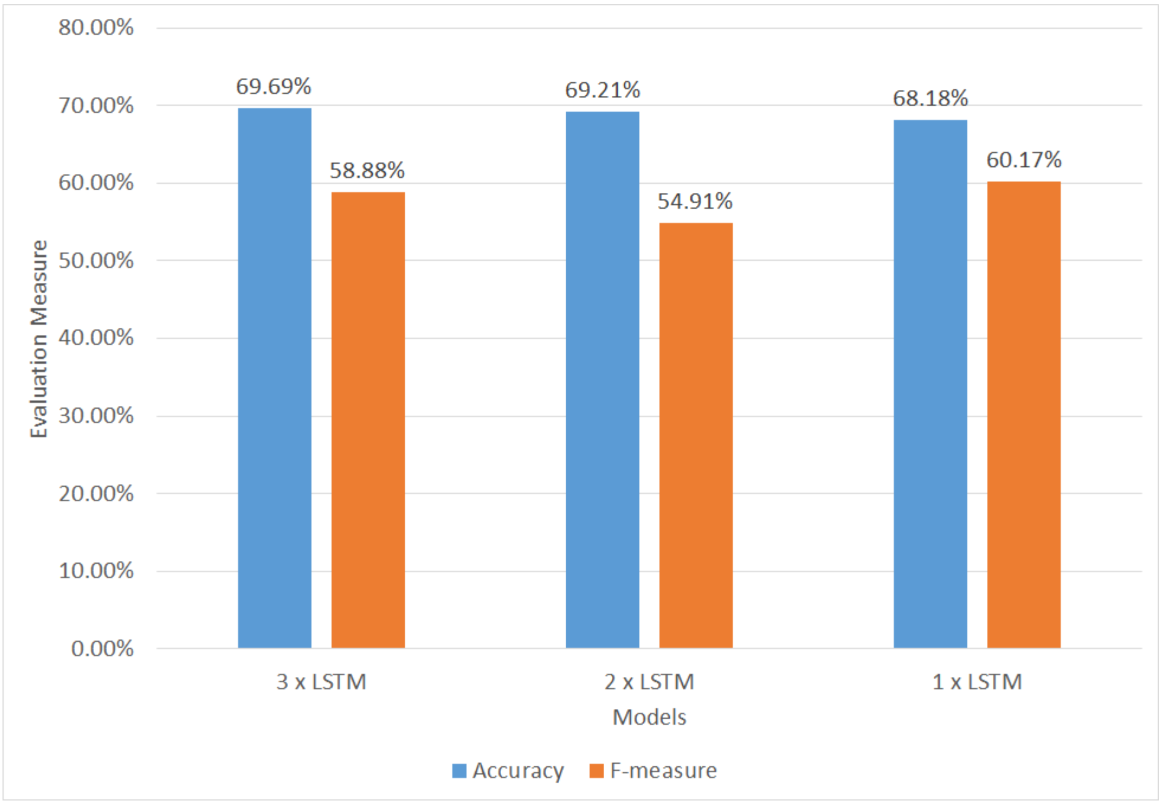


Figure 6.17: Bilingual LSTM Models

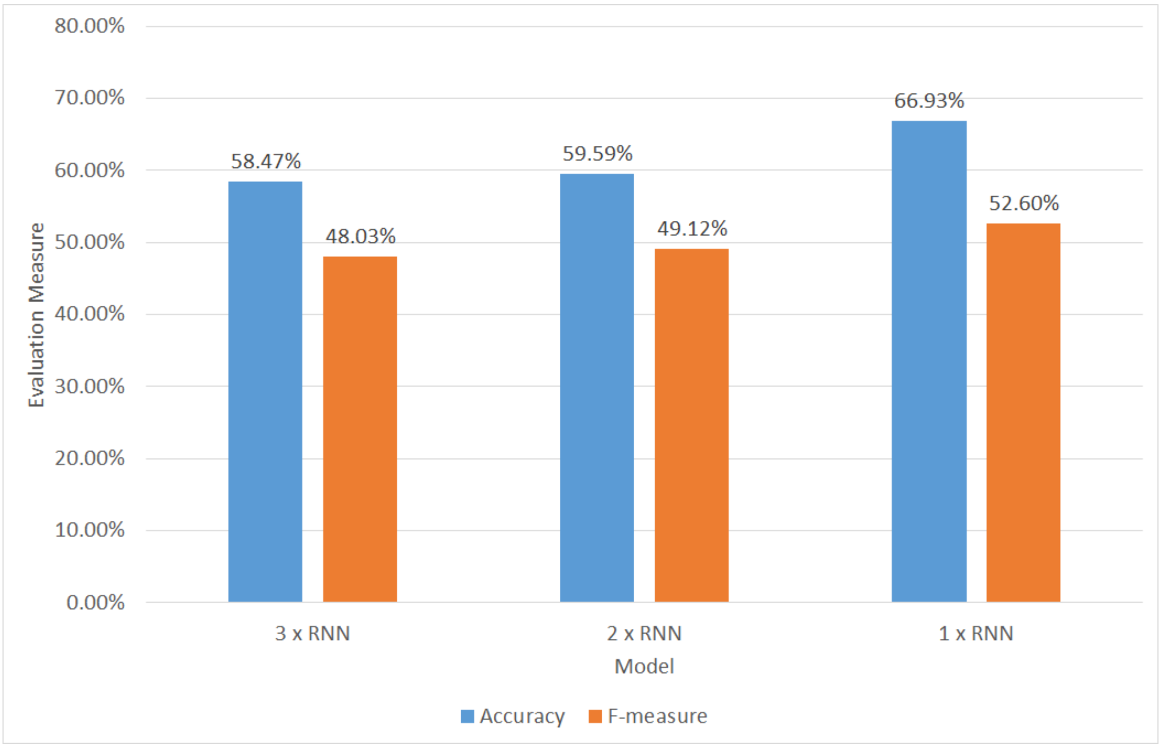


Figure 6.18: Bilingual RNN models

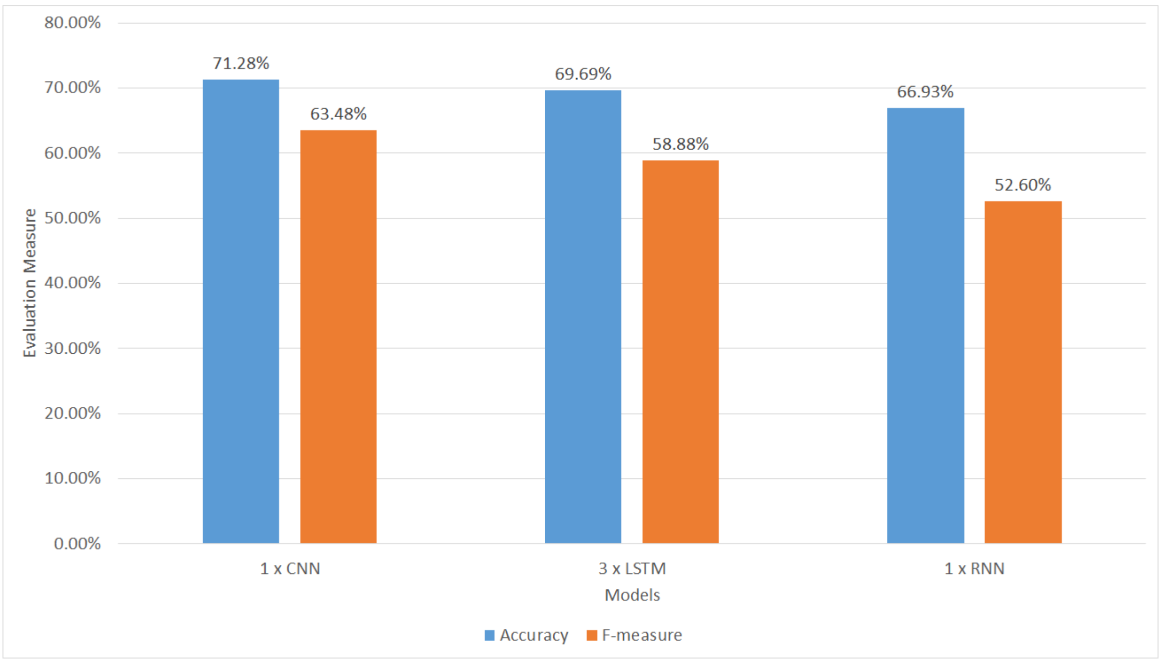


Figure 6.19: Comparison among best models of different architecture for Bilingual text

Bilingual Embeddings:

To study the effect of using bilingual embeddings in place of regular monolingual embeddings, (1 x CNN) model is trained over the code-mixed dataset. Two sets of embeddings with *100* and *300* dimensions are analyzed upon. As predicted, using embeddings with higher dimension gives relatively better results as can be seen in Table 6.6. Another inspection shows that extending the period of training jointly optimizes the model for all evaluation metrics in context shown by the evaluation under *8* epochs of training. Clearly

Table 6.6: Bilingual Embeddings result

Parameters	Accuracy	Precision	Recall	F-measure
100 dims, 4 epochs	68.00%	63.28%	52.37%	56.31%
300 dims, 4 epochs	69.08%	65.77%	52.59%	57.16%
300 dims, 8 epochs	68.26%	61.68%	63.57%	61.33%

in Figure 6.20, bilingual embeddings improve on all the metrics with increased training and are less optimum than monolingual embeddings. As for Figure 6.21, static embeddings represent the pre-trained embeddings which are not trained during model training. This is the worst performing word embeddings since they do not adapt to the sarcasm-specific dataset. Moreover, learnable embeddings, which are only trained simultaneously

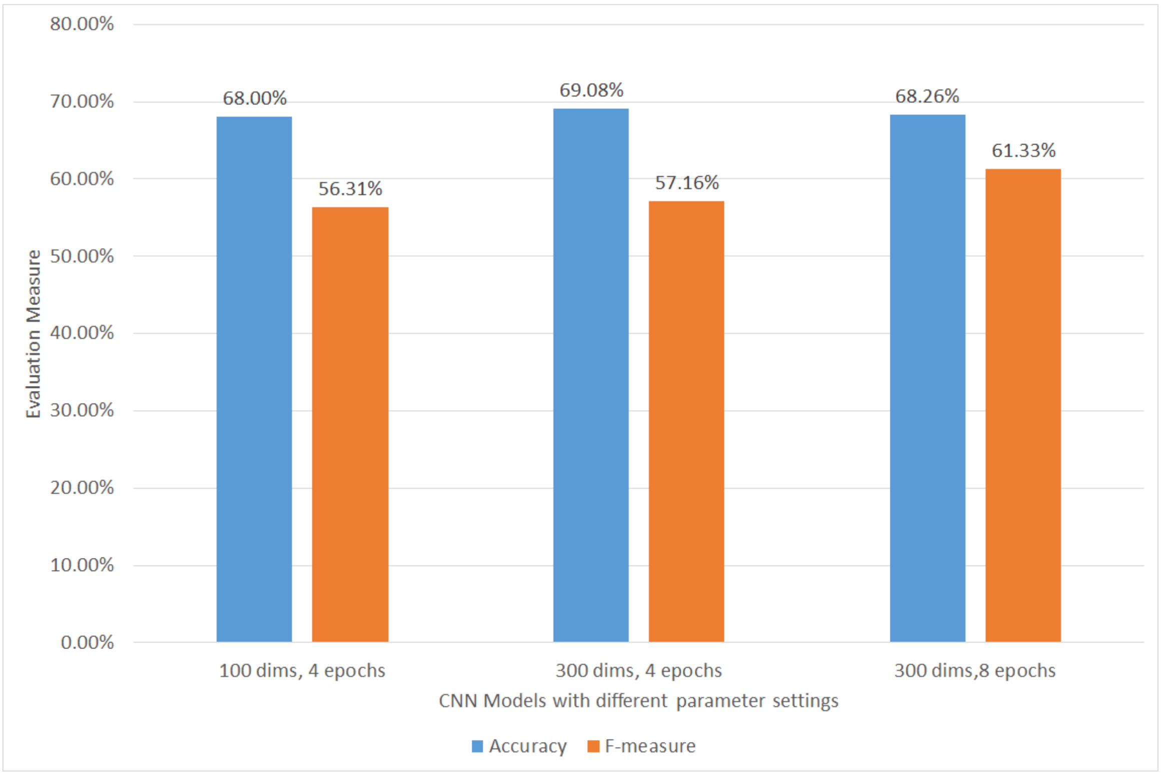


Figure 6.20: Bilingual CNN model with different settings

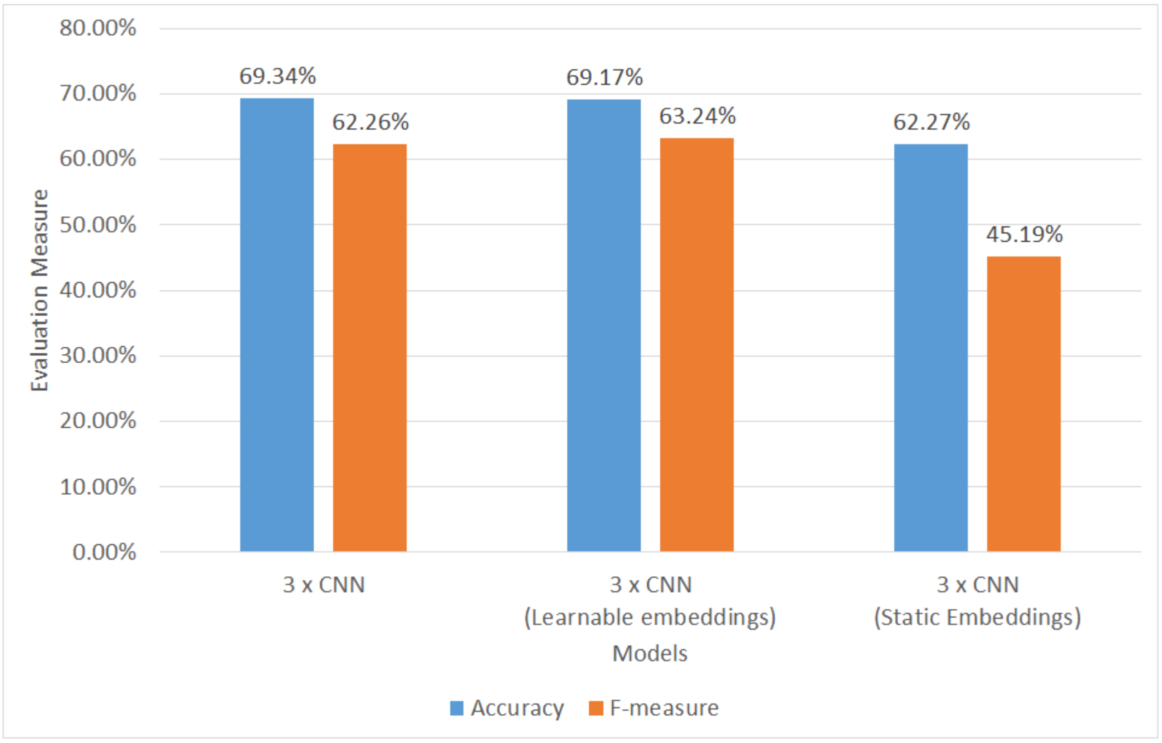


Figure 6.21: Bilingual CNN Variants Models

with model training, perform better due to their adaptability to sarcasm-specific dataset. The best performing embeddings are those which are pre-trained as well as post-trained, along with the model. They are the most generic word representations in terms of their adaptability to sarcasm as well as plain text data (For Example, Google News, Wikipedia etc.). Thus, overall comparison between the three approaches reveal that HMMW approach gave the best result with accuracy as evaluation parameter in consideration.

6.3 Concluding Remarks

This chapter is primarily focused on the idea of sarcasm detection. Using existing features in state of the art, first approach is based on traditional machine learning techniques experiments to detect sarcasm detection in English. The second approach focuses on both code-mix dataset and monolingual (English) dataset. Further, it also inspects upon a variety of neural network models for best suitability for the task of sarcasm detection. Current work limits itself to the existence of two languages in the dataset, English and Hindi. Experiments are performed with simple word2vec based monolingual embeddings and expand to using bilingual embeddings. While the usage of former gave expected improvements, experimentation with the latter did not perform well. Coming to the adaptability of neural models for the task, CNN and LSTM layers improved performance of NLP system in general and hence, the model analysis shows the potential of LSTM and CNN based networks for this task. Experimentation with the weighted model HMMW showed significant improvements due to the generalization over all the specific trained models. The code-mix sarcasm dataset which is created from English-based dataset for the task of code-mix sarcasm detection, with the help of machine translation, lacks in encompassing the basic construct of real social media text. This is caused by direct machine translation forced by the scarcity of annotated code-mix social media dataset. Major improvement can be made by developing a systematic procedure for such a generation. While the real social media data can mix over a number of languages, the current system is based on code mixing of two languages only. This implies an expansion of the HMMW model developed so far. Development of deeper neural network based on CNN, LSTM layers is also quite promising for the task of sarcasm detection.

Chapter 7

Code Mixing Tools

The exponential growth in the popularity of social media has created a plethora of textual data. It has become a severe issue to recognize textual data efficiently for various reasons. With the significant boost in the technology along with easy availability and accessibility for gadgets, social media has become a profound platform for people to express their opinions. The population participating in social media comprises of different and diverse individuals. People who do not have English as their mother tongue tend to use their native language to express their thoughts. These words are often written using phonetic typing or Roman script and are frequently found fused between English words or phrases, which is widely known as code-mixing or code-switching. The English language still forms the basis for all the primary social media communication in India. Owing to language diversities and dialect variations, there is a requirement of developing technologies for processing other languages in social media text. The use of non-native script by the users can be attributed to bilingualism and multilingualism existing in the country. People are usually more familiar with the Roman script while using electronic gadgets such as smartphones, desktops and laptops. Because of smart phone usage and easy internet accessibility, it has been found that users are utilizing the Roman script to write their native language. This form of writing is evolving as a new type of communication at the social platforms, and there has been significant growth of such type of content. This kind of text poses a new challenge in the area of text analytics as there is a need to process such data automatically for various applications.

In formal communication, people are cautious about what they speak and how they speak but at informal platforms like Facebook, Twitter and WhatsApp, they tend to use mixed up words and slang of other languages. These words may be in their native script or transliterated into English. This trend of mixing words from two or more languages is called code mixing. Code mixing is embedding of linguistic units such as phrases, words, morphemes of one language into an utterance of another language whereas code-switching refers to the co-occurrence of speech extracts belonging to two different grammatical systems. Here, code-mixing is used to refer to both unless explicitly stated. With the introduction of smartphones, people from all over the world have started using social media platforms, and this has generated a lot of code mixed text.

India is a multilingual country with more than *1600* languages being spoken such as Hindi, Punjabi, Bengali, Telugu, Marathi, Tamil, Gujarati and many more. With the introduction of Indic keyboards and articles that use Indic languages, people started using Indic languages in their ordinary conversation, and this has made people converse easily on the internet. In a country like India, where almost everyone grows up learning at least two languages, code mixing is inevitable. Processing these code mixed scripts is a task of massive proportions since the underlying syntactic structure of the languages differs immensely. Although code mixing is not always considered a right way of communicating, it is essential to have tools for analyzing code mixed languages with the increasing demand. There are excellent NLP tools present for English and Spanish languages. In the recent years, NLP tools have been developed for Indic languages such as Hindi, Bengali, Telugu, Tamil and Marathi, but there aren't many tools present for code mixed Indic languages. These tools can be helpful in extracting useful information in many applications such as product analysis and symptoms analysis.

In this chapter, algorithms have been proposed for basic NLP tools, such as POS tagging, named entity recognition and language identification, to process code mixed text. Part of speech tagging is an essential tool for text processing and is widely used for many applications. Similarly, named entity recognizer has been an essential tool of NLP and text mining community for various applications. Though much of the work has been done for these tools in English and other monolingual languages but there has been an increase in demand of these tools for processing code mixed text. Section 7.1 discusses the approaches proposed for POS tagging for code mixed social media text for Indian languages. Section 7.2 discusses the approaches and its variants for named entity recognizer for code mixed social media text for Indian languages.

7.1 POS Tagging for Indian Code Mixed Social Media Text

One of the most fundamental part of linguistic pipeline is POS tagging, a basic form of syntactic analysis which has countless applications in NLP. POS tagging is the process of annotating words in a text to their corresponding part of speech based on the context of the word and the definition of the word. POS tags give a linguistic structure to the sentence. Besides, it is used in many applications such as text to speech conversion, speech recognition, word sense disambiguation, machine translation, sentiment analysis and information retrieval.

Much work has been already done for POS tagging in monolingual text and algorithms have achieved quite high accuracies. With the increase in the code mixed text on social media, these algorithms do not perform well because of the syntax and grammatical changes. Hence, it has become important to develop part of speech tagger for code mix text.

In this section, three approaches have been proposed to build POS tagger for code mixed social media text. Each of these approaches have their variants. First two proposed algorithms for part of speech tagger used traditional machine learning technique. Most of the current techniques for POS tagging in monolingual scripts focus on feature engineering. Hence, first two algorithms identify and engineer features for identifying and associating correct POS tags. However, feature engineering requires linguistic knowledge, and people might miss out on the underlying features. Moreover, traditionl machine learning techniques need to engineer the features on available dataset. Deep learning is a branch of machine learning and is an emerging field with a lot of interesting work being done in various domains such as image recognition and text summarization. It uses deep neural networks with multiple layers which models high level abstractions.

BRNN have been used in the past for various application that involves sequence learning and were proved quite effective. Few of these applications involved predicting sequential data like handwriting and speech. POS tagging requires sequential information. The in-

formation of the tags preceding the current tag and the subsequent tags should be known to make the correct prediction. Problem of knowing sequential information and the problem of vanishing gradients is solved using neural networks. The attraction that deep learning holds is that one does not require to identify features or build resources like a dictionary for morphemes and other linguistic units.
Third proposed approach for POS tagging is based on BRNN. LSTM performs better as compared to simple recurrent neural networks, hidden neural networks and other alternative neural networks. LSTM network can learn from its past experience, and unlike traditional RNN(s), LSTM network is well-suited to learn from experience to classify, process and predict time series when there are very long time lags of unknown size between important events. This is one of the primary reasons why LSTM outperforms alternative RNN(s) and hidden markov models and other sequence learning methods in numerous applications. Hence, proposed approach is using LSTM with its variant. LSTM and RNN will be used interchangeably due to the above mentioned explanation in this section.
Two kinds of datasets [Jamatia & Das 2016] are used for the problem of POS tagging; Fine grained and Coarse Grained. The fine grained dataset has *39* POS tags present while coarse grained has *19* POS tags. Corpus is based on three languages pairs, Telugu-English, Bengali-English and Hindi-English. For each language pair, data is collected using three different data source namely Facebook, Twitter and WhatsApp.

7.1.1 Approach 1: Ensemble based POS Tagger

A pipelined process has been proposed in this approach which consists of mainly three phases i.e. pre-processing, feature extraction and classification as shown in Figure 7.1. Further, two variants are proposed for this approach named ensemble_v1 and ensemble_v2. The difference in both the algorithms lies in the last phase of classification where ensemble_v2 uses an additional dictionary to identify the POS Tags.

1. Pre-Processing
   Since many Facebook posts, tweets, WhatsApp messages in dataset [Jamatia & Das 2016] consisted of more than one sentence. They are demarcated into individual sentences using the presence of full stop, exclamation mark, question marks

and words starting with capital letters. After that, all words are converted into lowercase for normalization.

2. Feature extraction

   A lot of importance is given to feature set extraction. Table 7.1 depicts all the features extracted from this dataset. Features like prefixes, suffixes, bag-of-words which are popular in many NLP tasks are used in the approach. For this purpose, a list of suffixes, prefixes and a bag-of-words are created from the training dataset. The suffixes and prefixes are limited to three letters. The position feature is given by the position of the word in the sentence divided by the length of that sentence. It is partitioned into discrete values start, middle and end. For selecting the best combination of features, the module GridSearchCV [Pedregosa *et al.* 2011b] is used. The best combination of features found are startsWith@, startsWith#, contains_digit, Pos_1, word level information, suffix and the current word.

Table 7.1: Features

Features	Description
startsWith@, startsWith#, startsWith_http, contains_digi	Binary Features
Position	Nominal Feature with values start, middle and end
POS_1, POS_2	Nominal Feature POS_1- POS tag of previous word, POS_2- POS tag of previous to previous word
Word Level information of current word	Nominal feature
Prefix, Suffix and the Current Word	Sparse Matrix

3. Classification for ensemble_v1

   After the stages of pre-processing and feature extraction, the model is trained using three classifiers. For classification, an ensemble of the three classifiers, i.e. random forest, logistic regression and naive bayes is used, particularly weighted majority voting with more weights is given to the classifier, which is performing better.

4. Classification for ensemble_v2

   For ensemble_v2, few changes are incorporated into the ensemble_v1 system. First of all, a dictionary of the *100* most common English words along with their POS tags is manually constructed both for the fine grained and coarse grained tag set. These words comprised about *20%* of the whole dataset, and led to a noticeable improvement in the model. Since many sentences resembled the structure of English sentences and hence a POS tagger for English is used. The NLTK tagger[1] is used but since the tag set used by NLTK is different from the tag set used in the dataset, tags are manually mapped to each other.

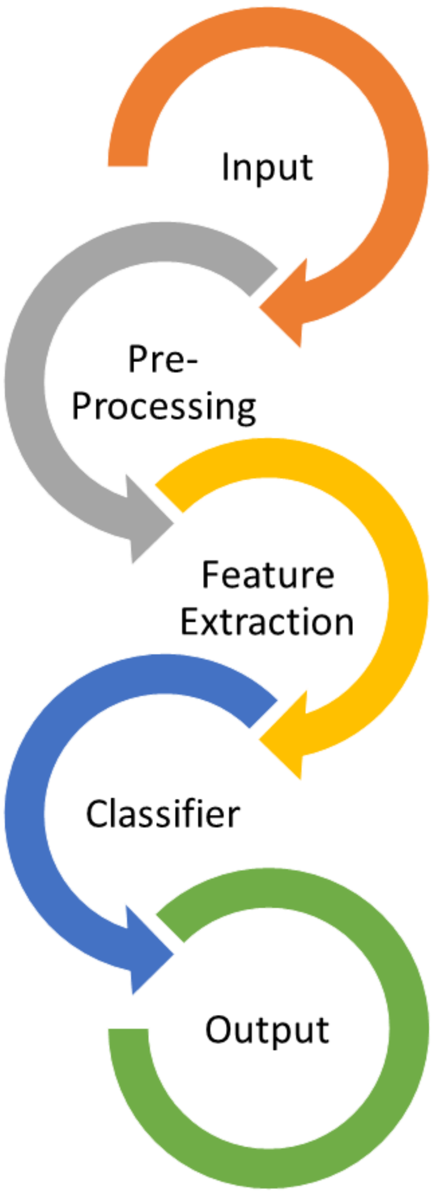


Figure 7.1: Block diagram of ensemble based POS tagger

[1]http://www.nltk.org/

7.1.1.1 Algorithm for Approach 1: Ensemble based POS Tagger

Algorithm 7.1 explains the proposed algorithm for POS tagging of code mix social media text. The algorithm is designed such that it takes given dataset as input and returns POS tagged words as output for each of the token. Initial preprocessing has been done by the demarcation of sentences and converting the dataset to lower case using *Case_Conversion* function. A bag of words is then created using the *create_bag_of_words* function. Suffix and prefix dictionary are created using *suffix* and *prefix* function which considers *n* characters, where *n* varies from *1* to *3*. Tokenize function then separates the words and create a list of tokens. For each token in the list, features mentioned in Table 7.1 are extracted using the *feature_extraction* function. POS tag of previous token is identified using *POStag* function, which has been passed to aid feature extraction. Finally, ensemble classifier of random forest, logistic regression and naive bayes is used to detect POS tags.

Algorithm 7.1: Algorithm for POS Tag detection

```
input : Mixed language dataset D with one of
        languages(Hindi, Telugu or Bengali) mixed with
        English
output: POS tagged words W
1  Initialization: test_predictions P=[];
2  Case_Conversion(D);
3  bag_of_words=create_bag_of_words(D);
4  suffix_dict=suffixes(D);
5  prefix_dict=prefixes(D);
6  mylist =tokenize(D);
7  for $i = 0$ to $mylist.length$ do
8      listp=POStag(mylist[i-1]);
9      features=feature_extraction(mylist[i], suffix_dict, prefix_dict,
         listp);
10 tag=ensemble.predict(features);
```

7.1.2 Approach 2: Tree based classifier for POS Tagging

Second approach for POS tagging is also based on traditional machine learning techniques. It is different from approach 1 in terms of the classifier and the usage of features. Two variations are proposed for the approach 2 as well, named tree_v1 and tree_v2.

1. Proposed tree_v1 approach

   The proposed tree_v1 approach consists of a set of features that comprise the feature vector for each token. Following features are used:

   (a) IsNumberPresent : This is a binary feature which checks for the presence of digits in the given token. It returns *1* if present and *0* if not present.

   (b) ISHashTag Present: This is a binary feature which checks for the presence of hashtags in the given token. It returns *1* if present and *0* if not present.

   (c) IS@Present : This is a binary feature which checks for the presence of the symbol '@' in the given token. It returns *1* if present and *0* if not present.

   (d) IsSymbol Present: All Characters which aren't either digits or numbers or @ or hash tag are considered to come into the category of symbols. This is a binary feature which checks for the presence of the symbols in the given token. It returns *1* if present and *0* if not present.

   (e) Language of Prev word(L_{-1}), Prev Prev Word(L_{-2}), Current word(L_0) & Next Word(L_{+1}): These features are considered as they help in analyzing the way words are used.

   (f) POS Tag of Prev word(P_{-1}), Prev Prev Word(P_{-2}): These features are considered as they help in analyzing the structure of the sentence.

   (g) Position of Word in sentence: There are many ways of representing this feature. Following formula is used in proposed algorithm:

   $$\frac{No\_of\_Words\_present\_before\_current\_word}{Total\_words\_in\_sentence}$$

   (h) Prefix of the Token(Pr_1, Pr_2, Pr_3): Presence of most common Prefix is considered as a feature. Prefixes of length *1* to *3* are considered. The list of most common prefixes are extracted from the given test data.

   (i) Suffix of the Token(S_1, S_2, S_3): Presence of most common Suffix is considered as a feature. Suffixes of length *1* to *3* are considered. The list of most common suffixes are extracted from the given test data.

The feature vector is built by extracting features for each token in a sentence. Finally, list of all feature vectors is extracted and trained using random forest.

2. Proposed tree_v2 approach

   The tree_v2 approach has the same features as that of the tree_v1, with an addition of one more extra feature, which is three topmost common POS tags possible for a given token. This is predicted by using a dictionary which is developed using the training data of ICON 2015. The resulting feature vectors are trained on an extremely randomized tree [Pedregosa *et al.* 2011b].

7.1.2.1 Algorithm for Approach 2: Tree based classifier for POS Tagging

Algorithm 7.2 depicts the working of the tree_v1 and tree_v2. Both of them have the same algorithm, but the difference exists in the working of *Create_Feature_Vector* function. In tree_v2 an extra feature has been added as described in the previous section. The input *S* contains a list of sentences, and each sentence is a list of tokens. The list also contains the respective language of each token. For example, the sentence "I like to watch movies." is represented as ['I', 'en', 'like', 'en', 'to', 'en', 'watch', 'en', 'movies', 'en'].
The feature vectors are computed for each token of the sentence of the input data (implemented by: *Create_Feature_Vector*). While creating feature vectors *prev* and *pprev* list are passed as inputs to create the feature vector where *prev* represents the previous occurring token and *pprev* represents the previous to the previous occurring token. These features are then given to a *classifier* for predicting the POS Tag. Random forest and extremely randomized tree classifier are used as a classifier for tree_v1 and tree_v2 respectively.

7.1.3 Approach 3: Bidirectional LSTM based POS Tagger

Traditional neural networks face the problem of rigidity due to fixed size vector of input and output. The number of computational steps in the neural network is also fixed. Recurrent neural network is superior to traditional neural network because they can handle sequences. Traditional neural networks are also limited by the number of layers in the model which can only perform a fixed number of computations.

Algorithm 7.2: Algorithm for Detecting paraphrases

```
input  : List of Code-Mixed sentences along with their
         Language, S, Training class labels, T
output: Predicted Parts of speech Tag, P
Initialization: P=[], FV=[];
for $i = 0$ to $S.length$ do
    prev = { };
    pprev= { };
    for $j = 0$ to $(S[i].Length)/2$ do
        FV = Create_Feature_vector(S[i ][ j ], prev, pprev);
        tag = clf.predict(FV);
        pprev =prev;
        prev= {S[i][2*j], S[i][2*j+1], tag };
        P.append(tag);
```

The appeal of recurrent neural networks lies in the fact that they can operate on sequences and are more capable of building intelligent systems. At every state, RNN also combines the input vector with the state vector using a step function to create a new state vector. The output vector obtained from the RNN is an amalgamation of the current input and the inputs that have been considered in the past. Step function ensures that RNN remembers the context. The internal state of the RNN is updated every time the step function is called.

Algorithm 7.3 specifies the step function which updates the hidden state of the RNN. The

Algorithm 7.3: Step function for a RNN

```
def step(self, x):;
# update the state of the RNN;
self.h = tanh(np.dot(self.S_hh, self.h) + dot(self.S_xh, x));
# compute the output vector;
y = np.dot(self.S_hy, self.h);
return y;
```

parameters are three matrices *S_hh, S_xh, S_hy*. The hidden state *h* is initialized with zero vector. In *tanh* function, one of the inputs specify the current input, and the other specifies the state of the previous iteration. The *tanh* function ensures that the activation range is normalized in the range from *-1* to *1*. The final output is dot multiplication of current state vector and the parameter *S_hy*. LSTM is a better form of a recurrent neural network because it has a more powerful update equation and backpropagation algorithm.

7.1.3.1 Algorithm for Approach 3: Bidirectional LSTM based POS Tagger

Algorithm 7.4 explains the working of proposed approach 3. Given an input sequence $a = a_1, a_2, a_3, ...,$ task is to generate an output sequence $b = b_1, b_2, b_3, b_4, ...,$ where a is the code mixed data and b is the corresponding POS tags.
For pre processing, tags for the words in training data are stored in a dictionary. The data is then converted to sentences containing POS tags only. For example, if the given sentence is Mixed/G_J dabay/G_N Wala/G_PRT, it is converted to G_J G_N G_PRT.
Each POS tag in the input is encoded into a vector using *1-of-k* encoding (i.e. all zero except for a single one at the index of the character in the vocabulary) and fed into RNN one by one using step function. On every output vector, a softmax classifier or cross entropy loss function is applied at the same time.
The RNN is trained with mini-batch Stochastic Gradient Descent to stabilize the updates. For RNN, following parameters are chosen based on the performance of training data. The size of hidden layer is *256*, with the number of layers as *8* and sequence length as *2*. The RNN is trained for 50 epochs with the learning rate as *0.002* and decay rate as *0.97*.
At test time, if a word is present in the dictionary, the tag is chosen from the dictionary. Otherwise, RNN predicts a distribution of next likely tag. Two versions of approach 3 are proposed named, Bi_LSTM_v1 and Bi_LSTM_v2. Propsed approach, Bi_LSTM_v2 has an additional dictionary for English words with the corresponding POS tags.

7.1.4 Data Analysis

The dataset for ICON 2016 task [Jamatia & Das 2016] on POS tagging in code mixed social media text has three different language-pairs, i.e. Hindi-English, Bengali-English, Telugu-English, each for three different platforms - Facebook, Twitter and WhatsApp. The training dataset contained all the words in Roman script along with their word level information and POS tag.Figure 7.2 shows the different tags used at coarse grained and fine grained level. Table 7.2 depicts the overall dataset statistics i.e. Facebook, Twitter, and WhatsApp whereas Table 7.3 depicts only the Facebook dataset statistics. The average sentence length for each language is calculated. The average sentence length is abnormally

Algorithm 7.4: Algorithm for POS tagging code mixed text

```
input  : A sequence$a$ = $a_1, a_2, a_3, ...,$ where $a$ is the code
         mixed data with language tags.
output: An output sequence $b = b_1, b_2, b_3, b_4, ...,$ where $b$ is
         the POS tags corresponding to the input data.
1  Preprocessing;
2  Convert input sequence to V (1-of-k vectors);
3  Feed the vector into the RNN individually with the step function;
4  Use the standard Softmax classifier on every output vector
     simultaneously;
5  Train RNN on V;
6  for $x_i$ in $x$ do
7      if $x_i$ in dictionary then
8          if $x_i$ has only one tag then
9              $y_i$ = dictionary[$x_i$];
10         else if dictionary[$x_i$].length>1 then
11             $y_i$ = predict(dictionary[$x_i$]);
12         else
13             $y_i$ = predict(y);
```

high for Hindi and Bengali as compared to Telugu for Facebook dataset. Table 7.4, 7.5, 7.6 shows the complete analysis of coarsed grained dataset for Hindi, Telugu and Bengali respectively.

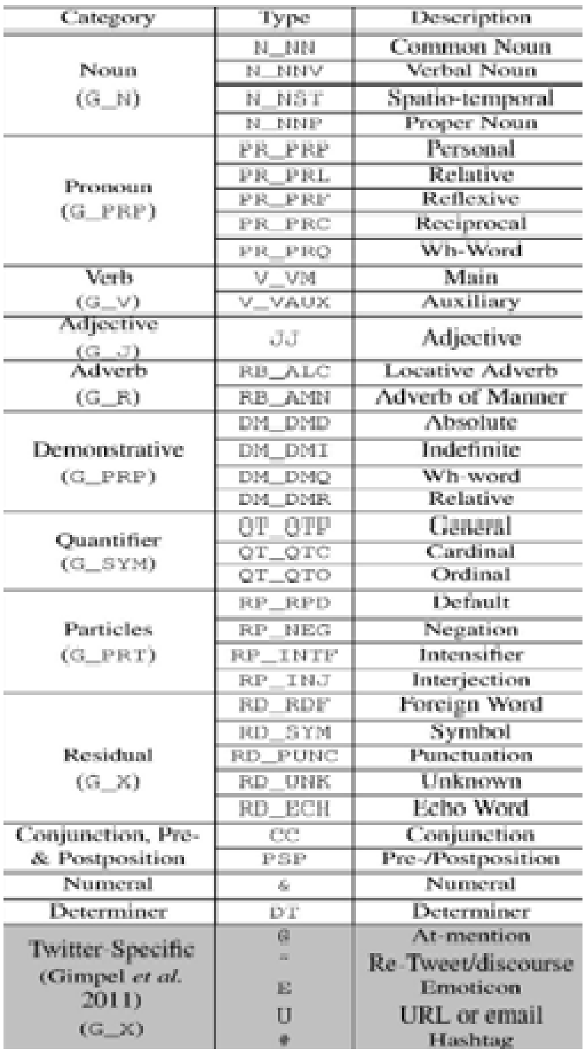
Table 7.2: Overall Dataset Statistics

Language	Words	Sentences	Average Sentence Length
Hindi	37260	2317	16
Bengali	15326	634	24
Telugu	24460	1576	15

Table 7.3: Facebook Dataset Statistics

Language	Words	Sentences	Average Sentence Length
Hindi	21386	771	27
Bengali	7609	147	51
Telugu	10780	749	14

As shown in Figure 7.3, 7.4 and 7.5 the distribution of language tags are different for all the three languages. There are differences in the composition of language tags in testing and training data as well.



Category	Type	Description
Noun (G_N)	N_NN	Common Noun
	N_NNV	Verbal Noun
	N_NST	Spatio-temporal
	N_NNP	Proper Noun
Pronoun (G_PRP)	PR_PRP	Personal
	PR_PRL	Relative
	PR_PRF	Reflexive
	PR_PRC	Reciprocal
	PR_PRQ	Wh-Word
Verb (G_V)	V_VM	Main
	V_VAUX	Auxiliary
Adjective (G_J)	JJ	Adjective
Adverb (G_R)	RB_ALC	Locative Adverb
	RB_AMN	Adverb of Manner
Demonstrative (G_PRP)	DM_DMD	Absolute
	DM_DMI	Indefinite
	DM_DMQ	Wh-word
	DM_DMR	Relative
Quantifier (G_SYM)	QT_QTF	General
	QT_QTC	Cardinal
	QT_QTO	Ordinal
Particles (G_PRT)	RP_RPD	Default
	RP_NEG	Negation
	RP_INTF	Intensifier
	RP_INJ	Interjection
Residual (G_X)	RD_RDF	Foreign Word
	RD_SYM	Symbol
	RD_PUNC	Punctuation
	RD_UNK	Unknown
	RD_ECH	Echo Word
Conjunction, Pre- & Postposition	CC	Conjunction
	PSP	Pre-/Postposition
Numeral	$	Numeral
Determiner	DT	Determiner
Twitter Specific (Gimpel *et al.* 2011) (G_X)	@	At-mention
	~	Re-Tweet/discourse
	E	Emoticon
	U	URL or email
	#	Hashtag

Figure 7.2: Tags used for POS Tagging [Jamatia & Das 2016]

Telugu dataset contains a lot of universal, English and Telugu tags. All the Telugu words present are transliterated into English. It has some named entities and acronyms present in small quantities. Though there is some variation in the percentages of tags in testing data, the composition almost remained the same.

In the case of Bengali dataset, there is a wide range of tags present. Majority of the tags are Bengali, and these words (Bengali) are transliterated into English. It is followed by English, universal and undefined tags. There are a significant amount of named entities present. Moreover, transliterated Hindi words and acronyms are present.

The Hindi dataset has English and Hindi tags. The Hindi words are also transliterated into English. There are named entities and universal tags in significant quantities along with acronyms in minute quantities. For Hindi and Telugu languages, testing data is less

Table 7.4: Analysis of Coarse Grained dataset for Hindi (Facebook, Whatsapp and Twitter

Hindi	FB	Hindi	TWT	Hindi	WA
No of Sentences	771	No of Sentences	1096	No of Sentences	763
POS Tag		POS Tag		POS Tag	
G_X	3356	G_X	4156	G_X	589
CC	671	CC	196	CC	62
G_V	3788	G_V	2466	G_V	525
DT	1247	DT	300	DT	38
G_R	1088	G_R	410	G_R	26
G_SYM	153	G_SYM	139	G_SYM	27
G_N	4186	G_N	6689	G_N	1209
G_J	1184	G_J	772	G_J	170
G_PRT	766	G_PRT	249	G_PRT	190
PSP	1894	PSP	708	PSP	174
Null	1	null	1	null	1
G_PRP	2008	G_PRP	1071	G_PRP	192
$	270	$	154	$	15
Language Tag		Language Tag		Language Tag	
undef	2	en	3731	hi	2539
En	13213	ne	413	en	363
Ne	656	hi	9779	univ	281
Hi	2855	mixed	1	ne	35
mixed	7	rn	1		
Univ	3628	univ	3354		
Acro	251	acro	32		

than that of training data, but it is otherwise for the Bengali dataset.

There are some inconsistencies in the datasets of all the three languages like an improper classification of words into language tags. Universal tags are classified incorrectly in more than *500* cases. For example, in Telugu corpus the word *'ntr'* is classified as *univ*, but *univ* tag is given to tokens containing # or @ or symbols. Results evevaluated for all the three approaches are compared with the *9* baseline systems in paper by [Jamatia & Das 2016]

7.1.5 Experiments and Results for Approach 1

The problem stated by the ICON-2016 organizers involves part-of-speech tagging of code-mixed data in three different formats (Hindi-English, Bengali-English, Telugu-English) belonging to three different platforms Twitter, Facebook and WhatsApp. In all four combinations of each system, namely ensemble_v1 and ensemble_v2 both for fine-grained

Table 7.5: Analysis of Coarse Grained dataset for Telugu (Facebook, Whatsapp and Twitter

Telugu	FB	Telugu	TWT	Telugu	WA
No of Sentences	743	No of Sentences	743	No of Sentences	493
POS Tag		POS Tag		POS Tag	
$	133	G_X	4261	G_X	2434
CC	175	CC	105	CC	64
G_V	1147	G_V	1127	G_V	620
DT	269	DT	193	DT	138
G_R	244	G_R	224	G_R	160
G_SYM	28	G_SYM	33	G_SYM	1
G_N	4160	G_N	3940	G_N	2752
G_J	597	G_J	543	G_J	350
G_PRT	162	G_PRT	288	G_PRT	232
PSP	492	PSP	484	PSP	271
Null	71	null	132	null	75
G_PRP	332	G_PRP	537	G_PRP	247
G_X	2204	$	135	$	55
Language Tag		Language Tag		Language Tag	
En	3728	em	1	en	1887
Ne	391	en	3197	unin	1
Eb	1	ne	256	ne	97
Mix	1	nr	1	Te	2104
G_XN	1	mix	2	univ	3302
Te	2642	PSP	1	acro	8
Univ	3210	the	1		
Acro	39	te	4046		
Unit	1	univ	4472		
		acro	24		
		unit	1		

and coarse-grained. Ensemble classifier with same parameter settings, is used in both the variations of ensemble based POS tagger.

7.1.5.1 Evaluation and Discussion

The system performed well on coarse grained tag sets with an accuracy of 80.6% for Telugu English dataset but performed comparatively less on the fine grained tag set with the accuracy dropping to 76.4%. This may be because there is a significantly higher number of class labels in the fine grained dataset. For the WhatsApp dataset, the results are comparatively less because of the highly informal language used. Highest accuracy

Table 7.6: Analysis of Coarse Grained dataset for Bengali (Facebook, Whatsapp and Twitter

Bengali	FB	Bengali	TWT	Bengali	WA
No of Sentences	147	No of Sentences	172	No of Sentences	304
POS Tag		POS Tag		POS Tag	
G_X	1350	$	18	G_X	439
CC	144	CC	64	CC	95
G_V	925	G_V	641	G_V	502
DT	85	DT	51	DT	44
G_R	206	G_R	129	G_R	117
G_SYM	131	G_SYM	109	G_SYM	75
G_N	2494	G_N	1035	G_N	1475
G_J	376	G_J	120	G_J	194
G_PRT	253	G_PRT	118	G_PRT	74
PSP	614	PSP	192	PSP	152
Null	1	null	3	G_PRP	303
G_PRP	773	G_PRP	438	$	42
$	40	G_X	762	Language Tag	
Language Tag		Language Tag		en	7
undef	1	ne+bn_suffix	4	bn	692
En	2199	undef	25	ne	14
Bn	3589	bn	1793	undef	1320
Ne	215	ne	110	univ	1405
Hi	40	en+bn_suffix	4	acro	74
mixed	1	en	979		
Univ	1261	hi	10		
Acro	86	univ	730		
		acro	25		

achieved with the proposed system is for Telugu-English dataset.

The overall average results of proposed approach on three platforms i.e. Facebook, Twitter and WhatsApp (both coarse-grained and fine-grained) are depicted in Table 7.7 and 7.8.

7.1.5.2 Error Analysis

There are a few phases where the proposed approach might have resulted in errors. Firstly, in the pre-processing step, sentences might not have been appropriately demarcated because in platforms like WhatsApp the syntax used is very informal. This might have also caused the features such as position and POS tag of last word, to be of no significance for those particular instances. Moreover, in the training data for Hindi-English,

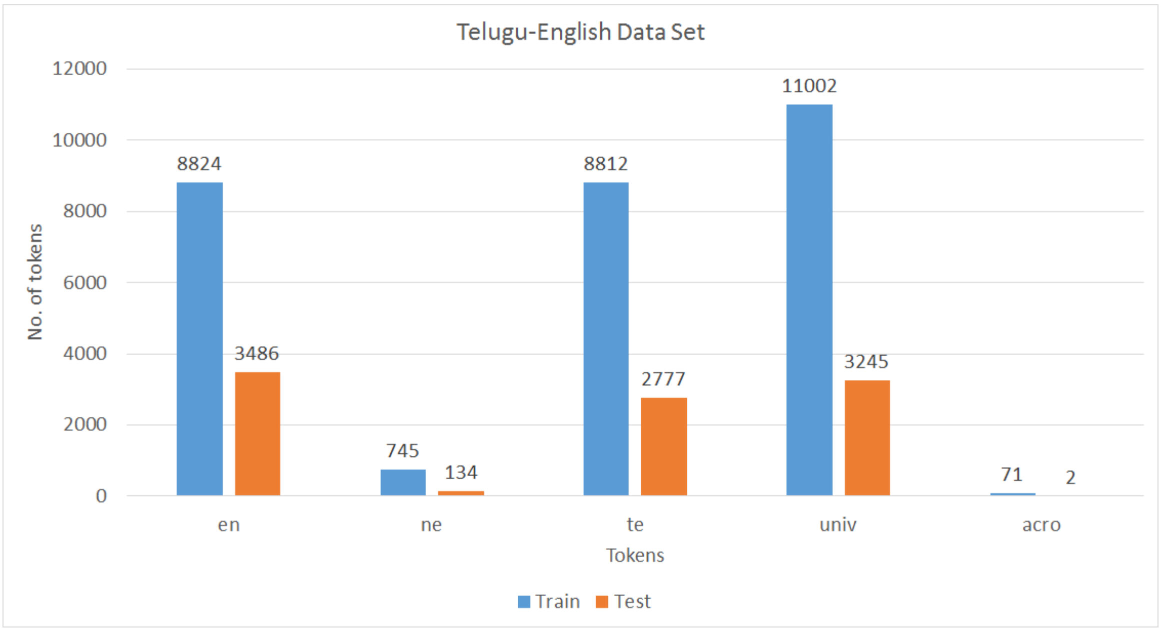


Figure 7.3: Language Tags in Telugu Dataset

Table 7.7: F-measure of Coarse Grained dataset using approach 1

	ensemble_v1	ensemble_v2
Telugu-English	80.06	77.7
Hindi-English	71.03	71.655
Bengali-English	71.03	71.83

the percentage of English words is very high compared to Hindi, and hence the trained model could be biased. Similar, is the case of Bengali-English dataset. However, for Telugu-English dataset, there is a balanced mix of Telugu and English words and hence, Telugu-English system performed comparatively much better than the others.

7.1.6 Experiments and Results for Approach 2

As shown in Table 7.9 and 7.10, tree_v1 has performed more or less the same as compared to the tree_v2 on all the three Bengali datasets. The system achieved the highest F-measure on the Facebook dataset for both fine grained *(74.5803%)* and coarse grained *(77.944%)* systems respectively.

On the Hindi corpus (as shown in Tables 7.11 and 7.12), there is a decrease in the F-measure when compared to Bengali (as shown in Tables 7.9 and 7.10) and Telugu(as shown in Tables 7.13 and 7.14) datasets. This might be due to the change in the com-

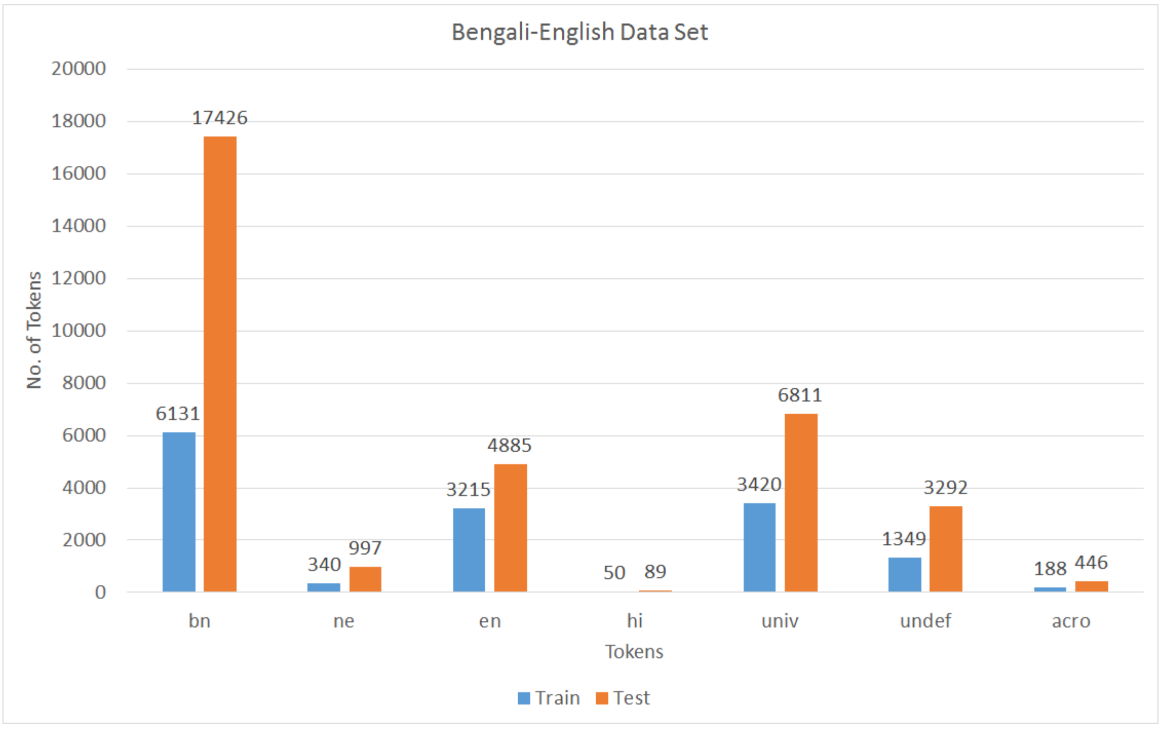


Figure 7.4: Language Tags in Bengali Dataset

Table 7.8: F-measure of Fine Grained dataset using approach 1

	ensemble_v1	ensemble_v2
Telugu-English	76.4	76.07
Hindi-English	74.12	79.03
Bengali-English	68.3	71.73

position of Hindi testing and training data. The Hindi dataset has more Hindi words as compared to English words in the testing data whereas they are almost the same in the training data. The highest F-measure achieved for Hindi is for the Twitter corpus. It achieved F-measure of *78.573%* in the fine grained system and *77.922%* in the coarse grained part.

The system performed the best on the Telugu datasets (based on average F-measure) in comparison to all the languages. The system has the highest F-measure on the WhatsApp data for the fine grained system with a value of *77.602%*, as shown in Table 7.14 but for the coarse grained model, the Facebook dataset performed the best with a F-measure *77.343%*.

It can be concluded from Table 7.9, 7.10, 7.11, 7.12, 7.13 and 7.14, that the proposed system has very high recall and low precision for all the three languages. One possible

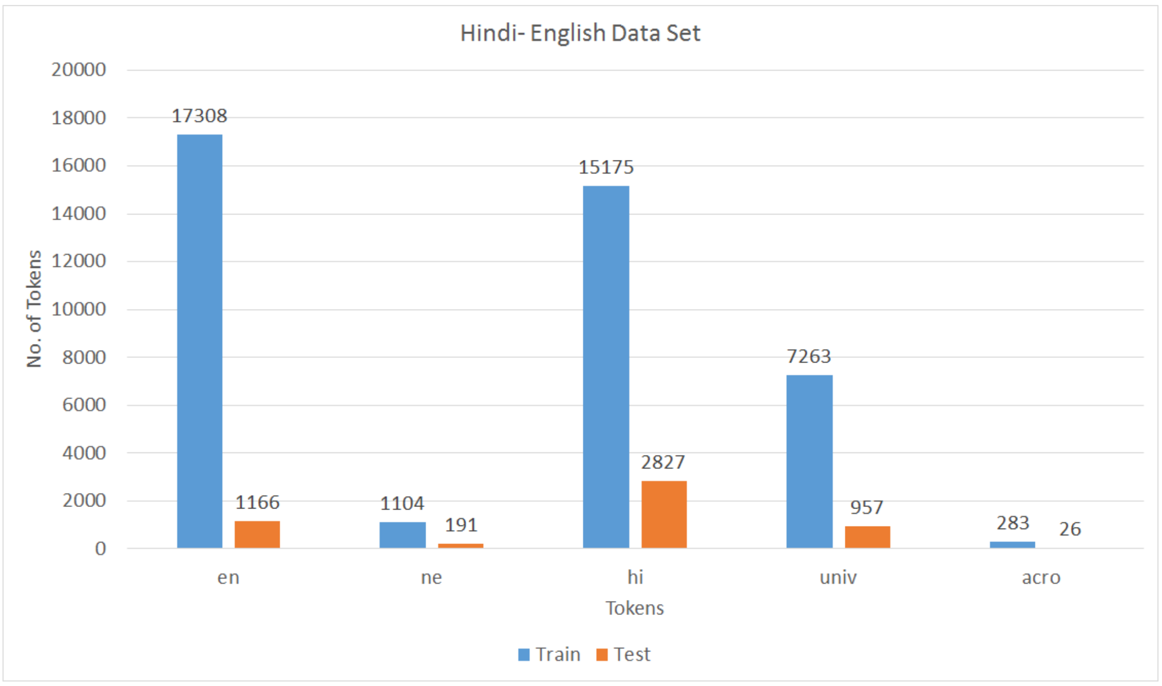


Figure 7.5: Language Tags in Hindi Dataset

Table 7.9: Results for Bengali Corpus (tree_v1)

Category	Precision	Recall	F Score
FB_FG	0.595	0.999	74.5803
FB_CG	0.639	0.999	77.94396
TWT_FG	0.567	0.997	72.28887
TWT_CG	0.557	0.998	71.512
WA_FG	0.581	0.996	73.377
WA_CG	0.619	0.992	76.23191

explanation can be, if one-vs-all approach is used while calculating precision, then even if few false negatives are present for a specific class they might have accumulated as false positive for another class precision.

7.1.6.1 Error Analysis

Some of the features used might have caused misclassification of tags. One of the feature might be the usage of a dictionary in tree_v2. Due to less data in dictionary created, this feature might have affected the system's performance. Hence, for building an efficient POS Tagger large dictionary is needed.

Table 7.10: Results for Bengali Corpus (tree_v2)

Category	Precision	Recall	F Score
FB_FG	0.595	0.999	74.5803
FB_CG	0.639	0.999	77.944
TWT_FG	0.566	0.996	72.1813
TWT_CG	0.559	0.998	71.67
WA_FG	0.603	0.996	75.103
WA_CG	0.627	0.991	76.8056

Table 7.11: Results for Hindi Corpus (tree_v1)

Category	Precision	Recall	F Score
FB_FG	0.488	0.978	65.089
FB_CG	0.481	0.999	64.92
TWT_FG	0.654	0.989	78.744
TWT_CG	0.639	0.999	77.922
WA_FG	0.54	0.989	69.805
WA_CG	0.618	0.618	61.845

7.1.7 Experiments and Results for Approach 3

For Bi_LSTM_v2, a dictionary is defined based on the data from previous ICON conferences. Approximately no changes are observed in the results obtained by Bi_LSTM_v2 and Bi_LSTM_v1. For coarse grained tagging, a better F-measure for WhatsApp code mixed data is obtained as compared to Facebook and Twitter for Hindi, and Bengali code mixed language. Recall for all the languages is close to 99% as can be seen from Figure 7.7 and 7.9. However, inconsistencies are observed for data (Facebook and Twitter) in Bengali and Twitter data of Hindi. The system has an F-measure score of *82%* for WhatsApp in the Bi_LSTM_v1 and *80%* in the Bi_LSTM_v2. One reason for this can be that training dataset for Bengali (Facebook and Twitter) has thrice number of Bengali words as compared to test dataset. As can be inferred from Figure 7.6 and 7.8, precision is low throughout. The reason for low precision might be an accumulation of false positives in all the classes of the POS tags. Usually, in traditional machine learning systems that use feature extraction, a drop in accuracy is seen from coarse grained tagging to fine grained training because of the increase in number of classes. However, proposed algorithm overcomes this issue. The same pattern can be observed for both coarse grained and fine grained POS tagging in case of English Bengali code mixing. For coarse grained and fine grained tagging for

Table 7.12: Results for Hindi Corpus (tree_v2)

Category	Precision	Recall	F Score
FB_FG	0.493	0.974	65.466
FB_CG	0.484	0.998	65.173
TWT_FG	0.652	0.989	78.573
TWT_CG	0.639	0.999	77.922
WA_FG	0.526	0.986	68.632
WA_CG	0.608	0.608	60.848

Table 7.13: Results for Telugu Corpus (tree_v1)

Category	Precision	Recall	F Score
FB_FG	0.617	0.992	76.12
FB_CG	0.631	0.99	77.06
TWT_FG	0.617	0.983	75.81
TWT_CG	0.564	0.993	71.94
WA_FG	0.645	0.976	77.7
WA_CG	0.608	0.992	75.39

Telugu comparable results are obtained with an F-measure of around 73%. As can be seen from Figure 7.8, for coarse grained tagging an F-measure of 70% for WhatsApp and 57% for Twitter and 63% for Facebook is obtained. For fine grained tagging in Hindi, 70% score for Twitter, 68% for Facebook and 64% for WhatsApp is obtained. The best results obtained through the proposed algorithm of Bi_LSTM_v1 as well as Bi_LSTM_v2 is 82%.

7.1.7.1 Error Analysis

Recursive neural networks have been used in the past for part of speech tagging but they have been used mostly for English language. Recurrent neural networks have not been used to tag code mixed data in the past. Since the performance of the RNN is dependent on the amount of data used to train it, better performance could be obtained given more data. Moreover, the noise in the data reduces the performance of the RNN. RNN(s) also depend on the sequence length of the input. The input of same sequence length has also given better performance. This also is one possible source of error. If there had been enough data, sentences of the same length could have been clustered together to train one RNN model. RNN(s) are good at exploiting patterns in data. With code mixed data, one problem could be that of missing patterns. In English, a subject is followed by a verb,

Table 7.14: Results for Telugu Corpus (tree_v2)

Category	Precision	Recall	F Score
FB_FG	0.594	0.985	74.092
FB_CG	0.635	0.99	77.343
TWT_FG	0.619	0.977	75.793
TWT_CG	0.553	0.992	71.0131
WA_FG	0.646	0.972	77.602
WA_CG	0.595	0.989	74.29

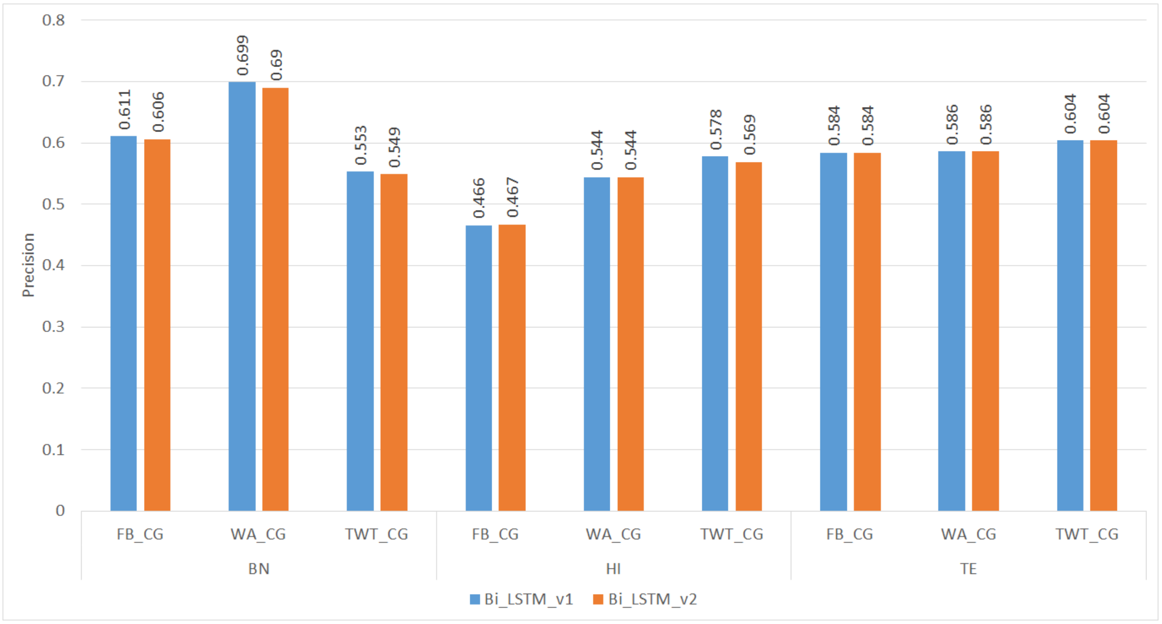


Figure 7.6: Precision for Coarse Grained POS Tagging using approach 3

but in Hindi, a subject is followed by an object. Bilingual users do not tend to follow syntactic rules of the language while writing on social media sites and thus that could be one source of error.

7.2 Named Entity Recognition for Code Mixed Social Media Text

Entity recognition is a very important sub-task of information extraction and find its applications in information retrieval, machine translation and other NLP applications such as co-reference resolution. Named Entities (NE) are names of famous persons, organizations, locations and animals. NER has also found its use in sentiment analysis, where recognizing named entities is important as they do not add much value to the sentiment of the statement. Similarly, while tagging articles named entities are required for better

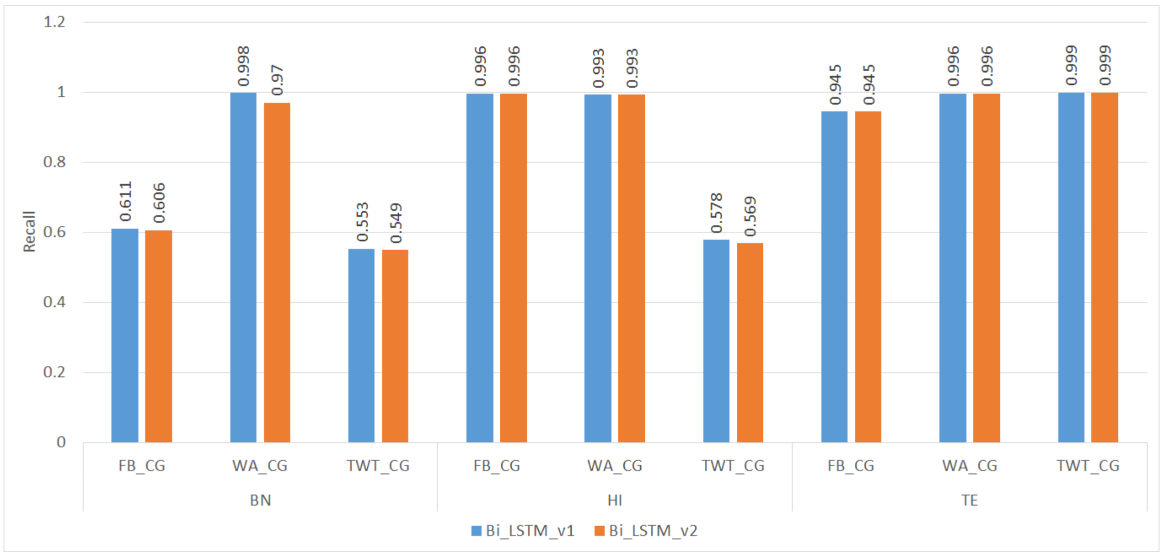


Figure 7.7: Recall for Coarse Grained POS Tagging using approach 3

search results. There are many such applications where named entities play an important role in the automatic processing of text. Although, much work has been done in this area for monolingual text, recognizing named entities in code mixed text for Indian languages still remains a problem at large, due to the informal nature of the text. In this section, proposed approaches for the task of entity recognition in code mix tweets for Indian languages are discussed. Proposed approach identifies named entity in code mix tweets of English-Hindi and Tamil-English code mixed tweets and can be further extended to other languages. The problem is to identify the name of various entities such as a person, organization, movie, location in a given code mixed tweet.

7.2.1 Approach for Named Entity Recognition for Code Mixed Social Media Text

A word level NE recognition system is designed to recognize named entities in a tweet. As shown in Figure 7.12 the proposed methodology involves a pipelined approach for detecting each NE tag and has been divided into following four phases:

1. Pre-processing

2. Number Based Named Entity Recognition

3. Gazetteer List Based Named Entity Recognition

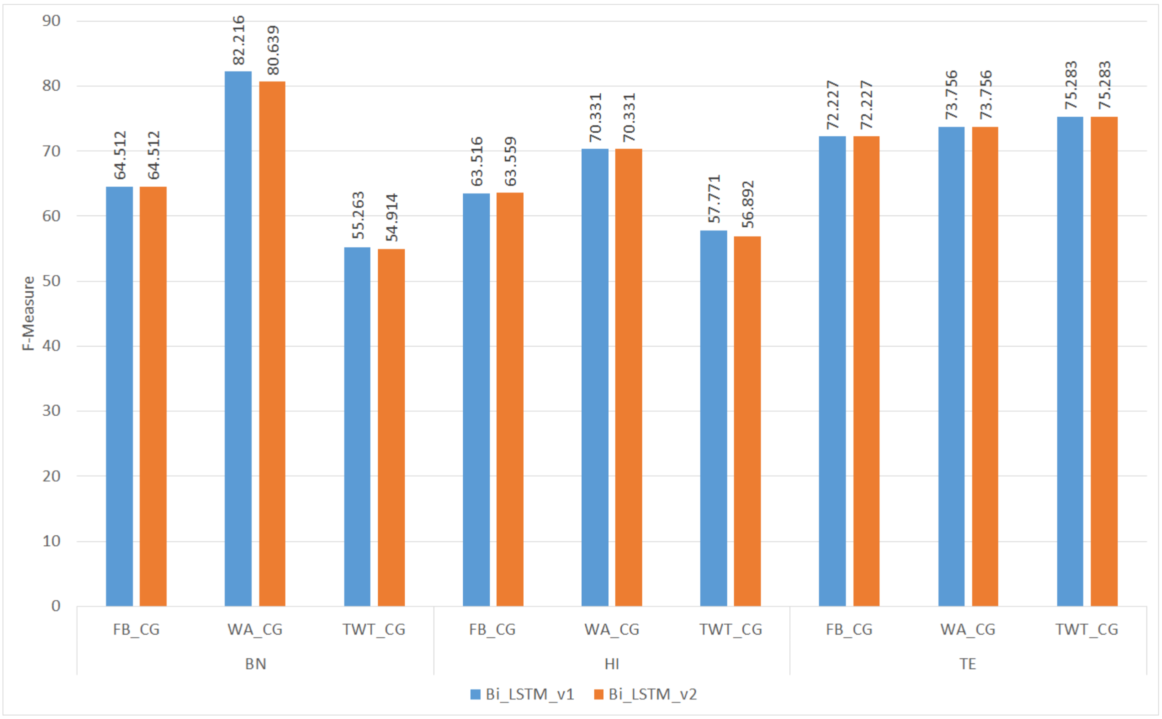


Figure 7.8: F-measure for Coarse Grained POS Tagging using approach 3

4. Tree Based Named Entity Identifier

7.2.1.1 Pre-Processing

The data is pre-processed before detecting named entities. This is done to ensure that the data is uniform and the system can benefit from that. String is converted in lowercase. It also removes all links present in the tweets, if any. These pre-processed tweets, along with the original tweets are then passed to the next phase.

7.2.1.2 Number Based Named Entity Recognition

This phase identifies number based entity such as date, time, month, day, year, money, period, quantity, distance and count using a set of regular expressions. Regular expressions are designed based on the common patterns observed in the annotations for these tags. Regular expressions helps in detection for these tags because there are limited variations possible for each of these tags. For example, the tag 'Day' can be only one of the seven possible days of a week in a language. Hence, detecting them using regular expressions will be efficient. While identifying the NE tags, there is a possibility of having multiple

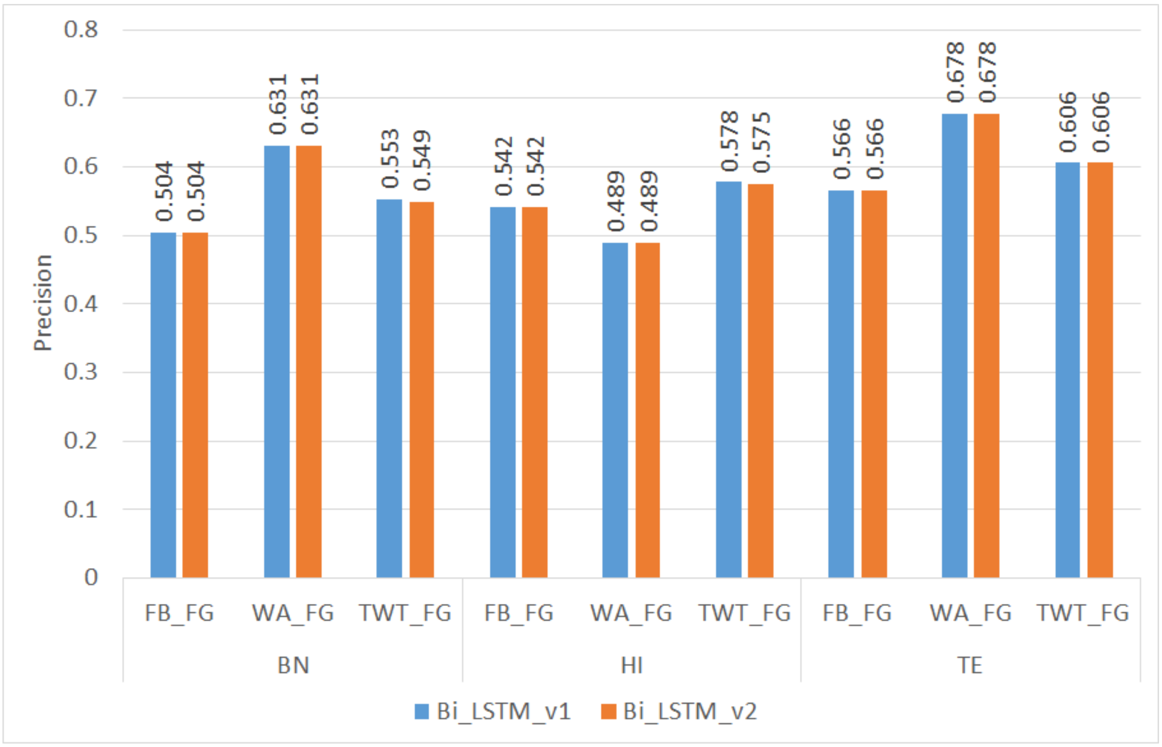


Figure 7.9: Precision for Fine Grained POS Tagging using approach 3

tags attached to the same token. To remove ambiguity, proposed approach checks for tags in a particular predefined order.

7.2.1.3 Gazetteer Based Entity Recognition

As shown in Table 7.15, except *Entertainment*, *Location*, *Person* and *Organization*, rest of the tags contain very less data that cannot be used to train a classifier. Hence, Gazetteer lists are used for identifying NE with insufficient training data. Gazetteer lists are created from the annotations given to the training data. While checking in gazetteer list # and @ symbols are ignored.

7.2.1.4 Classification

The rest of the NE Tags are identified by creating a feature vector for each token of a tweet. These feature vectors are then trained using a decision tree and extremely randomized tree classifier. The features considered for building feature vector are mentioned in Table 7.16. English dictionary feature is used to identify the presence and absence of an English word,

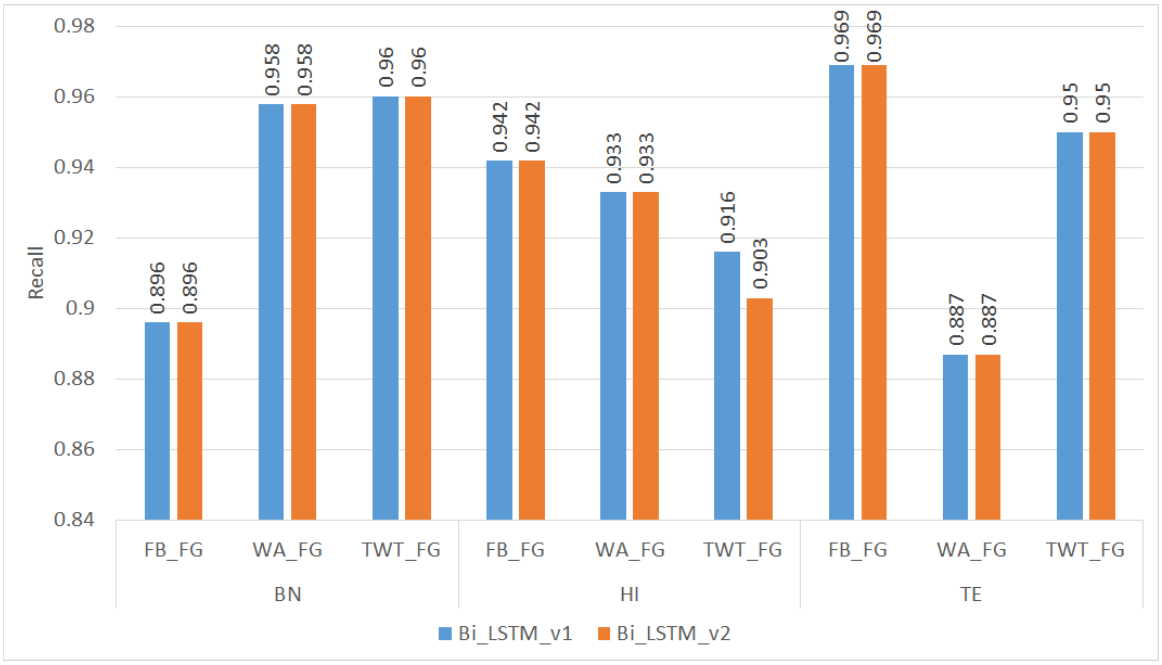


Figure 7.10: Recall for Fine Grained POS Tagging using approach 3

i.e. if it is an English word it is labelled as *1*, else *0*. Python dictionary called pyenchant[1] is used for extracting this feature. Using most common prefixes and suffixes (length = *1 to 3*), dictionaries are built for prefix suffix features. Presence of these prefixes and suffixes in tokens are identified using the same. Gazetteer list feature checks for the presence of the token in gazetteers list of the remaining tags and uses its presence and absence as a feature. *Is Previous token* tag is also taken into account to check the structure of the tweet. Using all features mentioned in Table 7.16, decision tree and extremely randomized trees are trained for classification.

7.2.1.5 Algorithm

Algorithm 7.5 explains the proposed approach for NER of code mixed text. The System first pre-processes the input by removing the website and Twitter links (implemented by: *Link_remover()*) and then converts the tweet into lowercase (implemented by: *Case_conversion()*). In the second phase, all numerical features like date, time, money, quantity, period, distance, day and count are identified using *check_Numerical()*. Before adding it to the final predictions, tweets are re-checked for overlapping tags and removed if found. This process is implemented using *add_without_repetition()*.

[1]http://packages.python.org/pyenchant/

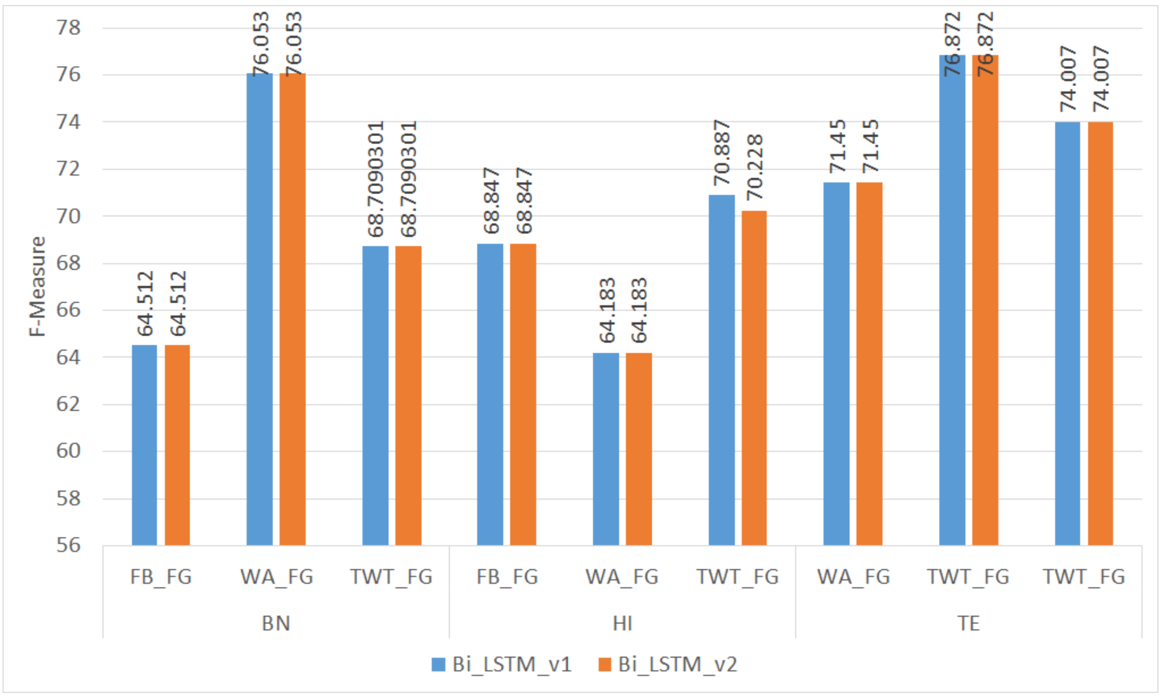


Figure 7.11: F-measure for Fine Grained POS Tagging using approach 3

In the third phase, tweets are tokenized using the function *Tokenize*. These token are then matched to the gazetteer list to check their presence in the list using *check_gazetteer_List()* and add them to the final list of tags of that tweet. In the final phase, feature vectors for each tweet are created and passed for prediction to *clf* classifier using *predict()* function. The classifier *(clf)* uses two variations of classifier, Decision trees and Extremely Randomized trees.

7.2.2 Data Analysis

CMEE-IL 2016 organizers has stated the problem of NER in Hindi-English and Tamil-English code mixed text. CMEE-IL dataset contained two code mix datasets, Tamil-English and Hindi-English. In each dataset, the training data consisted of two files, a text file containing raw tweets along with their tweetID and UserID and another text file containing annotations to the tweets present in the raw tweets file. The raw tweet files consist of *2700* tweets in the Hindi-English corpus and *3200* tweets in the Tamil-English corpus. All the tweets in the Hindi-English corpus are already romanized whereas the Tamil-English corpus has a mixture of both Tamil script and romanized script. There are *21* tags present in the corpus as mentioned in Table 7.15.

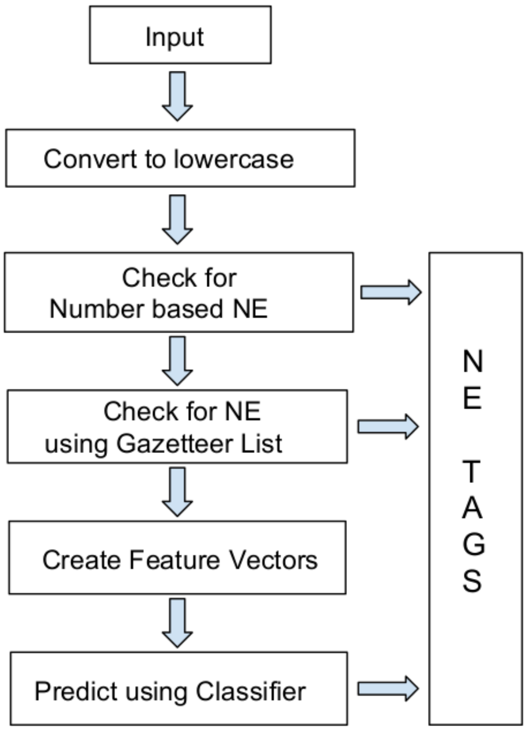


Figure 7.12: Block Diagram for Proposed Algorithm

Named Entity Tag Person, Entertainment and Location occupies the majority of the instances in Tamil-English corpus. Person tag comprises of names of famous actors, actresses, politicians, news reporters and social media celebrities. Entertainment comprises of names of famous TV shows and movies while location consists of names of famous cities, Indian towns and Names of countries. The remaining part of Tamil-English dataset comprises of some numerical and time based tags. These tags include *count, distance, date, money, month, time* and *year*. *Money* represents numbers along with a monetary tag like '15 dollars'. *Organization* is another tag which is associated with the names of organizations. The rest of the tags have very less occurence in the annotated file.

Comparing Hindi-English with Tamil-English, the percentage of minority tags remains almost same. However, in Hindi-English corpus, *entertainment* tag has the highest number of annotations present. It is followed by *person* tag which is close to *entertainment* tag. The rest of order remains the same as that of Tamil-English corpus but with varying percentages.

Table 7.15: Frequency of NE in both datasets

Type of NE	Tamil-English	Hindi-English
Artifact	18	25
Count	94	132
Date	14	33
Disease	5	7
Distance	4	0
Entertainment	260	810
Facilities	23	10
Livthings	16	7
Location	188	194
Locomotive	5	13
Materials	28	24
Money	66	25
Month	25	10
Organization	68	109
Period	53	44
Person	661	712
Plants	3	1
Quantity	0	2
Sday	6	23
Time	18	22
Year	54	143
Total	1609	2346

7.2.3 Experiments & Results

Four versions are created for the proposed approach, for each of the language pair. In all the versions, the numerical feature is detected using the numerical function as explained in section 7.2.1.2. Rest of the tags are classified using different versions created as specified in Table 7.17. The rest of the tags are classified using Gazetteer Based Entity Recognition phase as mentioned in section 7.2.1.3. This is done because less training data is available for few tags such as plants, disease and locomotive, as mentioned in Table 7.15. All the variations for the proposed algorithm is evaluated using F-measure for each language pair. Finally, three systems are compared for each of the language pair. Hindi-English used versions *1, 2 and 4* respectively whereas Tamil-English used versions *1, 3 and 4* respectively.

7.2.3.1 Evaluation & Discussion

As shown in Table 7.18 and 7.19, version *3* performed well for Hindi-English and Tamil-English. Based on the F-measure, it can be concluded that algorithm with more number

Table 7.16: Features used for creating the feature vector.

Sno	Features
1	Presence of token in English dictionary
2	Prefixes of length 1 to 3
3	Suffixes of length 1 to 3
4	Capitalization related features like starting letter capital, all letters capital, other letters capital.
5	Features based on presence or absence of special characters like #, @, numbers, other symbols.
6	Presence of emoticons
7	Token present in gazetteer list.
8	Is previous token a NE Tag.

Algorithm 7.5: Algorithm for Identifying Named Entity Recognition for Code Mixed Text in Indian Language

```
input  : Code-Mixed tweets list , S
output: Predicted Named Entity Labels, P
1  initialization: P=[]
2  for $i = 0$ to $S.length$ do
3      Link_remover(S[i]);
4      Case_conversion(S[i]);
5  for $i = 0$ to $S.length$ do
6      $d = check\_Numerical(S[i])$;
7      $add\_without\_repetition(d)$;
8      $tok = Tokenize(S[i])$;
9      for $j = 0$ to $tok.length$ do
10         $g = check\_gazetteer\_List(tok[j])$;
11         add_without_repetition(P , g );
12         f = Create_Feature_vector(tok[ j ]);
13         c=clf.predict(f);
14         add_without_repetition(P,c);
```

Table 7.17: Different Versions of Proposed System

Version	Tags trained on Classifier	Classifier Used
1	Person, Entertainment, Location, Organization	Decision Tree
2	Person, Entertainment, Location, Organization	Extremely Randomized Tree
3	Person, Entertainment, Location, Organization, Artifact, Facilities	Decision Tree
4	Person, Entertainment, Location, Organization, Artifact, Facilities	Extremely Randomized Tree

of gazetteer lists and extremely randomized forest (version 2) performed well in case of Hindi- English. However, in case of Tamil-English, an algorithm with less number of gazetteer lists and decision tree (version 3) proved to be effective.

The precision value could be less because of string matching with elements of the gazetteer lists. It can also be observed that recall value is low for all the versions, this could be due to less number of named entites for a sentence, which in turn reduces the average recall value. The recall might have increased if the partial identification of NE is considered. The proposed system is compared with other existing approaches as shown in Figure 7.13 & 7.14.

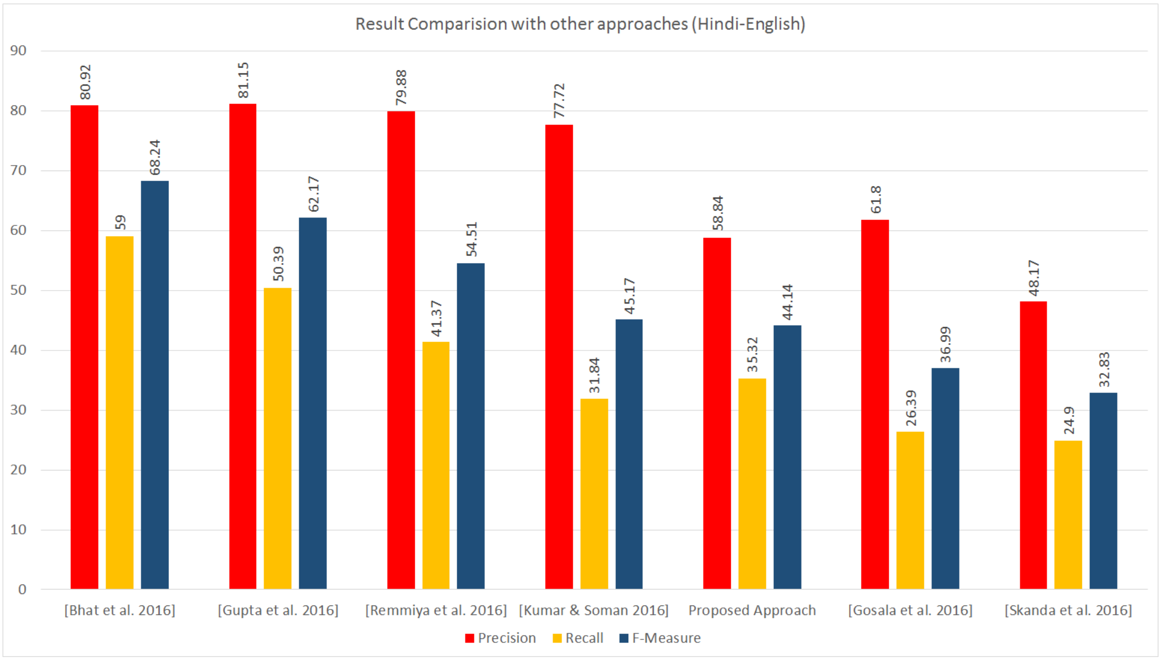


Figure 7.13: Result comparison with other approaches (Hindi-English)

Table 7.18: Results for Hindi-English Proposed System

Version	Precision	Recall	F-measure
1	58.66	32.93	42.18
2	58.84	35.32	44.14
4	59.15	34.62	43.68

Table 7.19: Results for Tamil-English Proposed System

Version	Precision	Recall	F-measure
1	55.86	10.87	18.20
3	58.71	12.21	20.22
4	58.94	11.94	19.86

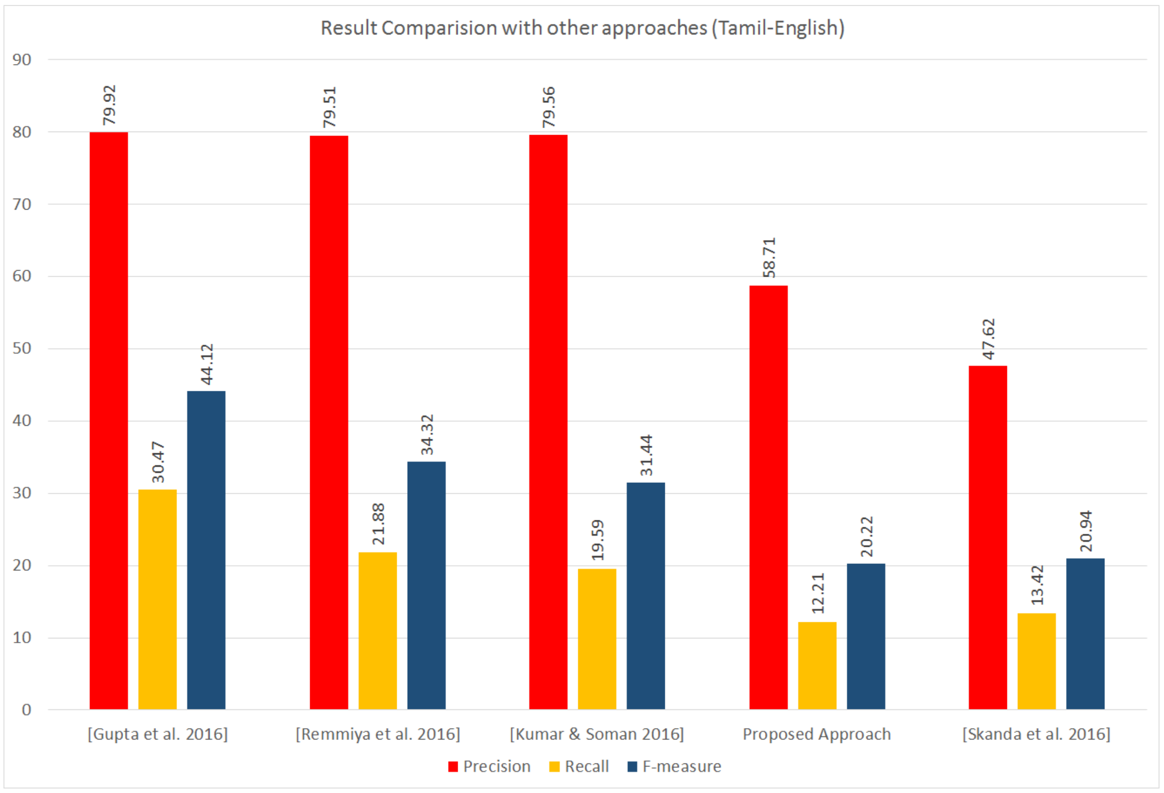


Figure 7.14: Result comparison with other approaches (Tamil-English)

7.2.3.2 Error Analysis

Few phases in proposed approach might have attributed to misclassification for few tags. One such phase can be Gazetteer Based Entity Recognition phase of the proposed approach which is a dictionary based approach and has disadvantages associated with it. If the data present in the dictionary is very less, it will correspond to lower precision. Hence, there is a need for increasing more elements in the list for a better recognition system. Apart from this, if there is an ambiguity in tags, then there is a chance of misclassification. For example, if there is a token 'Honey' it can represent a *Person* like 'Honey Singh' or as a tag *Material*. Problem of ambiguity can be resolved using a classifier.

7.3 Concluding Remarks

In this chapter, tools such as POS tagger, NER for code mix text has been proposed. Three approaches are proposed for POS tagging. Two of them are based on traditional machine learning techniques such as random forest, extremely randomized tree and ensemble method. The system is designed to tackle code mixed scripts for three language pairs Telugu-English, Hindi-English and Bengali-English. In future, more features can be added to improve the precision without affecting the recall. The proposed system did not use clustering, adding it might help in improving the accuracy of the system. WordNet can also be used to improve the system by clustering the words to its synonyms. Third proposed algorithm for POS tagger of code mixed data used recurrent neural networks for tagging. The technique is superior to conventional machine learning methods as no feature engineering is required but a lot of data is required to improve the results further. Another tool proposed for code mixed text is Named Entity Recognizer. A hybrid approach of a dictionary and supervised classification approach is proposed for identifying entities in code mix text of Indian languages such as Hindi-English and Tamil-English. The proposed system used a pipelined approach to identify the named entities. There are four variants of the system based on the number of tags and the classifier used.

Chapter 8

Code Mixing Applications

With the increasing amount of code mixed data on social media has hindered the automatic analysis of the text. Applications such sentiment analysis, question classification, recommendation system and many other analysis tools use social media text to aid the process. However, to automate the process, they have address the challenges posed by social media text due to informal nature of the text. One such challenge in automated processing of text is of code mixing. This chapter focuses on applications of code mixed text i.e question classification and sentiment analysis. Section 8.1 and 8.2 discusses the applications of code mix text for Indian languages. In section 8.1, multiple algorithms for question classification are proposed and their evaluation is discussed. Algorithms for sentiment analysis in code mixed text has been discussed in section 8.2. To understand a code mix text, it is essential to analyze and identify the language at word level, and hence an algorithm has been proposed to identify the same to aid the process of sentiment analysis.

8.1 Question Classification for Code-Mixed Cross Script Question

Nowadays tasks ranging from shopping to medical consultation are performed by chatbots. Intelligent home automation systems are in the market which can answer user questions, place online orders and perform a myriad of other functions. However, these question answering frameworks are almost exclusively monolingual. The most basic mechanism in these frameworks is identifying the type of question posed by the user as this reduces the scope of search for answer by the system. For example, the question "Which continent does India belong to?" should be classified as 'Location', as the answer to this question is of type 'Location'. A lot of existing question answering frameworks contain rules that have been manually built to map the question to its type, but these frameworks are not easy to update and maintain.

The interaction using question answering has been researched well in the monolingual scenario with good results, but not much progress has been made in the case of code mixed scripts. This can be attributed to the fact that there is no pre-defined standard for writing spellings in the non-native script. Another reason is that there are no grammatical/syntax rules that are followed by code mixed scripts. Question classification helps in diminishing the number of candidate answers and furthermore can be utilized to decide the viable answer. Being a classic application of NLP, Question Answering (QA) has practical applications in various domains such as education, health care and personal assistance. QA is a retrieval task which is more challenging than the task of the standard search engine because the purpose of QA is to find the accurate and concise answer to a question rather than just retrieving relevant documents containing the answer [Li & Roth 2002]. Recently, [Banerjee *et al.* 2016b] formally introduced the code-mixed cross-script QA problem. The first step of understanding the question is to perform a question analysis. Question classification is an essential task of question analysis which detects the type of answer expected to the question. Question classification helps not only to filter out a wide range of candidate answers but also determine answer selection strategies [Li & Roth 2002]. Furthermore, it has been observed that the performance of question classification has a significant influence on the overall performance of a QA system.

This section addresses the task of code mixed cross script question classification where *'Q'* represents set of factoid questions written in romanized Bengali along with English. The task is to classify each given question into one of the predefined coarse-grained classes. Two approaches have been proposed for classification of questions in this section. Initially proposed approach used traditional machine learning classifiers like NB, RF and LR. Four versions have been proposed for the first approach. The first version uses translation before NER, the second version uses only NER followed by classifier algorithms, the third version uses only translation followed by classifier algorithms, and the fourth version uses translation after NER.

The second approach uses CNN for classification of the questions. Convolution can be thought of as a sliding window function being applied to a matrix. CNN(s) are fundamentally a few layers of convolutions with non-linear activation functions like *ReLU* or *tanh* connected to the outcomes. In a conventional feed forward neural system, every input neuron is associated to yield a neuron in the following layer. Instead in CNN(s), the output layer is computed by using convolutions over the input layer. This results in local associations, where every region of the input is associated with a neuron in the output. Every layer applies different functions, commonly hundreds or thousands, and combines their outcomes. During the training phase, depending on the input a CNN automatically learns the values of its filters. CNN(s) have been used extensively in the past for image classification and are the core of most computer vision systems today, for example, Facebook's automated tagging feature. More recently CNN(s) are being used for NLP tasks like classification.

8.1.1 Approach 1: Question classification using traditional machine learning techniques

A word-level *n*-gram based approach has been proposed to classify code mixed cross script questions into nine different coarse-grained question type classes. The approach can be broadly divided into four phases.

1. Preprocessing

   (a) Separation of class labels from training dataset.

   In the dataset, the questions are labelled with the class they belong to. The sentences are separated from their class type to build a feature vector for the questions.

   (b) Case Conversion.

   To normalize the text, case conversion to lower case is performed.

2. Named Entity Recognition and removal

   With the end goal of classification of the questions into one of the classes, the presence of these named entities can be insignificant, as these elements may not contribute in building question structure for determining the class type. The preprocessed dataset contains questions which have a lot of Named Entities. Predefined categories can allude to NE, for example, names of people, currency names, instances of time and locations. Named Entities are recognized utilizing a dictionary-based approach. The information set used for NER mostly contained the passages from FIRE 2015 sub-task1's dataset [Sequiera *et al.* 2015b].

3. Translation

   In this phase, the romanized Bengali words are transliterated into their native scripts. The words are further translated into their corresponding English counterpart by utilizing the Google Translate API[1]. This phase helped to build a monolingual dataset from the code mixed script dataset for efficient classification. For instance, the question record "Phulera janya kata?" and the record "Phulera how much?", both allude to a similar question. These sentences utilize a diverse mix of words, and thus normalizing sentence to its English interpretation, would possibly prompt to an expansion in the precision.

4. Classification

   The input to the classifier is different depending upon the dataset used. In the first version, translation is performed followed by Named Entity removal. In the second version, only Named Entity removal is performed, and in the third version, translation is performed without Named Entity removal. The orders and phases used by the versions can be inferred from Table 8.1. *n*-grams are used to make feature vectors corresponding to each question in the dataset. The value of *n* lies from *2 to 4*. The transposed matrix of these feature vectors along with the numerically encoded class label matrix is then used as input to the classifiers. The following different classifiers are used:

[1]https://translate.google.co.in/

(a) Gaussian Naive Bayes Classifiers

(b) Logistic Regression Classifier

(c) Random Forest Classifier with Random State = 1

Four different versions are created using above mentioned phases as shown in Table 8.1. Different versions are created with an attempt to validate which phase in the pipeline contributes, most to the results.

Table 8.1: Description of versions for Approach 1

Version Number	Phases Used in respective order
Version 1	Preprocessing, Translation, Named Entity removal, Classification
Version 2	Preprocessing, Translation, Classification
Version 3	Preprocessing, Named Entity removal, Classification
Version 4 [Bhargava *et al.* 2016b]	Preprocessing, Named Entity removal, Translation, Classification

8.1.2 Approach 2: Question classification using Deep Learning

In this proposed approach, CNN has been used for question classification into nine different classes.

1. Preprocessing

   The steps involved in preprocessing are:

   (a) Class labels and sentences are separated. Stemming and case conversion is performed to normalize the text.

   (b) Pad each sentence to the maximum sentence length, which is *11* in case of the dataset [Banerjee *et al.* 2016a]. <PAD >tokens are appended to sentences with length lesser than *11* words. Padding sentences to the same length is useful because it allows to efficiently batch data.

   (c) Map each word to an integer between *0* and vocabulary size by building a vocabulary index. Each sentence can now be represented as a vector of integers.

2. Deep Learning Model

Let, $a_i \in R^k$ be the k-dimensional word vector corresponding to the i^{th} word in the sentence. A sentence of length n comprises of the words a_1, a_2, and so on till a_n. Let $a_{i:i+j}$ allude to the window of words a_i,a_{i+1}, . . . ,a_{i+j}. A convolution operation involves a filter w which is applied to a window of h words to create another element. For example, an element g_i is created from a window of words $a_{i:i+h1}$ by $g_i = f(w * a_{i:i+h1} + b)$. Here $b \in R$ is a bias term, and f is a non-linear function like *ReLU* or *tanh* function. This filter is applied to every conceivable window of words in the sentence $a_{1:h}$, $a_{2:h+1}$, . . . , $a_{nh+1:n}$ to deliver a feature map = $[g_1, g_2, ..., g_{nh+1}]$, with $c \in R^{nh+1}$.

A maximum pooling operation is then applied over the feature map and the max value of *max{g}* is then selected as the feature, corresponding to this filter. The idea behind this is to capture the most imperative component of every feature map. The model uses numerous channels (with shifting window sizes) to obtain multiple features. These features are accumulated in the last but one layer. The accumulated features are then passed to the last softmax layer which gives the output. The output is the probability distribution over the class labels.

The convolutional layer's parameters comprise of an arrangement of filters that can be learned. Even though each filter is small in scope, it reaches out through the full depth of the information volume. In the forward pass, filter slides (convolves) over every channel in the input and computes the dot products between the sections of the filters and the input. The filter sliding over the input produces an activation map that represents the relationship between filter response and spatial position. The system learns filters that are activated when some feature is seen, for example, the words like 'fee', 'charge' and 'taka', that relate to one class label.

After convolutional layer, pooling is done to ensure that over-fitting does not occur and to reduce the amount of computation and parameters in the network. The model built for approach *2* is represented by Figure 8.1. In the model built for approach *2*, the primary layers insert words into low-dimensional vectors. The following layer performs convolutions over the inserted word vectors utilizing different filter sizes. For instance, sliding more than *3, 4 or 5* words at once. Next, the result

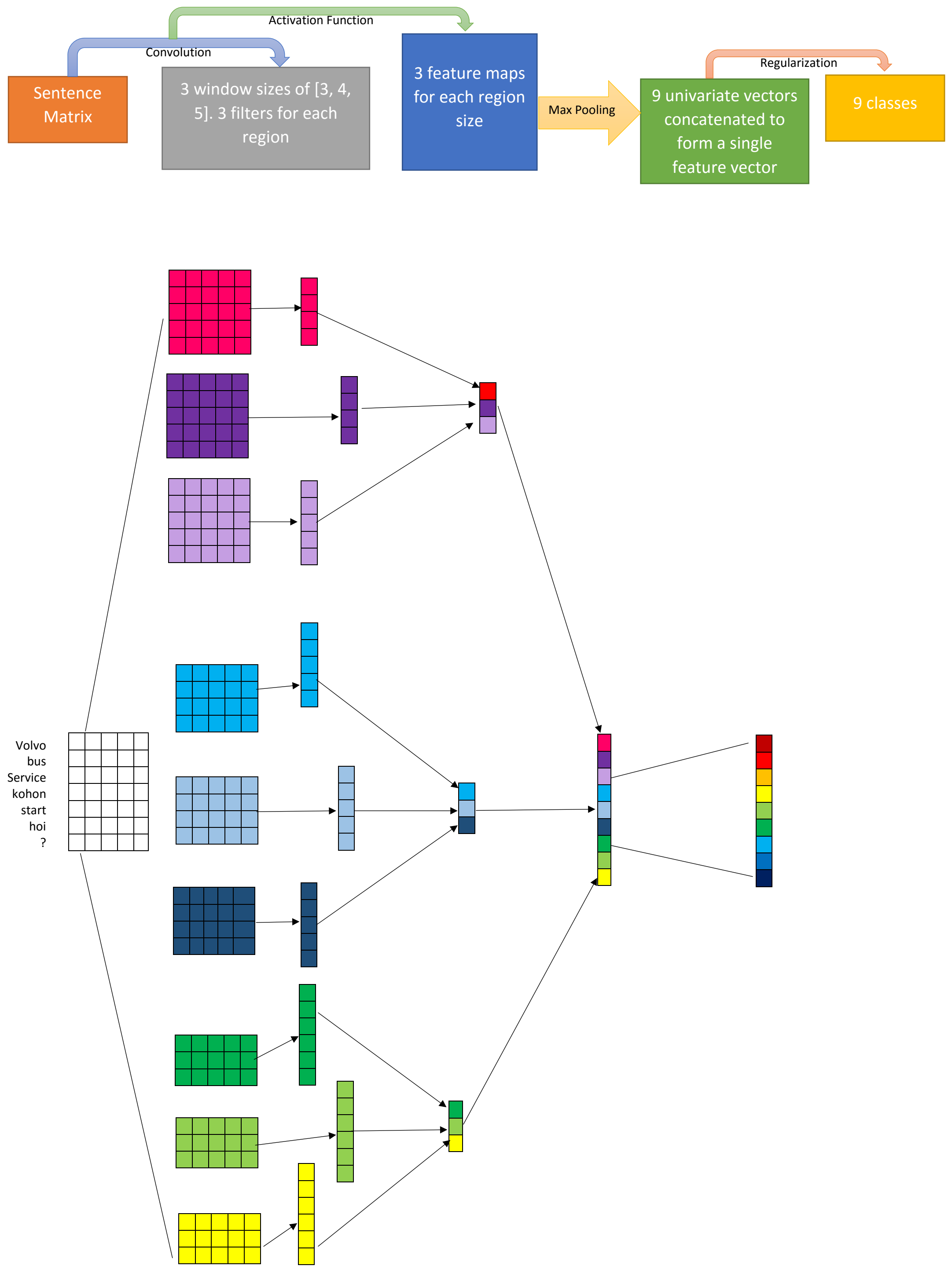

Activation Function
Convolution
Sentence Matrix
3 window sizes of [3, 4, 5]. 3 filters for each region
3 feature maps for each region size
Max Pooling
Regularization
9 univariate vectors concatenated to form a single feature vector
9 classes
Volvo
bus
Service
kohon
start
hoi
?

Figure 8.1: Convolutional Neural Network Model for Approach 2

of the convolutional layer is max-pooled into a long feature vector, which includes dropout regularization. Classification is done by using the final softmax layer.

8.1.3 Algorithms for Question Classification for Code Mixed Cross Script Questions

Algorithm 8.1 elaborates the traditional machine learning approach for question classification. The input contains the questions along with the class labels. First, preprocessing is performed where the class labels and corresponding questions are separated. This is implemented by the function, *Label_Separation()*. Case conversion is done to normalize the input data by the function, *Case_Conversion()*.

To build the feature vectors, *n*-gram technique is applied. The function *Count_Vectorizer()* converts the text into *n*-gram tokens, where *n* ranges from *2* to *4*. *Analyzer()* function produces n-gram tokens when called on each row of the dataset. The word level *n*-grams corresponding to each row is appended (using function *append()*) to the *n*-gram list. The class labels are enumerated with the help of *Encode_Class()* function. Feature vectors for these classes are generated using the encoded values.

In the final step, the two feature vectors built in the previous steps are passed to the classifier. In approach 1, explained in section 8.1.1, three classification algorithms, namely gaussian naive bayes, logistic regression and random forest have been used. The classifier predicts the class labels for the questions, generated as output.

In algorithm 8.2, CNN is initialized with the following hyperparameters. The sequence length is *11* which is the maximum length of the entry in the NER dataset. The number of classes are nine. The embedding size is set to the default value of *128*. *filter_size* refers to the number of words required in convolutional filters. In approach 2, mentioned in section 8.1.2, the value of the filter size lies in the range *[3, 5]*, which implies that the filters slide over *3, 4 and 5* words respectively. The function *CreateEmbedding()* defines the embedding layer which maps vocabulary word indices to vector representations. This is similar to creating a look-up table. Embedding matrix *'W'* is learned during the training. Since the filters are of different sizes and each convolution produces vectors of different shapes, a layer is created for each of them, and the results are then merged into one feature vector. Once the output vectors are pooled from each filter size, they are combined to give the final vector. The vector from max-pooling is used to generate predictions by performing matrix multiplication and then choosing the class with the maximum score.

Algorithm 8.1: Question Classification using traditional machine learning

```
input  : Questions code mixed in Bengali and English, Q.
         Training class labels T.
output: An output sequence P, the predicted class labels.
1  Initialization: $n$-grams = [];
2  for $i \leftarrow 1$ to Q.length do
3      Label_Separation(Q[i]);
4      Case_Conversion(Q[i]);
5      if version1 then
6          Translate(Q[i]);
7          NER(Q[i]);
8      else if version2 then
9          Translate(Q[i]);
10     else if version3 then
11         NER(Q[i]);
12     else if version4 then
13         NER(Q[i]);
14         Translate(Q[i]);
15 Vectorizer = Count_Vectorizer(ngram range=(2,4));
16 Analyzer = Vectorizer.Build_Analyzer();
17 for $i \leftarrow 0$ to Q.length do
18     row=Analyzer(Q[i]);
19     for $j \leftarrow 0$ to row.length do
20         n-grams.append(row[j]);
21 Matrix_Data = Create_Feature_Vector(n-grams);
22 Class_List = Encode_Class(T);
23 Matrix_Class = Create_Feature_Vector(Class_List);
24 clf = Classifier(Matrix_Data, Matrix_Class);
25 clf.fit(Matrix_Data, Matrix_Class);
26 P = clf.predict(Matrix_Test);
```

8.1.4 Data Description and Analysis

The dataset [Banerjee *et al.* 2016a] used for training purpose consisted of 330 questions. The dataset contains code mixed text of Bengali and English with nine question classes. The number of words in the dataset varied from *2* to *11*. On average, question has an approximately *6* words. As shown in Figure 8.2, class-types 'ORG' and 'TEMP', comprises

Algorithm 8.2: Question Classification using Deep Learning.

```
input  : Questions code mixed in Bengali and English, X.
         Corresponding training class labels Y.
output: An output sequence P, the predicted class labels.
1  Initialize the CNN with sequence_length=11, num_classes =
   9, embedding_size = 128, filter_sizes = [3,4,5], num_filters
   = 3;
2  Initialize a matrix W with a random distribution.;
3  Initialization: pooled_outputs=[];
4  CreateEmbedding(W, X);
5  for i in filter_sizes.length do
6      Initialize filter matrix F, bias matrix b ;
7      conv = CreateConvolutionalLayer(F,b);
8      h = Non-linearity(conv,b);
9      pooled_outputs = Max-pool(h);
10 Combine pooled features(pooled);
11 Prediction = max(W*Feature_Vector);
```

the majority of the instances. Each of these classes represents a particular type of question related to specific entities. These classes help in reducing the scope that automated system has to look at when searching for an answer. For example, Class type 'MNY' stands for Money related questions and the instances comprises of words like 'fare', 'price' and helping words like 'koto' (bn) and "how much". Class type 'PER' stands for person related questions mostly comprising of words like 'who' and 'whom' implying for the subject of the sentence being a person. Class type 'TEMP' implies time related questions mainly comprising of words like 'when' and 'at'. Class type 'OBJ' stands for the entity/object implying that subject of the sentence is an entity and mainly comprising of words like 'what' and 'kon'. Class type 'NUM' stands for numeric entity related questions and mainly involves usage of words like 'how many' and 'koto'. Class type 'DIST' stands for distance and implies that question is related to the distance between places. Class type 'LOC' stands for location and thereby mainly comprises of words like 'where' and 'jabe'. Class type 'ORG' stands for organization and relates to questions centred on a particular organization, team or any other group of people and these questions mainly comprises of words like 'which', 'what' and 'team'. Class type 'MISC' stands for miscellaneous, this class has the minimum representation in the dataset and relates to a variety of questions. The entire dataset has sentences in a code-mixed format, consisting of words which are

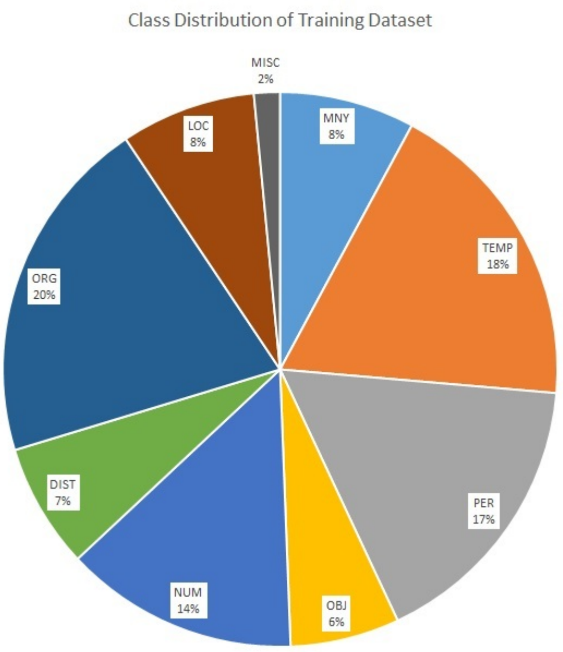


Figure 8.2: Class Distribution of Training dataset

either in Bengali or English dialect. The dataset does not contain any code-mixing at word level. There are no punctuations in the dataset except that of the question mark (?). The dataset has a considerable amount of NE in both English and Bengali language. A detailed distribution of classes and its count for code mixed question classification task is mentioned in Table 8.2.

Table 8.2: Distribution of classes in the dataset.

Class	Number of instances	Proportion
ORG	67	20%
TEMP	61	18%
PER	55	17%
NUM	45	14%
MNY	26	8%
LOC	26	8%
DIST	24	7%
OBJ	67	6%
MISC	5	2%
Total	330	100%

8.1.5 Experiments

Two approaches are proposed for question classification. The first approach used traditional machine learning classifiers, and four versions are created. All four variations of Approach 1 are different from each other in terms of classifiers used (Gaussian Naive Bayes, Logistic Regression and Random Forest Classifiers). For example, variations of fourth version are named as 4.1, 4.2 and 4.3. For version 4.1 (gaussian naive bayes classifier), an accuracy of *81.12%* is obtained, version 4.2 (using logistic regression), an accuracy of *80%* is obtained and version 4.3 (random forest classifier), an accuracy of *72.78%* is obtained. All the versions performed comparably, albeit version *1* and version *2* work better than version *3*. Approach *2* performs better than approach *1* for few labels and overall comparable with approach *1*.

8.1.5.1 Evaluation and Discussion

A comparison of accuracy is performed with existing work as shown in Figure 8.3.Version *4.1* achieved a second highest accuracy of *81.12%* while the highest accuracy achieved is *83.34%* by [Bhattacharjee & Bhattacharya 2016]. Choice of gaussian naive bayes classifier leads to the maximum accuracy attainment, as the proposed algorithm deals with the problem involving continuous attributes. Usage of naive bayes helps in building simplistic and highly scalable models, which are fast and scale linearly with number of predictors and rows. Moreover, the process of building a naive bayes model is highly parallelized, even at the level of scoring. It is also observed from the results, that the proposed algorithm generated highest F-measure scores for the classes of 'ORG', 'MNY' and 'MISC'. Figure 8.4 shows the comparison of the F-measure obtained for the class organization with other existing approaches. The proposed algorithm (version 4.1) got the highest scores of *0.74418* using gaussian naive bayes approach. This implies that the questions mainly being framed with words like 'which' and 'what', related to the organization could be efficiently classified. These scores can be attributed to the fact that the instances of the class 'ORG' are maximum in the dataset (*67* out of *330*). Proposed algorithm involves the formation of word level *n*-grams due to which words and phrases like 'which', 'team', 'series' and 'sponsor', got associated, and thus might have contribute to an increase in score.

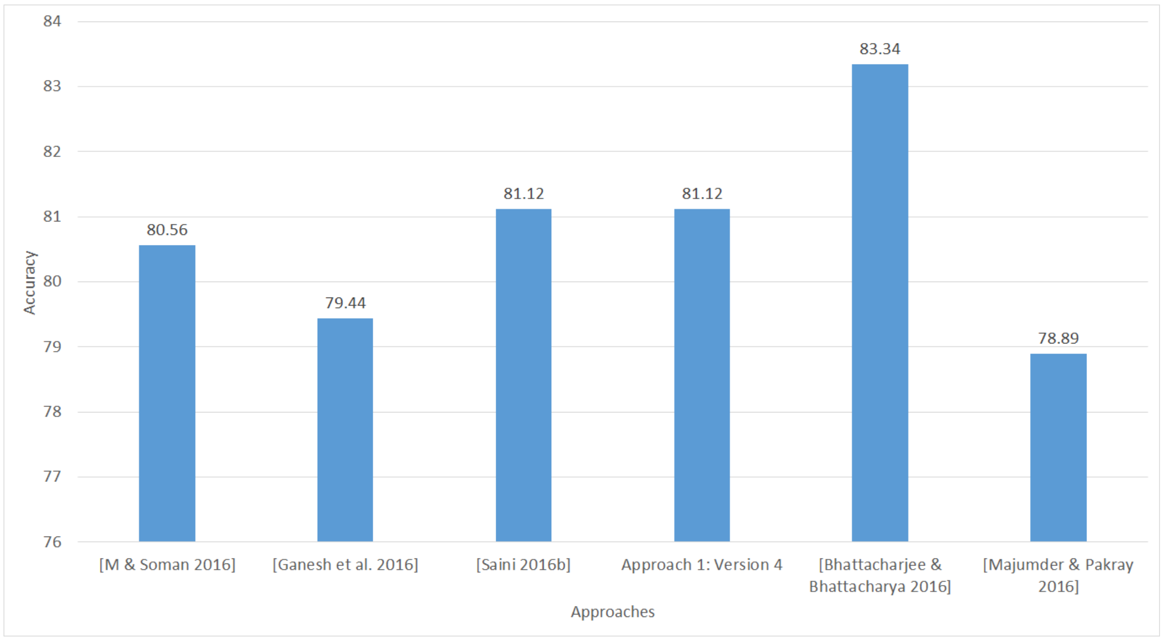


Figure 8.3: Comparison of accuracy with existing work and proposed approach

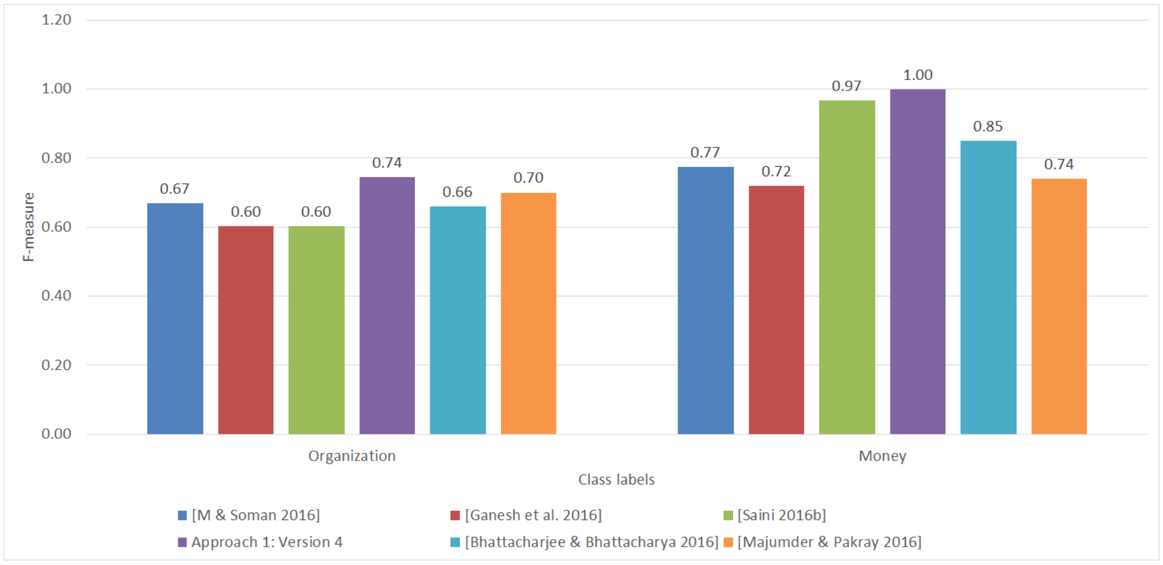


Figure 8.4: Comparison of F-measure for Organization and Money class with existing work and proposed approach

Figure 8.4 also shows the comparison of F-measure obtained for the class 'MNY' with other existing approaches. Version *4.2* achieved the highest scores of *1* and hence, all the questions relating to money, being framed with words like "how much", 'price' and 'fare', could be efficiently classified. These high F-measures could be attributed to the efficient deployment of the word level *n*-gram techniques which in a way linked the words like 'fare', 'how', 'much' and 'price', and thus might contributed to an increase in accuracy. The evaluated results also show that only two approaches (Version 4 of approach 1 and [Majumder & Pakray 2016]) are able to identify instances belonging to 'MISC' class. This may be because there are only *5* out of *330* instances of 'MISC' class in the training dataset. The proposed approach (version 4.1) got the highest scores of *0.2*, which may be because of the simplistic approach of gaussian naive bayes classifiers and the efficient deployment of the word level *n*-gram technique. Figure 8.5 shows a comparison of accuracy obtained

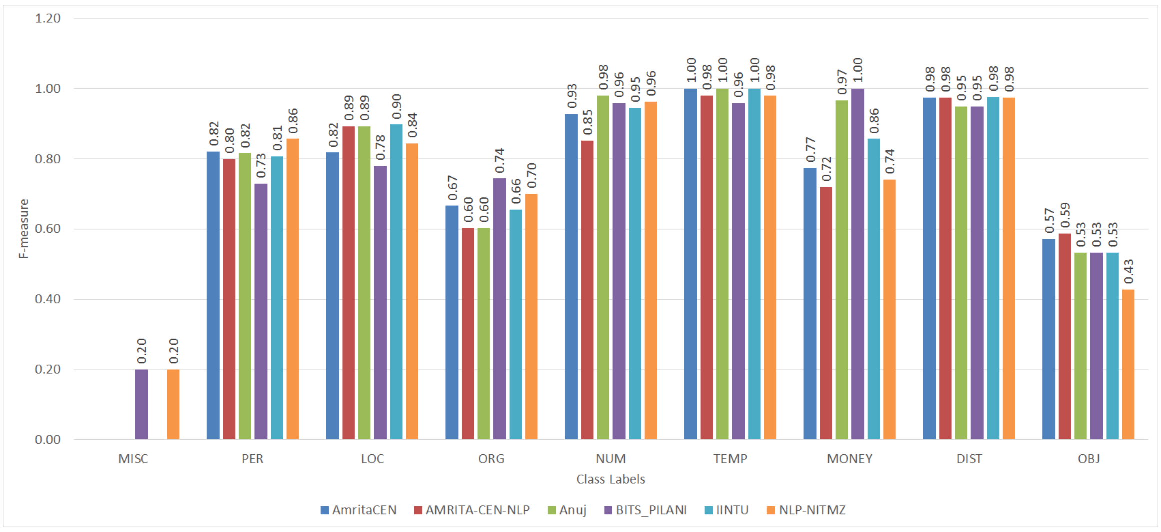


Figure 8.5: Comparison of F-measure with existing work and proposed approach for various classes

for classifying each of the nine classes. As evident from the Figure 8.5, the proposed approach (version 4) works well in identifying the correct class labels particularly in the cases of 'MISC', 'ORG', 'MNY', 'NUM' and 'OBJ' classes with an F-measure of *1* obtained for the class 'MNY'. Table 8.3 shows the scores of precision, recall and F-measure for each of the nine different classes, for the proposed algorithm (version 4 and its variations). The results are further evaluated on the gold dataset for the MSIR sub-task1 in FIRE 2016 [Banerjee *et al.* 2016a]. All the variants of approach *1* and approach *2* are compared to an-

Table 8.3: Class wise score for all the runs submitted

Classes	Scores	Runs		
		version 4.1	version 4.2	version 4.3
PER	Precision	0.574	0.52	0.456
	Recall	1	0.963	0.963
	F-measure	0.73	0.675	0.619
LOC	Precision	0.8	0.889	0.889
	Recall	0.695	0.695	0.695
	F-measure	0.744	0.780	0.780
ORG	Precision	0.842	0.8	0.933
	Recall	0.667	0.667	0.583
	F-measure	0.744	0.727	0.718
NUM	Precision	1	0.897	0.684
	Recall	0.923	1	1
	F-measure	0.96	0.945	0.8125
TEMP	Precision	0.92	0.96	0.954
	Recall	0.92	0.96	0.84
	F-measure	0.92	0.96	0.894
MONEY	Precision	0.889	1	1
	Recall	1	1	0.875
	F-measure	0.941	1	0.933
DIST	Precision	1	1	1
	Recall	0.904	0.81	0.476
	F-measure	0.95	0.895	0.645
OBJ	Precision	0.667	0.75	0.8
	Recall	0.4	0.3	0.4
	F-measure	0.5	0.429	0.533
MISC	Precision	0.5	0	0
	Recall	0.125	0	0
	F-measure	0.2	NA	NA

alyze the results. It can be seen from Figure 8.6, F-measure for version *1*, version *2* and *4* is comparable for class 'PER' whereas approach *2* outperforms the rest. The class 'MNY' (Figure 8.7) has distinctive words in the questions. For example, the words 'fee' (en), 'charge' (en), 'taka' (bn), 'khoroch' (bn), appear only in questions that belong to 'MNY' class. For the class 'NUM', version *2* naive bayes approach performs the best. As can be inferred from Figure 8.8, approach *2* achieves an F-measure of *0.95*. The class 'LOC' has the best score from approach *2*. For the class 'LOC' approach *2* and approach *1* version *2* give comparable results with the F-measure score as *0.8* (Figure 8.9). For both 'PER' and 'LOC' classes, approach *2* outperforms the rest because of the ability of the filters to learn the spatial patterns of the words in the questions belonging to this class. Another factor is that the training dataset has the most number of example queries corresponding to this class.

For class 'DIST', approach *2* and approach *1* version *1.1* and version *2.1* are comparable, with F-measure of *0.95*, as can be seen from Figure 8.10. Version *1.2* and version *2.2* gives F-measure of *0.96* for the class 'TEMP' (Figure 8.11). In the case of class 'OBJ' random forest classifier of version *1* and version *2* give the best F-measure of around *0.53*. Naive bayes classifier also gives comparable performance to random forest classifier for the class label 'OBJ'(Figure 8.12). For the class 'MISC'(Figure 8.14) owing to the lesser number of training instances only version *2* naive bayes classifier can predict one instance in the test set. Overall results of version *1* are comparable to the results of version *4*. In version *1*, a pipeline with translation followed by named entity removal is proposed as opposed to the pipeline with named entity removal followed by translation proposed by [Bhargava *et al.* 2016b].

As can be observed from Table 8.4, versions using Gausian Naive Bayes classifier and approach *2* results in maximum F-measure. Naive bayes is helpful in building scalable models, and the model scales linearly with the number of classes in a comparably faster way. The Bengali word 'ki' appears *22* times in the training data out of which the class label for the corresponding question is 'ORG' *9* times, 'OBJ' *7* times, and 'PER' *4* times. Out of the *9* times, the word 'ki' appears in the question instances of the class 'ORG' *6* times the question ends with the word 'ki'. Approach *2* can exploit all such patterns that exist in the data, provided that there are sufficient number of such instances.

Figure 8.15 shows the comparison between all versions of approach *1* and approach *2*. F-measure of approach *1* version *2* is very close to approach *1* version *4*. This is understandable as named entity removal followed by translation should be no different from translation followed by named entity removal. For the classes 'ORG' and 'DIST', approach *2* outperforms the rest. This is because neural network can learn the patterns for each class.

In approach *1*, version *3*, where only named entity removal is performed, the performance is least among all proposed approaches. This can be because there is no normalization of the words in the text. For example, the words 'chalu' and 'start' mean the same, but are considered different by the classifier and thus the accuracy is reduced. Approach *1*, version *2* performed better in most cases than version *1* and version *4*. This might be because the named entities do contribute in differentiating between the classes. Table 8.4 shows the scores of precision, recall and F-measure for each of the nine different classes, for the proposed approaches, approach *1* and approach *2*.

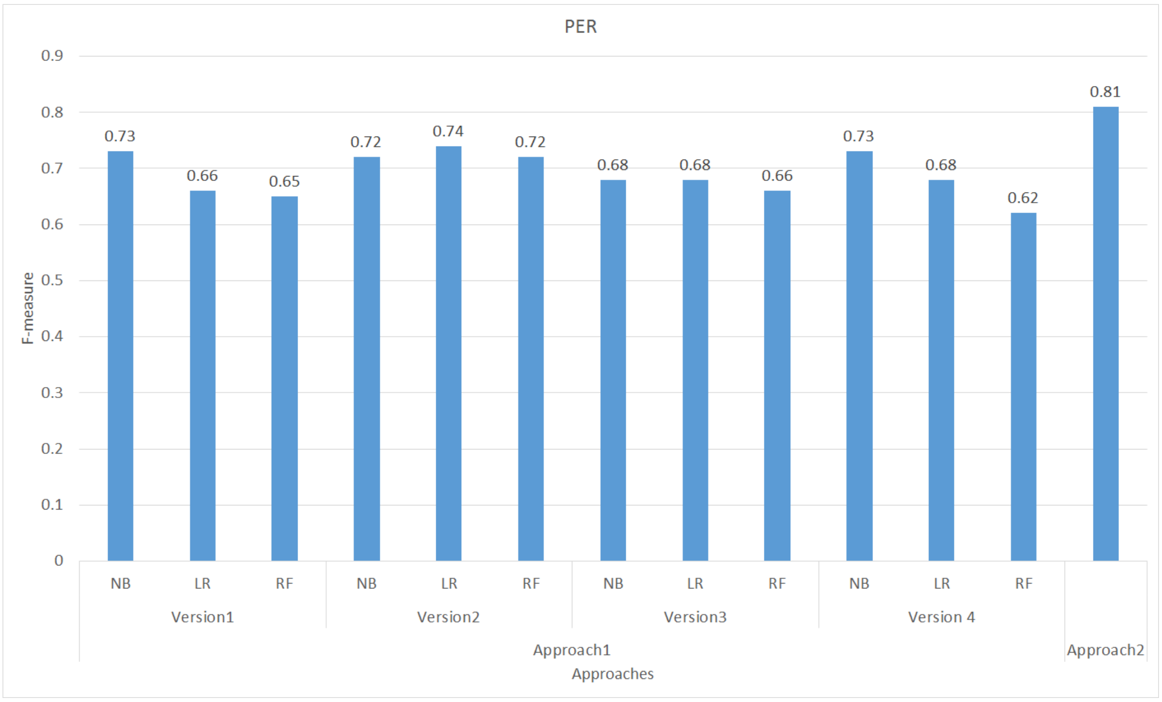


Figure 8.6: F-measure for the class 'PER'

8.1.5.2 Error Analysis

The first approach includes a lexicon based strategy for named entity recognition for which the corpus utilized has limited number of entries because of which a portion of the entities may not have been recognized. Additionally, the information set has named entities which refer to same entity but has different spellings. For example, in the information set, words 'masjid' and 'mosjid' both alluded to a similar word inferring 'mosque' yet, has distinctive spelling. Since the proposed approach utilized a corpus for NER, these elements could not be eliminated unless all possible spellings of these words are added to the corpus. The proposed approach additionally includes the utilization of an interpretation framework (Google Translation API) for deciphering expressions of Bengali to English, yet since the interpretation framework did not consider the semantics of

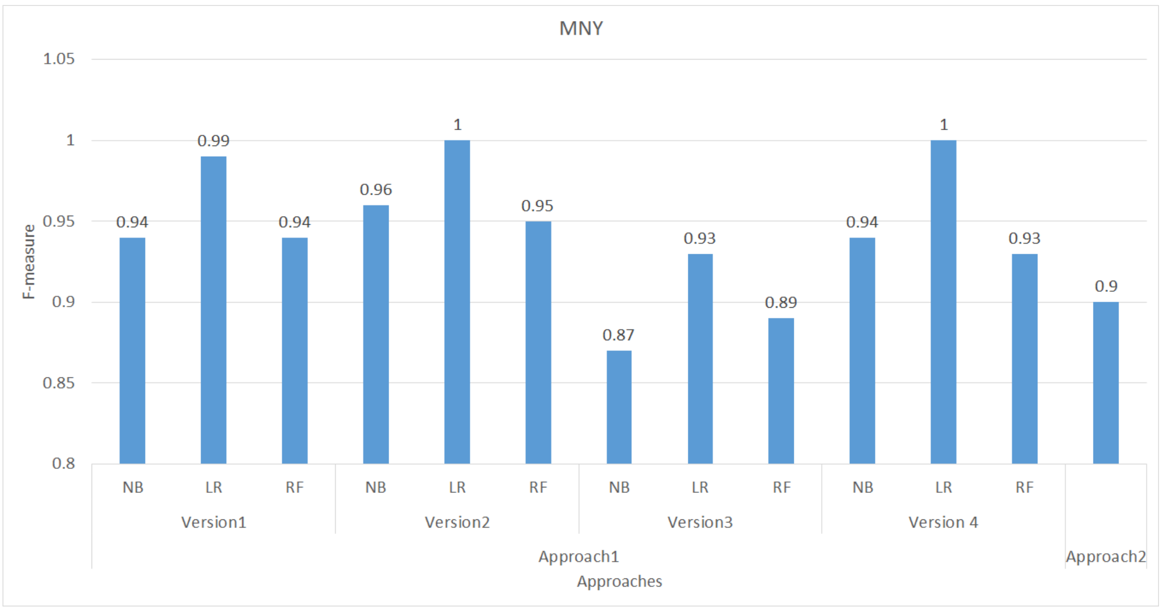


Figure 8.7: F-measure for the class 'MONEY'

the sentence where the word is being utilized, it might have happened that the specific Bengali word would have been inaccurately interpreted. The given information set did not have a uniform distribution of classes, as shown in Table 8.2. The dataset included *1.51%* of 'MISC' class while 'ORG' class includes *20%* of the passages in the information set because of which the model prepared could be biased. Due to the lesser number of instances of class 'MISC', apart from two instances in the proposed approach, no version can predict its instance in the test data. For the second approach proposed, the training dataset has relatively a small number of entries for a deep learning technique to perform well. Since the dataset is not balanced, the deep learning model is more biased towards the classes that have more instances in the training data. Since, one can only make a deep learning network as big as the system can manage, building a more extensive network is not possible, which might have contributed to the training error in the model.

8.2 Sentiment Analysis for Code Mixed Social Media Text

Large amount of data is being produced everyday corresponding to customer reviews, social media monitoring, user responses and survey feedbacks, which could be efficiently used for business analytics, computation of customer satisfaction metrics, where identifying sentiments or opinions for a given entity becomes integral. Sentiment analysis as

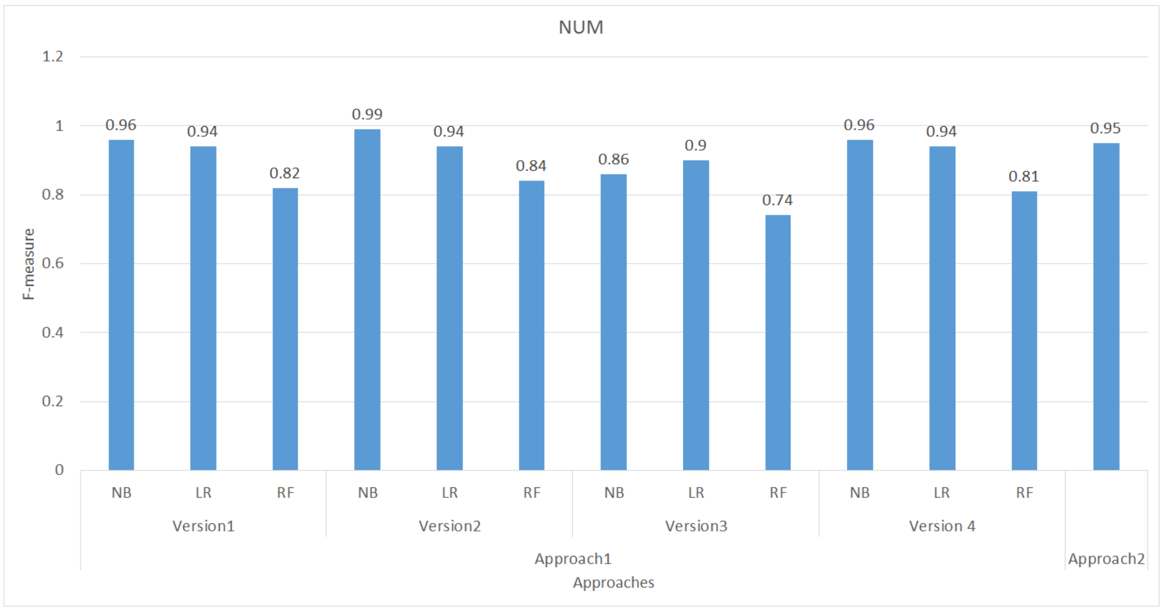


Figure 8.8: F-measure for the class 'NUM'

a part of text processing is therefore very useful, particularly as large quantities of data could be analyzed efficiently. This data usually comprises of code-mixed entries with records being written in more than one language and hence, there is a growing demand of efficient technologies for code-mix sentiment analysis. In this section, an approach has been proposed for sentiment analysis using language identification for code-mixed cross script data. For each sentence, words are first annotated with language tag they belong to, using the proposed approach of language identification. These words are transliterated, and sentiments of these words are computed using the proposed approach for sentiment analysis.

For the text processing of social media communications, in a way to understand a statement written in any social media text (blog, micro-blog, WhatsApp messages, tweets or posts), containing words belonging to different languages, the system must identify to which language, does the given utterance belong to. A statement like "Bhai this work requires zyada efforts, tu su kare che?", (translation: "Brother, this work requires more efforts on your part, what are you doing?"), represents the code mixing of three different languages of English, Gujarati and Hindi. Hence, language identification becomes an essential task for automated text processing system in a way, for extracting the sentiments of the statement.

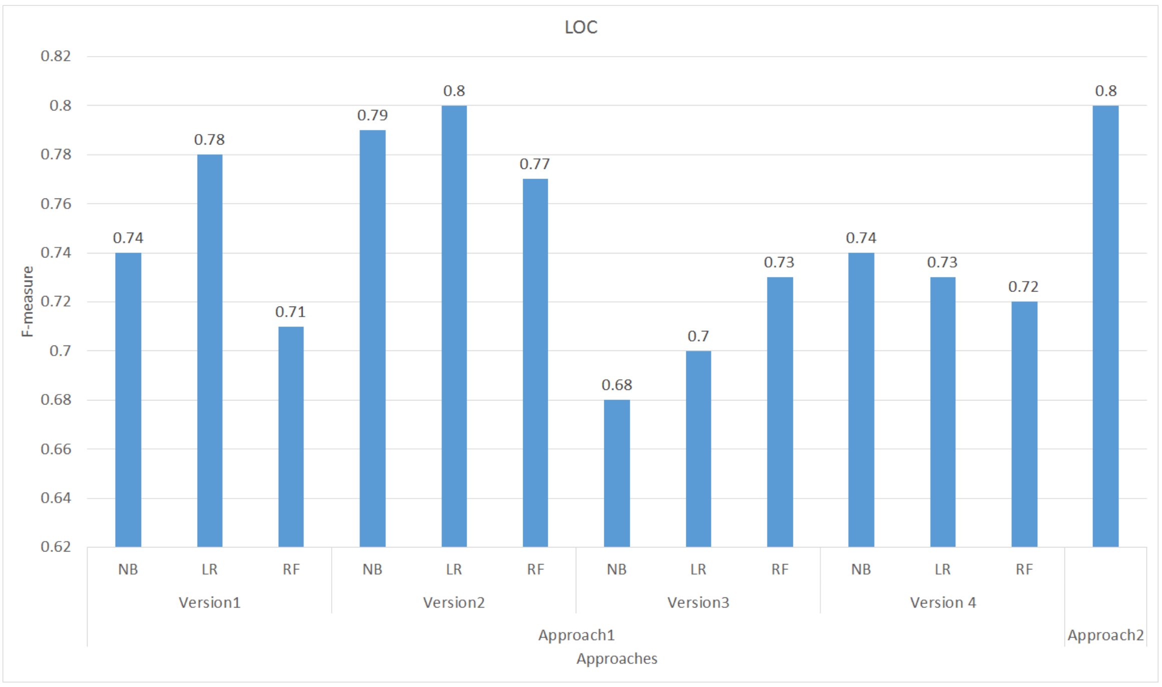


Figure 8.9: F-measure for the class 'LOC'

8.2.1 Building SentiWordNet Dictionaries for Indian Languages

SentiWordNet dictionary is well known for sentiment analysis, but there is lack of such dictionaries for Indian Languages. [Das & Bandyopadhyay 2010b] has proposed an approach to create SentiWordNet dictionary for Bengali, Hindi, Telugu and Tamil language. There is a demand for SentiWordNet dictionaries for other languages as well. In this section, SentiWordNet dictionaries are created for thirteen different Indian languages including Gujarati, Marathi, Punjabi, Urdu, Kashmiri, Konkani, Kannada, Malayalam, Bengali, Tamil, Telugu, Assamese and Manipuri.

Two sets of SentiWordNet dictionaries are created. The first set of dictionaries has a structure similar to that of dictionaries proposed by [Das & Bandyopadhyay 2010b]. The dictionary is divided into four parts, i.e. POSITIVE (words having positive sentiments), NEGATIVE (words having negative sentiments), NEUTRAL (words having neutral sentiments) and AMBIGUOUS (words having ambiguous sentiments), by their respective polarity scores.

In second set of SentiWordNet dictionary, each record comprises of POS tag, followed by the corresponding word and then its synset ID, positive and negative polarity score. These SentiWordNet dictionaries are generated by using the bilingual mapping dataset

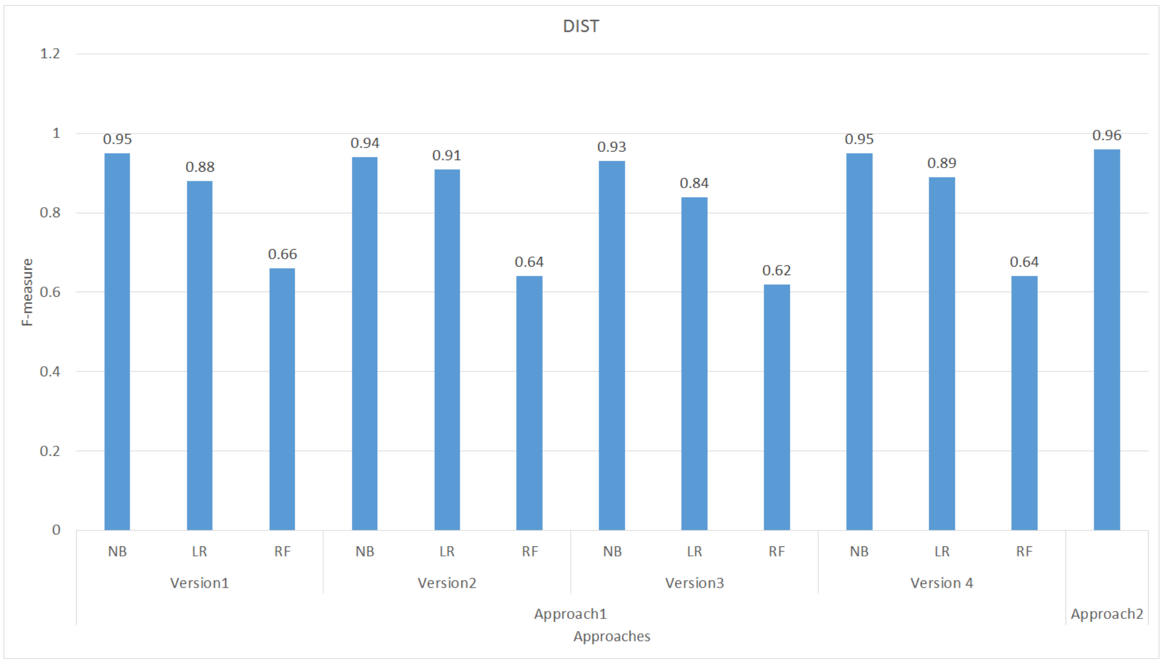


Figure 8.10: F-measure for the class 'DIST'

[Singh *et al.* 2016] and the English SentiWordNet dictionary[Baccianella *et al.* 2010c]. The dataset of bilingual mapping comprised of records such that words of one language are indexed to different synonymous words of another language. Bilingual mapping with English as their first language is used to create SentiWordNet. The English words in the English-Indian_Language mapping are looked up in the English language SentiWordNet, and their corresponding POS tag, positive and negative polarity scores are hashed to the particular word. These hashed values are then added for each of the synonyms (belonging to the Indian_Language) in the bilingual mapping, and thereby a new entry is created in the proposed Indian_Language SentiWordNet for each of these words. For duplicate entries (Indian_Language words that are synonymous with more than one English word), the largest polarity scores (both for positive and negative) are considered as the polarity score for that word. Moreover, using the reverse mapping (i.e. Indian_Language to English), a cross reference is made to measure the positive and negative scores accurately in the new Indian_Language SentiWordNet dictionary. These dictionaries are used for different versions of the proposed algorithm of sentiment analysis.

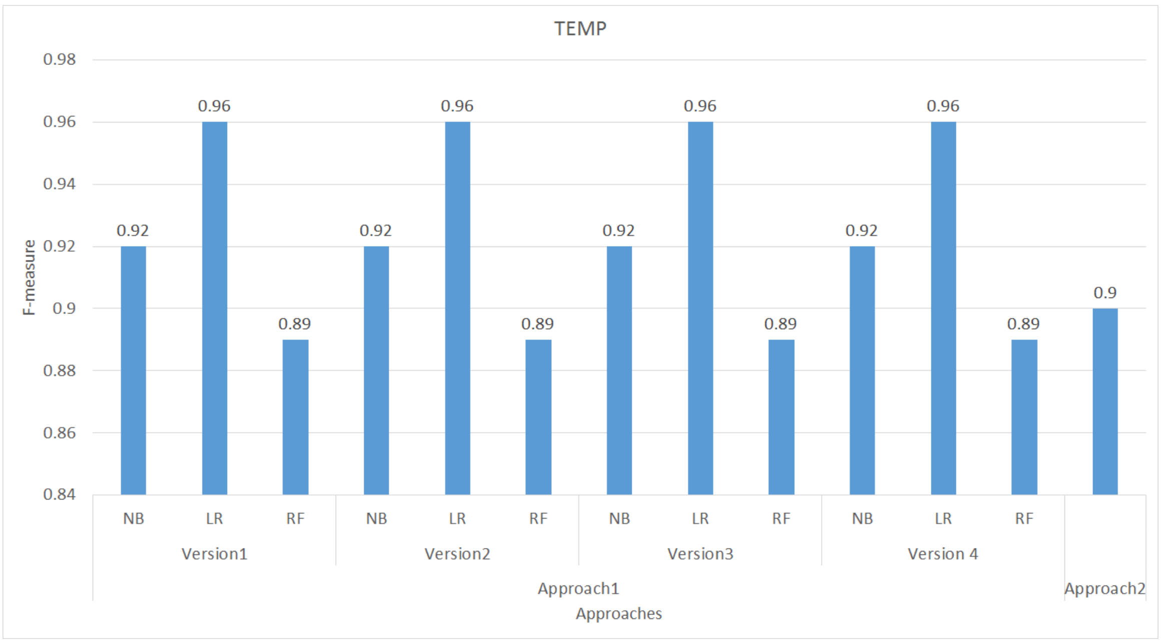


Figure 8.11: F-measure for the class 'TEMP'

8.2.1.1 Algorithm for Generation of SentiWordNet Dictionaries

Algorithm 8.3 describes the approach used for creating both sets of dictionaries. The proposed algorithm takes as input the bilingual mapping dataset of *En-Indian_Language* and English SentiWordNet(SWN) dictionary. The words in the English SWN are hashed to get values of the corresponding synsets, POS tag, positive and negative polarity scores. The corresponding English word in the mapping is then indexed to get the values of the fields mentioned above, and the same values are added for each of the synonymous words, in the mapping, thereby forming entries of the new Indian_Language-SentiWordNet dictionary. Finally, a redundancy check is applied using the function *Has_duplicates*, that checks if a word has a duplicate entry in the SWN generated. If word is duplicate, then highest score is added as polarity (both positive and negative) score for the given word. The duplicate entry (indexed by *di*, extracted through function *Duplicate_Index()*) is removed by the function *remove()*. The final word count is decremented.

Algorithm 8.3: Generation of SentiWordNet Dictionaries

```
input  : Bilingual Mapping dataset, En-IND_lan, English
         SentiWordNet dictionary En_Swn
output: Indian Language SentiWordNet Dictionary
         (IND_lan_Swn)
1  Initialization Hash_Synset=[], Hash_POS=[],
    Hash_Positive=[], Hash_Negative=[], word_count=0;
2  for i to En_Swn.length do
3      en_word =En_Swn[i].word;
4      Hash_Synset[en_word] =En_Swn[i].Synset;
5      Hash_POS[en_word] =En_Swn[i].POS;
6      Hash_Positive[en_word] =En_Swn[i].positive;
7      Hash_Negative[en_word] =En_Swn[i].negative;
8  for j to En-IND_lan.length do
9      en_word=En-IND_lan[j].word;
10     ind_words=En-IND_lan[j].mapping;
11     for k to ind_words.length do
12         IND_lan_Swn[word_count].word=ind_words[k];
13         IND_lan_Swn[word_count].Synset
            =Hash_Synset[en_word];
14         IND_lan_Swn[word_count].POS
            =Hash_POS[en_word];
15         IND_lan_Swn[word_count].positive
            =Hash_positive[en_word];
16         IND_lan_Swn[word_count].negative
            =Hash_negative[en_word];
17         word_count=word_count+1;
18 for i to IND_lan_Swn do
19     if IND_lan_Swn[i].word.Has_Duplicate() then
20         di=IND_lan_Swn[i].Duplicate_Index();
21         IND_lan_Swn[i].positive=
            max(IND_lan_Swn[i].positive,IND_lan_Swn[di].positive);

22         IND_lan_Swn[i].negative=
            max(IND_lan_Swn[i].negative,IND_lan_Swn[di].negative);

23         IND_lan_Swn.remove(di);
24         word_count=word_count-1;
```

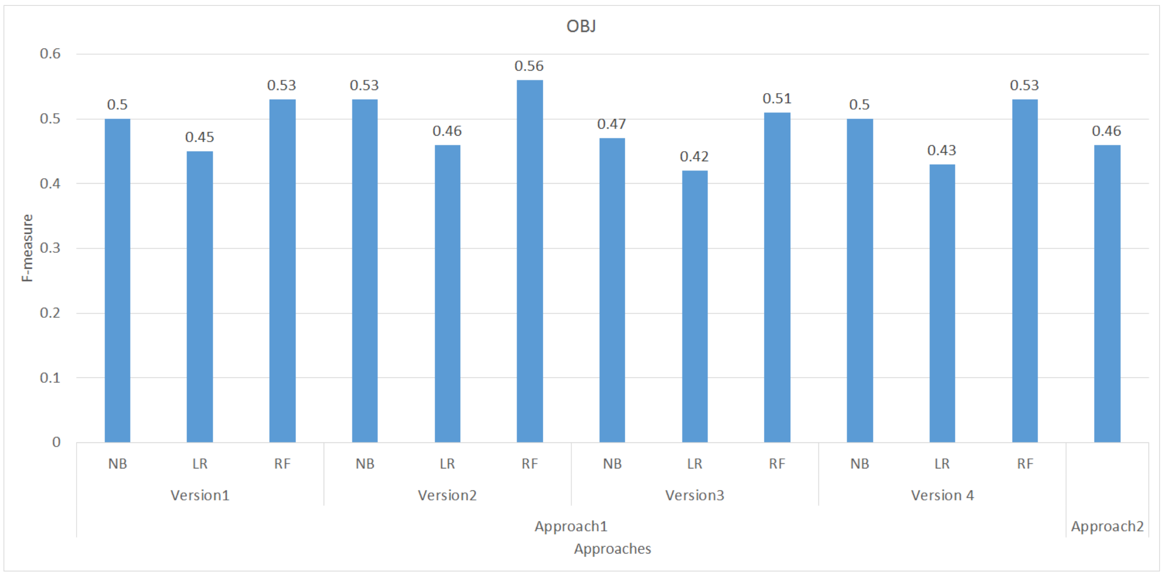


Figure 8.12: F-measure for the class 'OBJ'

8.2.2 Proposed Approach for Sentiment Analysis of Code Mixed Social Media Text

The proposed approach has been divided into three phases: language identification, transliteration and sentiment analysis. The first phase of language identification annotates each word in a sentence to their respective language. In the second phase, each sentence is transliterated to their native languages based on the annotation available from language identification phase. In the final phase, sentiment analysis is performed for individual languages based on the information available from previous two phases. All the phases are further explained below in detail:

8.2.2.1 Language Identification

Language Identification is an essential phase of the proposed approach where languages are identified at the word level. The proposed language identification system handle eight Indian languages namely Tamil, Telugu, Hindi, Bengali, Marathi, Gujarati, Kannada and Malayalam in combination with English. As named entities, emoticons and punctuations do not belong to any of the languages, they need to be classified separately. Named Entities are tagged as *NE* and Emoticons and Punctuations are tagged as *X*. In this section two approaches have been proposed for Language Identification. The first approach uses character *n*-gram and SVM whereas the second approach uses different features in combi-

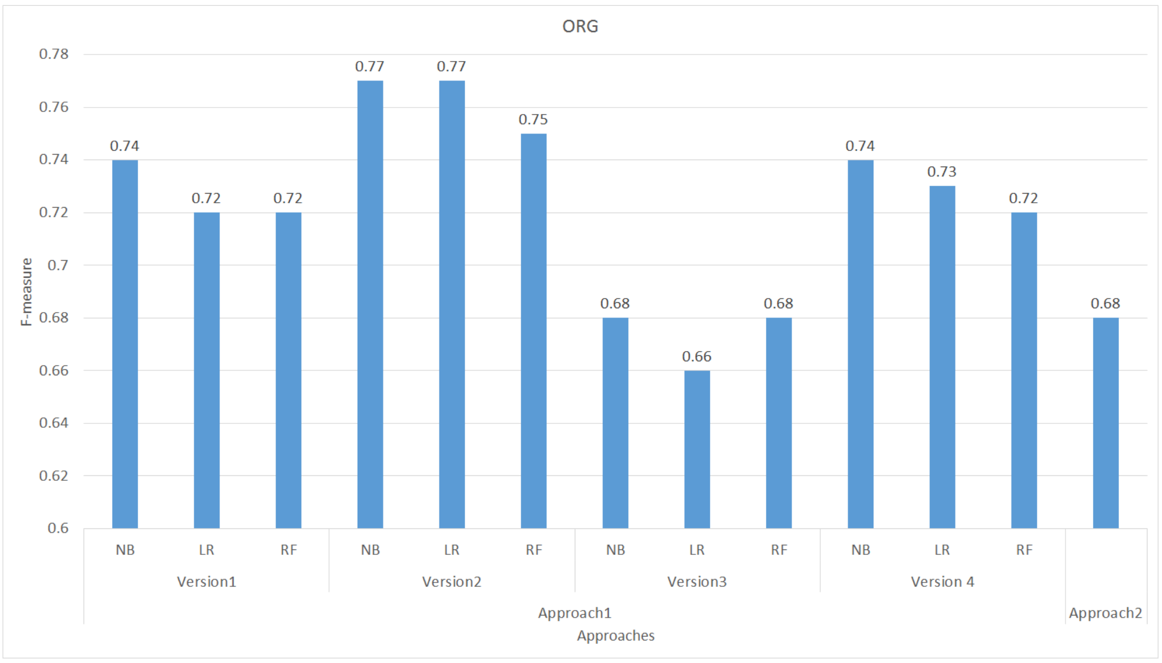


Figure 8.13: F-measure for the class 'ORG'

nation with CRF classifier for identification. For the first approach, two different versions are proposed to evaluate the system.

8.2.2.1.1 Approach 1 - Language Identification using *n*-grams

Named entities and punctuations marks do not add much information to sentiment analysis which is the final goal of section 8.2 and hence, they are identified separately. To identify NE, a dictionary containing NE and famous abbreviations are used as proposed in [Bhargava *et al.* 2015], and for identifying punctuations, the CMU ARK tagger [Owoputi *et al.* 2013] is used. The rest of the words are classified for different languages using two versions proposed below:

- Version 1 (ng_SVM): ng_SVM uses character *n-grams* where *n* varies from 1 to 4. Character *n-grams* are then collected for each language to create separate sparse matrices. These matrices are used to train one vs one multiclass SVM.

- Version 2 (h_ng_SVM): Majority of sentences in the dataset has less code switching and the number of languages in the a particuar sentence are only two. Hence, another function *H1* is added to version *ng_SVM*. Function *H1* keeps track of number of words of a particular language in a sentence. Named Entities and punctuations

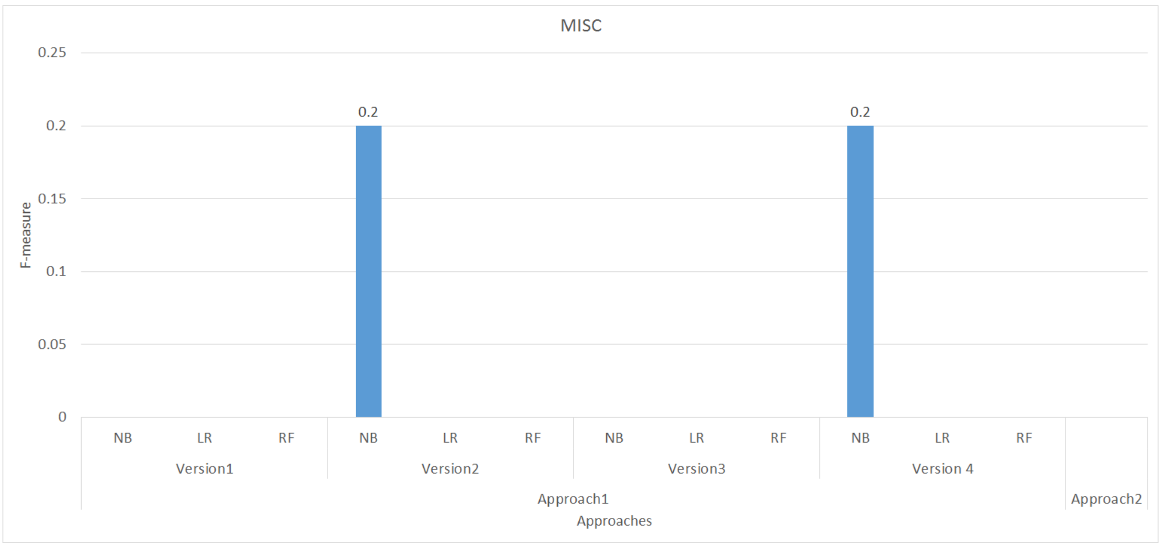


Figure 8.14: F-measure for the class 'MISC'

are excluded in the count. For the remaining words, majority voting is taken and the most common language tag is chosen. In case of a tie, most common language is chosen randomly. The rest of the Indian language tags are replaced with the most common tag.

8.2.2.1.2 Approach 2- Language Identification using Conditional Random Field

The second approach extracted features for each word and trained the classifier. The following features are extracted for each word.

- Prefix and Suffix of a word: The prefix and suffix of lengths *2* and *3* are extracted for the given word.
- Prefix and Suffix for words in context window: A context window having *3* previous words and 3 upcoming words is considered.
- Presence of word in dictionary: The presence of the current word in English dictionary (pyenchant[1]) has been used as a feature.
- Presence of context window words in Dictionary: Similar to the previous feature, the presence of context window words in English dictionary has been used as features.

[1]https://pypi.python.org/pypi/pyenchant

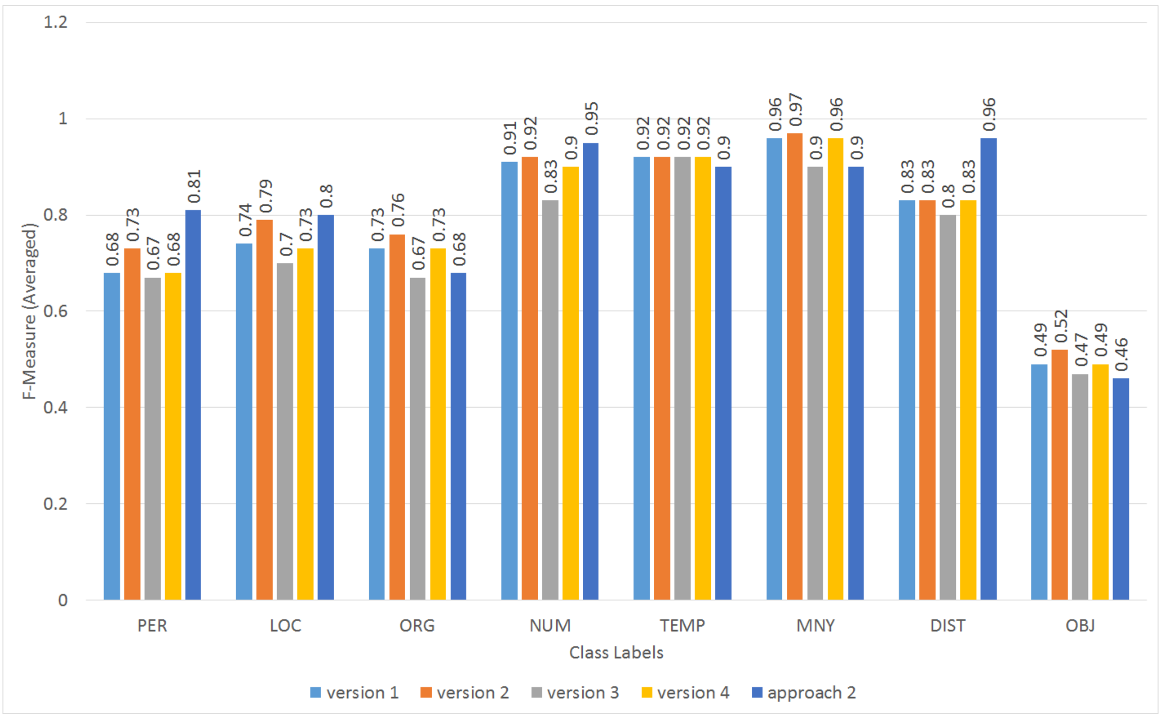


Figure 8.15: Comparing F-measure for v1, v2 and v3 for approach1 (averaged) and approach2

- POS Tags of current word: The CMU ARK tagger has been used to find out the POS tag of the current word.
- POS Tags of Context Window words: The POS tags of the previous and upcoming words have been used as features.
- Language Tags of previous three words: The language tags of the previous words that are predicted by the CRF are used.

In total for each word in a sentence, *45* features are computed *(4+4*6+1+6+1+6+3)*. These feature vectors are trained on a CRF.

8.2.2.2 Transliteration

In the second phase, each word is transliterated back to their native language script using the language annotations provided by the output of the first phase. Transliteration is performed using Google Transliteration API [Ruscoe 2009]. This transliterator is considered one of the best sources available for transliteration. It works on a dictionary based phonetic transliteration approach. However, accuracy of this phase cannot be evaluated due lack of knowledge of algorithm used by Google. Algorithm used by Google is continually

Table 8.4: Results for Approach 1 (A1) and Approach 2 (A2)

Class	Score	A1												A2
		Version1			Version2			Version3			Version4			
		NB	LR	RF	NB	LR	RF	NB	LR	RF	NB	LR	RF	
PER	P	0.58	0.51	0.49	0.59	0.62	0.57	0.52	0.54	0.51	0.57	0.52	0.46	0.75
	R	1	0.95	0.95	0.94	0.92	0.96	0.97	0.91	0.92	1.00	0.96	0.96	0.89
	F	0.73	0.66	0.65	0.72	0.74	0.72	0.68	0.68	0.66	0.73	0.68	0.62	0.81
LOC	P	0.81	0.86	0.79	0.84	0.9	0.87	0.76	0.83	0.81	0.8	0.89	0.89	0.82
	R	0.68	0.72	0.65	0.74	0.72	0.69	0.61	0.61	0.67	0.69	0.69	0.69	0.79
	F	0.74	0.78	0.71	0.79	0.8	0.77	0.68	0.7	0.73	0.74	0.73	0.72	0.8
ORG	P	0.83	0.78	0.94	0.86	0.84	0.96	0.77	0.72	0.81	0.84	0.8	0.93	0.88
	R	0.66	0.67	0.58	0.69	0.71	0.62	0.61	0.61	0.59	0.67	0.67	0.58	0.55
	F	0.74	0.72	0.72	0.77	0.77	0.75	0.68	0.66	0.68	0.74	0.73	0.72	0.68
NUM	P	0.98	0.89	0.69	1	0.91	0.73	0.89	0.91	0.63	1	0.89	0.68	0.94
	R	0.94	1	1	0.98	0.98	1	0.84	0.89	0.89	0.92	1	1	0.96
	F	0.96	0.94	0.82	0.99	0.94	0.84	0.86	0.9	0.74	0.96	0.94	0.81	0.95
TEMP	P	0.92	0.96	0.95	0.92	0.96	0.95	0.92	0.96	0.95	0.92	0.96	0.95	0.91
	R	0.92	0.96	0.84	0.92	0.96	0.84	0.92	0.96	0.84	0.92	0.96	0.84	0.90
	F	0.92	0.96	0.89	0.92	0.96	0.89	0.92	0.96	0.89	0.92	0.96	0.89	0.9
MNY	P	0.88	0.98	1	0.93	1	1	0.82	0.92	0.92	0.89	1	1	0.91
	R	1	1	0.89	1	1	0.9	0.92	0.94	0.86	1	1	0.88	0.9
	F	0.94	0.99	0.94	0.96	1	0.95	0.87	0.93	0.89	0.94	1	0.93	0.9
DIST	P	1	0.99	0.97	0.96	1	1	0.96	0.92	0.91	1	1	1	0.98
	R	0.91	0.79	0.5	0.93	0.84	0.47	0.90	0.77	0.47	0.90	0.80	0.48	0.94
	F	0.95	0.88	0.66	0.94	0.91	0.64	0.93	0.84	0.62	0.95	0.89	0.64	0.96
OBJ	P	0.67	0.74	0.8	0.73	0.77	0.84	0.64	0.71	0.79	0.67	0.75	0.8	0.69
	R	0.4	0.32	0.4	0.41	0.33	0.42	0.37	0.3	0.38	0.4	0.3	0.4	0.34
	F	0.5	0.45	0.53	0.53	0.46	0.56	0.47	0.42	0.51	0.5	0.43	0.53	0.46
MISC	P	0	0	0	0.5	0	0	0	0	0	0.5	0	0	0
	R	0	0	0	0.12	0	0	0	0	0	0.12	0	0	0
	F	NA	NA	NA	0.2	NA	NA	NA	NA	NA	0.2	NA	NA	NA

updated. This phase results in the transliterated sentences of the input file generated by the first phase.

8.2.2.3 Sentiment Analysis

In final phase, sentiment analysis for code mixed text is performed. For sentiment analysis, two approaches are proposed. The first approach uses SentiWordNet dictionaries (proposed in section 8.2.1). The second approach uses traditional machine learning techniques along with the usage of SentiWordNet dictionaries generated.

8.2.2.3.1 Approach 1: Sentiment Analysis using Dictionary based Approach

In this approach, each sentence in the transliterated form along with its annotation are passed to the analyzer. Since the data used is from social media, there would be many emoticons present. Emoticons are very good at depicting the sentiment of a sentence. Hence, sentiments from emoticons are also extracted and are given top priority. If emoticons are not present then, the analyzer searches for each word of the sentence, in SWN dictionary. To reduce the search time and narrow the search, the language annotation tag is used. As per the annotation tag, words are searched in the corresponding dictionary. If there is an entry for the given word in the SWN dictionary, then the polarity of the word contributes to overall sentiment of the sentence. This is done in two ways, based on the score and count of positive and negative words, and thereby two different versions of the algorithm are implemented. Finally, the polarity scores of each word contribute towards the computation of the sentiment of the sentence.

- Version 1 (SA_Count): In this version of the algorithm, the sentiment of a sentence is taken as the major polarity obtained by word count. If the number of positive words in the sentence is greater than the number of negative words, then the sentence is classified with positive sentiment, if the number of positive and negative words are equal then it is termed neutral, else it is classified with negative sentiment. The SentiWordNet dictionaries used for this purpose has the same structure as those created by [Das & Bandyopadhyay 2010b]. Each word is then looked up in the four different sub-versions (pos, neg, neu and ambi) of SentiWordNet dictionary of corresponding language. If the word belongs to 'pos' sub-version, the count of positive words is incremented, if it is found in the 'neg' sub-version, the count of negative words is incremented. However, the words belonging to 'neu' or 'ambi' sub-versions do not contribute towards the sentiment of the sentence. Finally, a comparison of the count of positive and negative words is made, such that the larger count (positive or negative) corresponds to the sentiment of the sentence, and if they are equal, then the sentence is said to be neutral.
- Version 2 (SA_Score): In this version, the sentiment is computed by polarity scores for each word in the sentence. The positive and negative polarity scores of each word

is added, and finally, if the total positive score is greater than the total negative score, the sentence is classified with positive sentiment, if the total negative score is greater than the positive counterpart, the sentence is classified with negative sentiment, else the sentence is said to be neutral. The SentiWordNet dictionaries used in this version of the algorithm, are the ones generated using algorithm 8.3, containing both the positive and negative polarity scores for each word. The positive and negative scores are added separately for each word of the sentence, and finally, the greater one is said to be the sentiment for the given sentence. Using this approach, the contextual information is not lost, as total of polarity scores (both negative and positive) are used to evaluate the sentiments.

8.2.2.3.2 Approach 2: Sentiment Analysis using traditional machine learning

In this approach, sentiment analysis is performed using traditional machine learning technique. Features for the same are built using SentiWordNet (created in Section 8.2.1). Each word in a sentence is checked for the presence in the SentiWordNet. If present, its positive and negative score are obtained and based on its polarity, the count of the respective polarity is increased. The sentence is also checked for presence of emoticons. This detection is implemented by a dictionary based approach. Separate counts for positive, negative and neutral emoticons are maintained. For each sentence, eleven features are extracted (as shown in Table 8.5) and a feature vector is build using the same. This feature vector is then passed through a random forest classifier to classify the sentiments of the particular sentence.

Table 8.5: Features used for Sentiment Analysis using traditional machine learning approach

S.No	Features
1	Count of positive, negative and neutral words
2	Scores of positive, negative and neutral words
3	Positive and negative scores of Ambiguous words
4	Count of positive, negative and neutral emoticons

8.2.3 Algorithm for Sentiment Analysis for Code Mixed Social Media Text

In this section, algorithm for language identification and sentiment analysis has been explained.

8.2.3.1 Language Identification

Algorithm 8.4 discusses the approach of version 1, *ng_SVM*. List of code-mixed sentences, *S* is fed to the algorithm as input such that each sentence is represented as a list of words. Hence, *S* becomes a list of lists. On similar lines, the annotations of the input sentences (represented by *P*) is also a list of lists. For each sentence, the POS tag of each word is computed using the CMU ARK tagger (represented by the function: *Tagger()*) and stored in the array *POS*. Now for each word, if the word is punctuation the *Tagger* function returns *X*. For identifying named entity, dictionary is used (represented by *NE*, initiated by function: *Named_Entity_dictionary()*) which searches for the presence of current word (implemented by function: *ispresent()*). If the word is a named entity or punctuation, then it is directly added to the final prediction. Otherwise, its character *n*-grams are built where *n* varies from *2* to *4* (implemented by function: *compute_char_n_grams()*) and are passed to the SVM classifier (already trained, represented by *clf*) for prediction of the language tag (implemented by function: *predict()*). The tag is updated to the final prediction list. The proposed approach of *h_ng_SVM* is explained in algorithm 8.5. Here, list of code-

Algorithm 8.4: Language Identification using ng_SVM

```
input : Code-Mixed sentences, S
output: Predicted Language Annotations, P
1  Initialization P=[], svm_data=[];
2  NE = Named_Entity_dictionary();
3  for $i = 0$ to $S.length$ do
4      POS= Tagger(S[i]);
5      for $j = 0$ to $S[i].length$ do
6          if NE.ispresent(POS[i][j]) or POS[ j ] ='X' then
7              P[i][j]=POS[j];
8          else
9              n_grams= compute_char_n_grams(S[i][j]);
10             P[i][j] = clf.predict(n_grams) ;
```

mixed sentences *S* is passed as input, which is a list of lists similar to algorithm 8.4.

The input sentences are passed through the *ng_SVM* classifier (implemented by function: *ng_SVM()*) and the predicted sentences are stored in a list of lists, *svm_data*. For each word in a sentence, counter (represented by *cnt*, initialized by *counter()*) increments the count of respective tags, if token is not a punctuation (*X*) or named entity(*NE*), as shown in step 9 of algorithm 8.4. After counting all the words in a sentence, the most common language tag is obtained (implemented by function: *most_common()*). Then, all other Indian language tags are replaced with the majority tag. The final updated tags are stored in the list, *P*. Second proposed approach for language identification uses CRF model as explained in

Algorithm 8.5: Language Identification using h_ng_SVM

```
   input : Code-Mixed sentences, S
   output: Predicted Language Annotations, P
1  Initialization P=[], svm_data=[];
2  svm_data=ng_SVM(S);
3  for i=0 to svm_data.length do
4      cnt = counter();
5      for j = 0 to svm_data[i].length do
6          if svm_data[i][j]!='NE' or svm_data[i][j] !='X' then
7              cnt[svm_data[i][j]]+= 1;
8      lang_tag = cnt.most_common(1);
9      for j = 0 to svm_data[i].length do
10         if svm_data[i][j]!='NE' or svm_data[i][j]!='X' or
            svm_data[i][j]!='en' then
11             P[i][j]=lang_tag;
12         else
13             P[i][j]=svm_data[i][j];
```

algorithm 8.7. The code mixed input sentences are present in the list *S*. Each sentence in the input data is already tokenized and stored in a list. *S* is a list of lists and each sentence is represented by *S[i]*, where *i* is the index of the array. For each word in a sentence (represented by *S[i][j]*), feature vectors are computed (implemented by function: *Build_Feature_Vector()*). While computing the feature vector, the information of previous language tags are required, so a list *prev* is used to store the previous tags. These feature vectors are given to the CRF classifier for predicting the appropriate tag (implemented by function: *predict()*). After prediction, the current tag is updated in the *prev* list for the next word.

Algorithm 8.7: Language Identification using CRF

```
input  : Code-Mixed sentences, S
output: Predicted Language Annotations, P
1 Initialization P=[], FV=[];
2 clf = CRF_classifier();
3 for $i = 0$ to $S.length$ do
4     prev=[];
5     sent_tag=[];
6     for $j = 0$ to $S[i].length$ do
7         FV = Build_Feature_Vector(S[i], prev);
8         tag = clf.predict(FV);
9         sent_tag.append(tag);
10        prev.append(tag);
11    P.append(sent_tag);
```

8.2.3.2 Sentiment Analysis

Algorithm 8.8 implements the proposed approach for sentiment analysis using count of SentiWordNet dictionary (SA_Count). The proposed algorithm computes the sentiment of a code-mixed sentence by majority word count supporting the required polarity. Code mixed sentences and predicted language annotations are passed as input. For each sentence, count of words with positive and negative sentiment are maintained separately. Each word in the sentence is looked up in the SentiWordNet dictionary of corresponding language using predicted language annotations. Each SWN dictionary is divided into four parts as described in section 8.2.1. If the word is found in the POS subversion, the count of positive sentiment is incremented else if the word belongs to NEG subversion, the count of negative sentiment is incremented. If the word is found in NEU or AMBI subversion or is not found in the SWN dictionary, then it does not contribute towards sentiment of the sentence. Finally, the count of positive and negative sentiments are compared, and the prominent sentiment of the input sentence is considered, else sentence is classified as neutral.

Algorithm 8.9 discusses the approach followed by SA_Score. The proposed algorithm computes the sentiment of a code-mixed sentence by the sum of polarity scores of each word in the sentence. The code mixed sentences and predicted language annotations are passed as input. For each sentence, the total positive and negative score of the words are maintained separately. Each word in the sentence is looked up in the SentiWordNet

Algorithm 8.8: SA_Count

```
input  : Code-Mixed sentences, S, Predicted Language
         Annotations, P, Indian SentiWordNet Dictionaries
         (without scores), IND_Swn
output: Sentiment of Input sentences, SE
1  Initialization SE=[];
2  for $i = 0$ to S.length do
3      positive_count=negative_count=0;
4      for j= 0 to S[i].length do
5          word= S[i][j];
6          lang_tag=P[word];
7          lang_Swn=Indian_Swn[lang_tag] ;
8          if word in lang_Swn.POS then
9              positive_count=positive_count+1;
10         else if word in lang_Swn.NEG then
11             negative_count=negative_count+1 ;
12     if $positive\_count > negative\_count$ then
13         SE[i]=positive;
14     else if $positive\_count < negative\_count$ then
15         SE[i]=negative;
16     else
17         SE[i]=neutral;
```

dictionary of corresponding language, using predicted language annotations. If there is an entry of that word in the SWN dictionary, then the positive and negative polarity score of that word is added separately to the total positive and negative scores of the sentence. Finally, total positive and negative scores computed for the sentence are compared. The greater one is regarded as the sentiment of the input sentence. If the count is equal, sentiment of the sentence is classified as neutral.

Algorithm 8.10 explains the proposed traditional machine learning approach for sentiment analysis. The sentences *S*, are constructed as a list of lists. For each sentence the number of positive, negative, neutral words are found out by searching each word in the SentiWordNet(s) (represented by *SWN*, loaded by the function: *load_SentiWordNets()*) of the respective language by the function *find_SWNcounts()*. The positive, negative, ambiguous positive and ambiguous negative scores are found out for all the sentences using function *find_SWNscores()*. Emoticons in the sentence are identified, and a total number of positive, negative and neutral emoticons are returned (implemented by function:

Algorithm 8.9: SA_Score

```
input : Code-Mixed sentences, S, Predicted Language
        Annotations, P, Indian SentiWordNet Dictionaries
        (without scores), IND_Swn_Score
output: Sentiment of Input sentences, SE
1  Initialization SE=[];
2  for $i = 0$ to S.length do
3      positive_score=negative_score=0;
4      for j= 0 to S[i].length do
5          word= S[i][j];
6          lang_tag=P[word];
7          lang_Swn=Indian_Swn[lang_tag] ;
8          if word in lang_Swn then
9              positive_score=positive_score
                +lang_Swn[word].positive;
10             negative_score=negative_score
                +lang_Swn[word].negative;
11     if positive_score > negative_score then
12         SE[i]=positive;
13     else if positive_score < negative_score then
14         SE[i]=negative;
15     else
16         SE[i]=neutral;
```

find_emoticons()). Using the information extracted *feature_vectors* are built (implemented by function: *Build_Feature_Vector()*) and the sentiment is predicted (implemented by function: *predict()*).

8.2.4 Data Description and Analysis

MSIR, FIRE 2015 dataset is used for building the models. The training data has *2908* sentences, and the test data has *792* sentences. The dataset consists of nine languages (Bengali, English, Gujarati, Hindi, Malayalam, Marathi, Kannada, Tamil, Telugu) and all the words are present in the transliterated form.

As shown in Table 8.6, utterances has majority of English tags followed by punctuation tags in both train and test data but there is a change in the composition of Indian language words. In the training data, there are many Telugu words *(12.57%)*, but in test data of Telugu, the percentage of words dropped to *4.36%*. In case of Tamil, there is a similar

Algorithm 8.10: SA_RF

```
   input  : Code-Mixed sentences, S
   output: Predicted Sentiment Analysis, P
1  Initialization P=[], FV=[];
2  clf = RandomForestClassifier();
3  SWN = load_SentiWordNets();
4  for $i = 0$ to $S.length$ do
5      counts = find_SWNcounts(S[i], SWN);
6      scores = find_SWNscores(S[i], SWN);
7      emoticons = find_emoticons(S[i]);
8      FV = Build_Feature_Vector(counts, scores, emoticons);
9      senti = clf.predict(FV);
10     P.append(senti);
```

Table 8.6: Number of tokens and their percentages in the testing and training data

	Training data		Testing data	
Language	Token Count	Percentage	Token Count	Percentage
Bengali (bn)	3562	6.91	1368	11.40
English (en)	17952	34.85	4048	33.73
Gujarati (gu)	890	1.72	185	1.54
Hindi (hi)	4456	8.65	1601	13.34
Malayalam (ml)	1159	2.25	231	1.92
Marathi (mr)	1960	3.80	454	3.78
Kannada (kn)	1671	3.24	598	4.98
Tamil (ta)	3170	6.15	543	4.52
Telugu (te)	6477	12.57	524	4.36
Punctuation (X)	7712	14.97	1872	15.60
Named Entities (NE)	2414	4.68	551	4.59
MIX	69	0.13	24	0.02
Others	14	0.02	0	0
Total	51506	100.00	11999	100.00

trend but not that distinguishable. The drop is from *6.15%* to *4.52%*. In case of Bengali words, there are only *6.9%* words in the training data, but in the testing data, the number of words have spiked up to *11.4%*. The Hindi words have a variation similar to Bengali. For other Indian languages, there is a small change in their respective compositions. All these languages have less than *4%* of the training data and the change in their percentages is not that distinguishable.

Figure 8.16 presents the amount of code switching present in a sentence. The figure

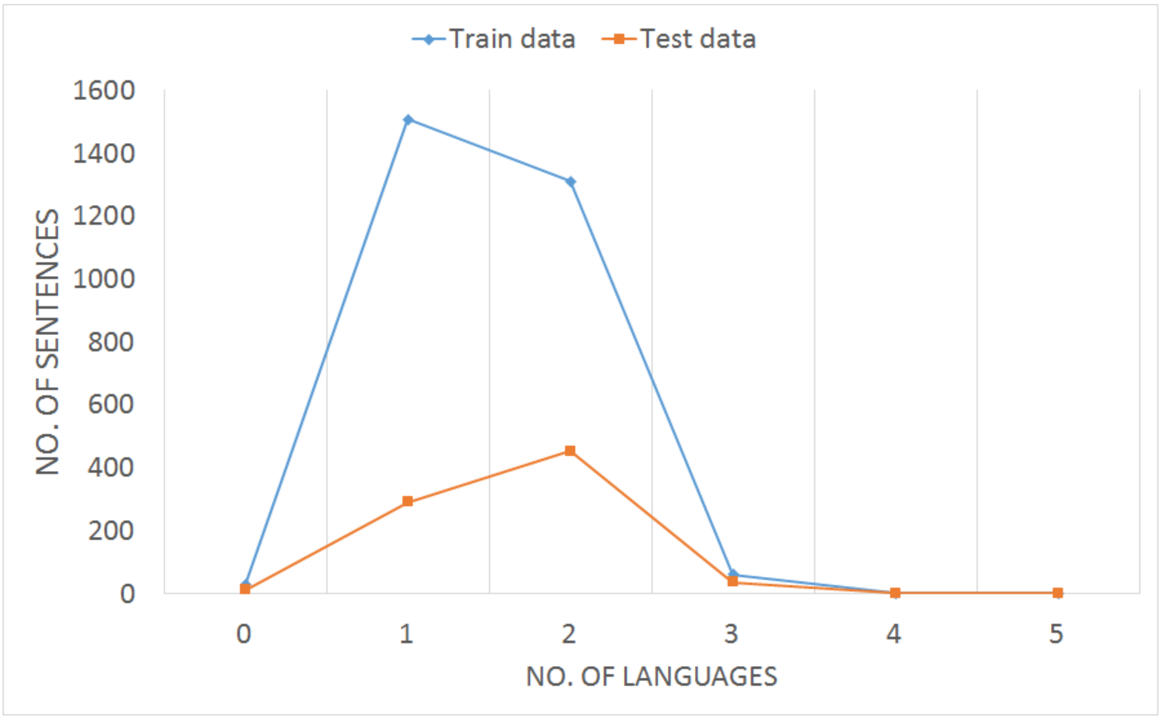


Figure 8.16: Number of Sentences per Language

excludes the presence of named entities and punctuations. If the number of languages in a sentence is zero, it implies that the whole sentence has only *NE* and *X* tags. Sentences with two or more languages are code mixed. As shown in Table 8.6, most of the sentences in the dataset are either monolingual or bilingual. There are very few sentences with three languages.

As depicted in Figure 8.17, the dataset has many sentences with English words followed by Hindi, Bengali, Kannada and Telugu. There are very few instances for rest of the languages. All the languages that have fewer sentences, have more subjective sentences. Telugu tag is associated with more subjective sentences. Difference in the number of subjective and objective sentence varies from language to language as shown in Figure 8.17.

8.2.5 Experiments and Results

In this section, the language detection and sentiment analysis are done on the MSIR 2015 dataset [Sequiera *et al.* 2015b]. For language identification, three versions are proposed. The first one used character *n*-grams as features and classified using SVM classifier

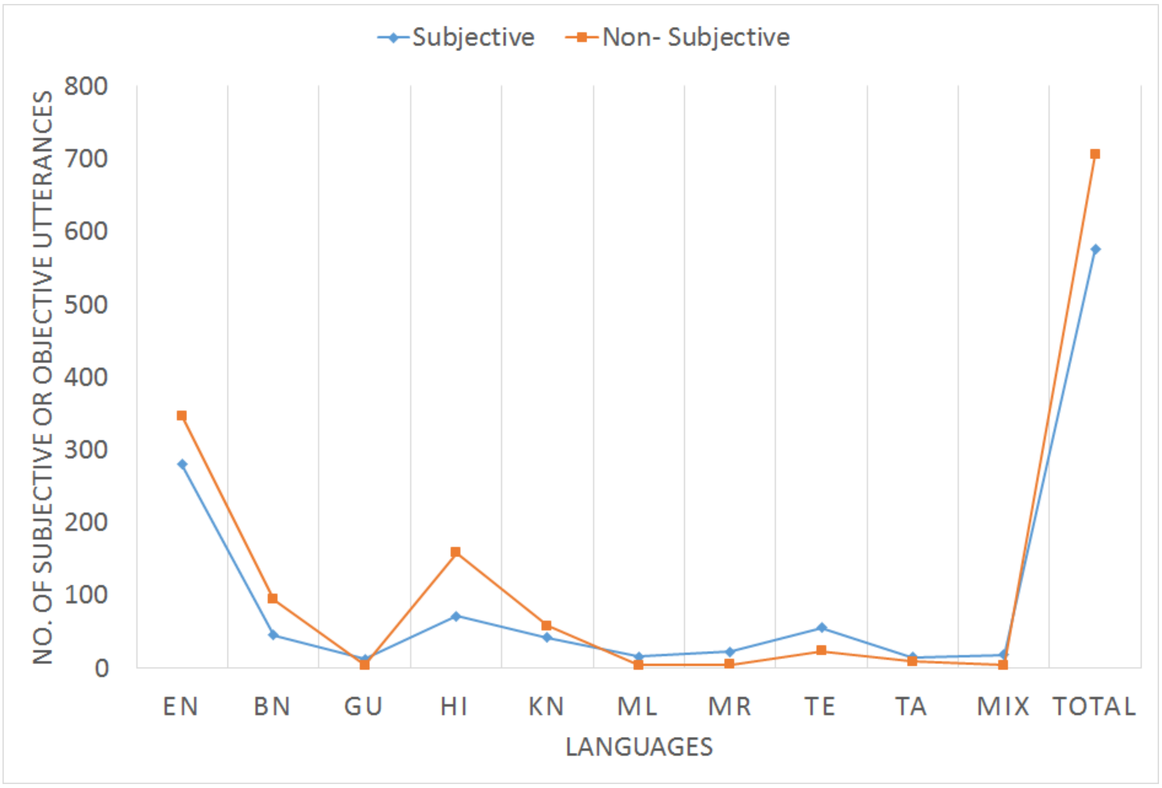


Figure 8.17: Number of Subjective/Objective Sentences per Language

(*ng_SVM*). The second model (*h_ng_SVM*) is an extended version of the first model. There is an extra majority voting done on the first model. The third model is CRF based Language detector. Similarly, three different approaches have been evaluated for sentiment analysis of code mixed data. Two of them uses SentiWordNet dictionaries of different Indian languages for sentiment analysis. The first approach computes the majority word count favouring a particular sentiment for a given sentence whereas the second approach uses polarity scores of each word for evaluating sentiment of the code mixed sentence. In the third approach, sentiment analysis is performed using traditional machine learning techniques. Features for traditional machine learning are built using SentiWordNet dictionaries as discussed in section 8.2.2.3.2. Section 8.2.5.1 provides results along with discussion and error analysis for each of the approaches mentioned.

8.2.5.1 Evaluation & Discussion

From Table 8.7, it is evident that the average precision for CRF is high compared to the other two models for language identification but while considering average F-measure

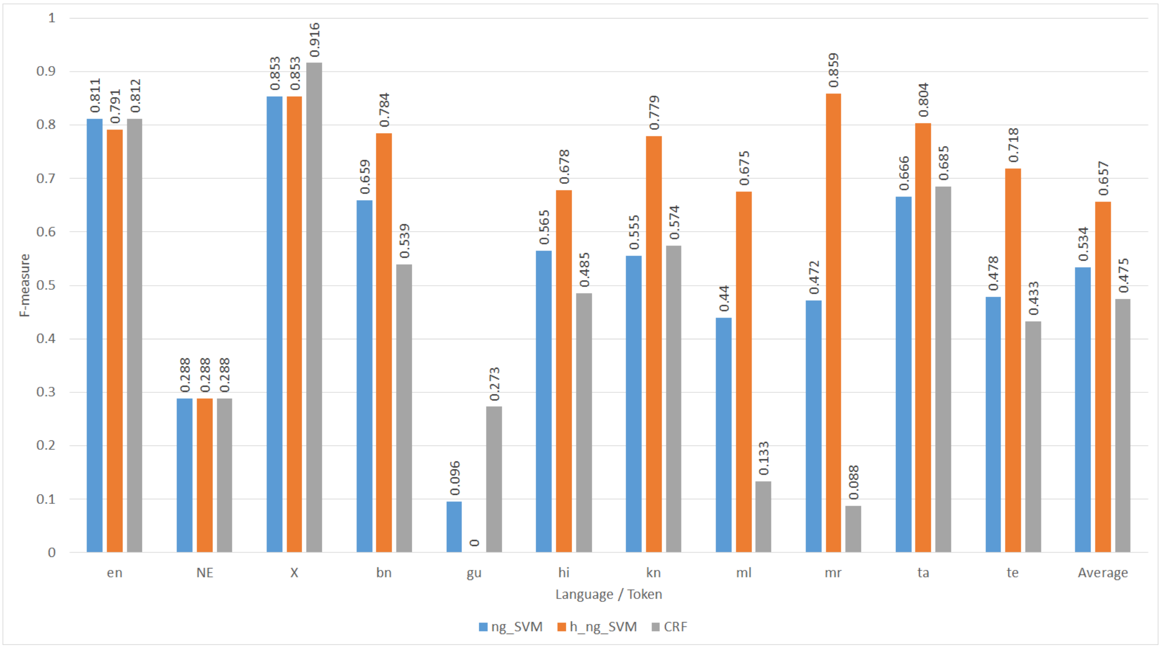


Figure 8.18: Comparison of all the three proposed approach for Language Identification

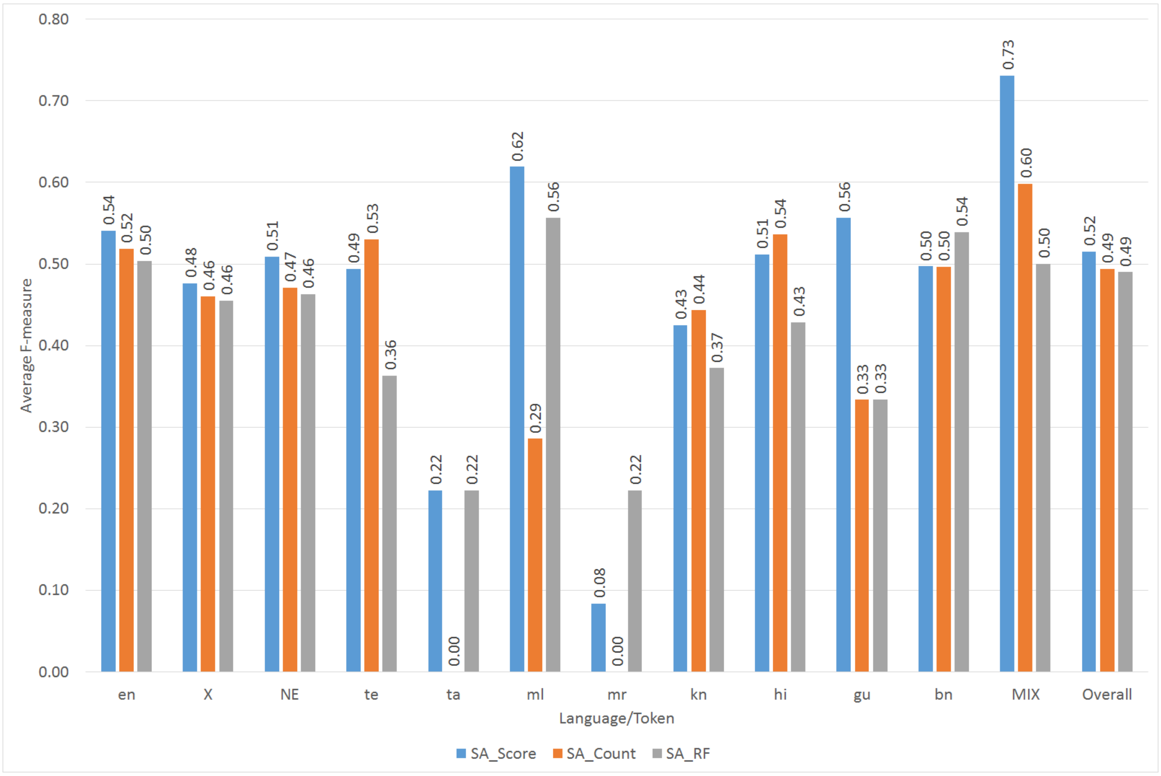


Figure 8.19: Comparison of all the three proposed approach for Sentiment Analysis

Table 8.7: Results for Language Identification

	Precision			Recall			F-measure		
	ng_ SVM	h_ng_ SVM	CRF	ng_ SVM	h_ng_ SVM	CRF	ng_ SVM	h_ng_ SVM	CRF
en	0.831	0.794	0.696	0.791	0.788	0.973	0.811	0.791	0.812
NE	0.225	0.225	0.18	0.399	0.399	0.728	0.288	0.288	0.288
X	0.791	0.791	0.88	0.926	0.926	0.956	0.853	0.853	0.916
bn	0.737	0.846	0.817	0.596	0.729	0.402	0.659	0.784	0.539
gu	0.101	0	1	0.092	0	0.158	0.096	0	0.273
hi	0.641	0.765	0.838	0.505	0.609	0.341	0.565	0.678	0.485
kn	0.67	0.88	0.786	0.473	0.699	0.452	0.555	0.779	0.574
ml	0.497	0.844	0.333	0.394	0.563	0.083	0.44	0.675	0.133
mr	0.535	0.909	0.667	0.423	0.815	0.047	0.472	0.859	0.088
ta	0.576	0.735	1	0.788	0.886	0.521	0.666	0.804	0.685
te	0.408	0.649	0.606	0.578	0.803	0.336	0.478	0.718	0.433
Avg	0.5466	0.6762	0.71	0.542	0.656	0.454	0.534	0.657	0.475

and average recall the *h_ng_SVM* performs the best. This trend is not followed for individual languages. As shown in Figure 8.17, for English and named entities all the versions performed approximately same. In case of punctuation and Gujarati, the CRF based model performed better than other two models. For the rest of the language tags the *h_ng_SVM* outperforms the rest of the models but *h_ng_SVM* has a zero F-measure for the Gujarati words. This can be explained as follows: *ng_SVM* classifier might have marked only some of the Gujarati words in a sentence but when *h_ng_SVM* applies majority voting, the count of Gujarati words may not be the majority tag. Hence, all the Gujarati words annotations are replaced. The poor classification of Gujarati words could be due to the fact stated by [Sequiera *et al.* 2015b], that Hindi-Gujarati is found to be most confusing language pair in MSIR 2015 dataset. Results of approaches for sentiment analysis have been discussed below:

8.2.5.1.1 SA_Count

A language wise detailed description of average precision, recall and F-measure for *SA_Count* (as described in algorithm 8.8) is shown in Table 8.8. It is evident from Figure 8.19 that word instances of Hindi language obtained highest F-measure. This approach also works well for sentiment analysis of English, Telugu and Bengali languages. The

Table 8.8: Results for Sentiment Analysis

	Average Precision			Average Recall			Average F-measure		
	SA_ Score	SA_ Count	SA_ RF	SA_ Score	SA_ Count	SA_ RF	SA_ Score	SA_ Count	SA_ RF
en	0.548	0.539	0.517	0.538	0.511	0.506	0.541	0.519	0.504
X	0.482	0.486	0.461	0.478	0.464	0.459	0.476	0.46	0.455
NE	0.518	0.497	0.471	0.508	0.462	0.474	0.509	0.471	0.463
te	0.559	0.576	0.412	0.723	0.754	0.36	0.493	0.530	0.363
ta	0.333	0	0.167	0.167	0	0.333	0.222	0	0.222
ml	0.667	0.333	0.667	0.583	0.25	0.5	0.619	0.286	0.557
mr	0.333	0	0.267	0.0477	0	0.190	0.083	0	0.222
kn	0.497	0.552	0.451	0.449	0.471	0.383	0.425	0.443	0.372
hi	0.531	0.566	0.412	0.515	0.526	0.513	0.511	0.536	0.428
gu	0.667	0.333	0.25	0.5	0.333	0.5	0.557	0.333	0.333
bn	0.546	0.546	0.571	0.478	0.478	0.523	0.497	0.497	0.539
MIX	0.722	0.611	0.517	0.857	0.690	0.524	0.731	0.598	0.5
Overall	0.539	0.532	0.503	0.507	0.485	0.489	0.515	0.494	0.49

lower F-measure scores of Gujarati and Malayalam instances could be attributed to the lower number of sentences of these languages. It can be seen that proposed approach works relatively better for instances where code-mixing is involved. Highest precision score is obtained for sentences with words of Kannada and Hindi languages, while highest recall is observed for Telugu language instances.

8.2.5.1.2 SA_Score

A detailed description of precision, recall and F-measure scores obtained for each language using the approach of *SA_Score* (as described in algorithm 8.9) is shown in Table 8.8. It can be observed from Figure 8.19 that the highest F-measure scores are obtained for instances belonging to Malayalam language. The approach also works well for sentiment analysis of languages of Hindi, English and Gujarati. F-measure score of *0.731* is observed for the code mixed sentences. The highest precision scores are found for utterances belonging to Gujarati language and highest recall are found for Telugu instances. It is also evident that the F-measure scores for words belonging to neutral category is consistently higher in case of most of the languages.

8.2.5.1.3 SA_RF

F-measure, recall and precision scores obtained using the approach of random forest (as described in Algorithm 8.10) classifier for sentiment analysis is shown in Table 8.8. It can be observed from Figure 8.19, that highest F-measure scores are obtained for sentences with words of Malayalam language. This approach also performs well for Bengali and English language instances. It is evident that this algorithm performs relatively better for code mixed sentences. Using this approach, the highest precision and F-measure are observed for Bengali language instances.

As evident from Figure 8.19, SA_Score version of algorithm performs relatively better as compared to the others, for sentiment analysis of most of the languages. It is seen that sentiment analysis of code-mixed sentences has a higher F-measure as compared to the other individual languages. It can also be observed that SA_Count approach performed best for Hindi and Telugu language instances, SA_RF approach performed best for Bengali language instances, while sentiment analysis for all the other languages is best achieved using SA_Score approach. This could be attributed to the fact that using scores for sentiment analysis does incorporate contextual information and negation, as positive and negative scores both are involved for computing sentiment of a sentence.

8.2.5.2 Error Analysis

For a better sentiment analysis model, perfect transliteration and language annotation are needed, but some errors might have crept in both the phases. The current language identification approach relies on a dictionary for identification of named entities. Since the *NE* is traversed from various languages, it would be difficult to maintain a record of all of them. Hence, there is a possibility of missing some Named Entities. A better NER System would decrease the risk of misclassification. Even though Google API is considered the best transliterator, but there may be few errors in the transliteration. This is because users write there native language in English, even a small change in the alphabets may result in wrong transliteration. This could cause problems while searching for the word in the SentiWordNet as only string matching is being performed. In code mixed text, it is common to have punctuations written without spaces, causing two or more words to be combined. This can lead to wrong language detection, resulting in the absence of a word in SentiWordNet.

8.3 Concluding Remarks

In conclusion, algorithms are proposed for some of the applications of code mixed text such as question classification and sentiment analysis. For question classification, two approaches are proposed. First approach is based on word level *n*-gram and second is based on CNN(s) to classify code mixed cross script questions into nine different classes. The proposed algorithm based on deep Learning performs at par with the traditional machine learning based approach. It can be further improved if a larger dataset is available. Version *2* of approach *1* performed best overall for the available dataset. The reason is that translations used in the approach, normalized the text. Further, not removing named entities helped the classifiers in identifying the correct classes. Deep learning techniques performed best for classes with more number of instances, such as in case of 'ORG' class. Sentiment analysis for code mixed social media text is another application discussed in this chapter. An algorithm is proposed for language identification to improve the sentiment analysis of code mixed text. Three different approaches have been proposed for both language identification and sentiment analysis. Approaches are proposed to identify eight different languages with the highest achieved F-measure of *65.7%*. The proposed approaches for sentiment analysis are also able to classify the code mixed sentences into positive, negative and neutral sentiments for eight different code mixed Indian languages.

Chapter 9

Conclusion & Future Work

9.1 Conclusion

The thesis set out to solve challenges of sentiment analysis. It starts with the discussion of motivations, research questions and objectives in the area of sentiment analysis. A detailed literature review of sentiment analysis and its related challenges are discussed in the thesis. Problems of sentiment analysis in Indian languages, temporal sentiment analysis, paraphrasing, text summarization and spam detection are focused in the thesis. Few aspects of code mixing are also covered. Tools for code mixed text such as part of speech tagger and named entity recognizer are proposed, and few applications related to code mixing such as question classification and sentiment analysis in code mixed text have also been targeted. The thesis also focuses on the problem of sarcasm detection in monolingual and code mixed social media text.

This thesis targets the problem of sentiment analysis in Indian languages and temporal sentiment analysis. Variants of neural network architectures have been proposed and compared. A hybrid architecture is adopted for the sentiment analysis in Indian languages where CNN, LSTM and RNN are used to create *39* different sequential models. It is observed that depending upon the size of data and the complexity of language, these combinations work well for identifying sentiments. The proposed approach is tested with three Indian languages namely, Hindi, Bengali and Tamil. An aspect based temporal sentiment analysis approach has also been proposed which keeps track of changing aspects and its respective sentiments with time. CNN and clustering based approach is used for identifying the changing important aspects and sentiments with an accuracy of *96.5%* and *83.5%* respectively.

Approaches are discussed for text summarization and paraphrase detection in English and Indian languages. A significant amount of social media text needs to be summarized for finding the right set of information and reducing redundancy. Two approaches have been proposed for paraphrase detection. In first approach, classifiers such as Logistic Regression, Support Vector Machine, Naive Bayes and Random Forest are used. Logistic Regression gives a maximum accuracy of 89% for paraphrase identification. The second approach for paraphrase detection uses deep learning techniques such as LSTM, Bi-directional LSTM, RNN and CNN. A variant of CNN with WordNet is also proposed which performed better as compared to other approaches. Sequential model of two LSTM(s) also performed at par with CNN for few languages including Malayalam and Punjabi.

Extractive and abstractive text summarization have also been explored in the thesis. Proposed paraphrasing approach has been used to generate labels for annotation of sentences in extractive text summarization. Fully connected CNN has been used to generate extractive summaries. For abstractive text summarization, two approaches are proposed. The first approach uses graph based sentiment infusion, where sentences are merged on the basis of their syntactic and semantic similarity using appropriate connectors whereas in second approach generative adversarial networks are used to predict next word in abstractive summaries.

Two approaches are proposed for spam detection in English. The first approach uses classifiers such as LR, NB and SVM whereas the second approach uses a sequential model where deep learning techniques such as RNN, CNN and LSTM are combined to yield a hybrid architecture for detecting spam. Sarcasm detection in English and code mixed social media text (English and Hindi) has also been explored in the thesis. Bilingual word embeddings are used to project and map similar words in different languages in same feature vector space such that similar words are placed nearby. A sequential model has been proposed using deep learning techniques to generate a hybrid model. Different variants for sequential models are experimented. It is found that a model with CNN followed by LSTM and RNN gave the highest accuracy among all.

Algorithms have been proposed for developing tools for code mixed text. Two problems have been targeted here, Part of Speech tagging for code mix text (Hindi, Telugu and Bengali) and Named Entity Recognizer for code mix text (Hindi-English and Tamil English). POS tagging is targeted for three Indian languages namely, Hindi, Bengali and Telugu. The first approach for POS tagging proposes an ensemble based POS tagger whereas the second approach proposes tree based POS tagger. The third approach uses bidirectional LSTM to predict POS tag of next word in combination with a dictionary based approach. Named Entity Recognizer is developed for English code mixed with Hindi and Tamil. Four variants of approach are proposed for predicting NER. Decision tree and extremely randomized tree are used as classifiers.
Few applications of code mixed text are also discussed in the thesis. Two applications have been discussed here, i.e. question classification and sentiment analysis. Two approaches have been proposed for classifying code mixed cross script questions. The first approach uses NER and translation based approach for classifying questions in nine different classes whereas the second approach uses CNN for question classification in code mixed text. An approach has also been proposed for code mixed sentiment analysis. Languages are identified in the text using the algorithm proposed for language identification where each word is tagged with a language and transliterated to its original script. Sentiment analysis is then performed using the SentiWordNet dictionaries of different Indian languages. SentiWordNet dictionaries are built using the bilingual mapping for different Indian languages. Three variants of algorithm (SA_Count, SA_Score, SA_RF) are proposed for sentiment analysis where SA_Score outperformed the other mentioned algorithms.

9.2 Thesis Contributions

Main contributions of the thesis are as follows:

1. Sequential neural network based model has been proposed to evaluate the sentiments in Indian languages.

2. An algorithm has been proposed to identify paraphrases in text and improve the process of text summarization.

3. An abstractive text summarization approach has been proposed to summarize monolingual text (including Indian languages) using GAN and sentiment infusion.

4. A hybrid sequential model has been proposed for spam detection using deep learning techniques.

5. An approach for sarcasm detection in code mixed text has been proposed.

6. Algorithms have been proposed for NLP tools such as NER and POS tagger for code mixed text.

7. An approach has been proposed for classifying cross script questions in nine different classes.

8. An approach has been proposed for sentiment analysis of code mixed text for multiple language pairs.

9.3 Future Work

This thesis focuses on different challenges faced while evaluating sentiments for social media text. However, all the aspects have not been covered. Few of them being implicit opinion mining and comparative opinion mining. One interesting future work would be, how important characteristic features for different problems can be embedded in a neural network based approaches to improvise the model further. Few approaches proposed in thesis use translation, though translation and transliteration have not attained a very high accuracy yet. Code mixing is present in social media text in sufficient quantity, but there is a lack of large dataset, which is typically required by deep learning techniques. Code mix text generation tools can generate the synthesized code mix data to solve above mentioned problem. It has been observed that code mixing often follows grammar rules of its own. This can be used to an advantage to improve the algorithms proposed further.

This can also help in building code mix text generation tools. Spam and Sarcasm detection algorithms can be further improved using author profilation. Many studies have shown that Word2Vec and GloVe embeddings work well as compared to word embeddings. However, they are not available for Indian languages and hence, word embeddings using algorithms followed by Word2Vec and GloVe can be proposed for Indian languages as well.

References

[Abburi *et al.* 2016] Harika Abburi, Eswar Sai Akhil Akkireddy, Suryakanth V Gangashetty and Radhika Mamidi. *Multimodal sentiment analysis of telugu songs*. In Proceedings of the 4th Workshop on Sentiment Analysis where AI meets Psychology (SAAIP 2016), pages 48–52, 2016. 24

[Aggarwal *et al.* 2013] Anupama Aggarwal, Jussara Almeida and Ponnurangam Kumaraguru. *Detection of spam tipping behaviour on foursquare*. In Proceedings of the 22nd International Conference on World Wide Web, pages 641–648. ACM, 2013. 44

[Ahmed & Abulaish 2012] Faraz Ahmed and Muhammad Abulaish. *An mcl-based approach for spam profile detection in online social networks*. In 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pages 602–608. IEEE, 2012. 44

[Ajmal & Haroon 2015] EB Ajmal and Rosna P Haroon. *Summarization of Malayalam Document Using Relevance of Sentences*. International Journal of Latest Research in Engineering and Technology, pages 8–13, 2015. 38, 40

[Akoglu *et al.* 2010] Leman Akoglu, Mary McGlohon and Christos Faloutsos. *Oddball: Spotting anomalies in weighted graphs*. Advances in Knowledge Discovery and Data Mining, pages 410–421, 2010. 44, 45

[Al-Rfou *et al.* 2013] Rami Al-Rfou, Bryan Perozzi and Steven Skiena. *Polyglot: Distributed word representations for multilingual nlp*. arXiv preprint arXiv:1307.1662, 2013. 55

[Alguliev *et al.* 2011] Rasim M Alguliev, Ramiz M Aliguliyev, Makrufa S Hajirahimova and Chingiz A Mehdiyev. *MCMR: Maximum coverage and minimum redundant text summarization model*. Expert Systems with Applications, vol. 38, no. 12, pages 14514–14522, 2011. 31

[Algur *et al.* 2011] Siddu P Algur, Amit P Patil, PS Hiremath and S Shivashankar. *WEB BASED CUSTOMER REVIEW SPAM DETECTION USING CONCEPTUAL LEVEL SIMILARITY MEASURE AND SHINGLING TECHNIQUE.* International Journal of Innovative Research in Science and Techniques (IJIRST), pages 1–9, 2011. 149

[Amir *et al.* 2016] Silvio Amir, Byron C Wallace, Hao Lyu and Paula Carvalho Mário J Silva. *Modelling context with user embeddings for sarcasm detection in social media*. arXiv preprint arXiv:1607.00976, 2016. 49, 174

[An *et al.* 2014] Xiaoran An, Auroop R Ganguly, Yi Fang, Steven B Scyphers, Ann M Hunter and Jennifer G Dy. *Tracking climate change opinions from twitter data*. In Workshop on Data Science for Social Good, 2014. 25

[Anagha *et al.* 2016] M Anagha, Raveena R Kuma, K Sreetha and PC Reghu Raj. *A Novel Hybrid Approach Based on Maximum Entropy Classifier for Sentiment Analysis of Malayalam Movie Reviews*. International Journal of Scientific Research, vol. 4, no. 6, pages 1–2, 2016. 19

[Anand Kumar *et al.* 2016] M. Anand Kumar, S. Shiv Karan, Kavirajan and K P Soman. *DPIL@FIRE2016: Overview of shared task on Detecting Paraphrases in Indian Languages*. In Working notes of FIRE 2016 - Forum for Information Retrieval Evaluation, Kolkata, India, December 7-10, 2016, CEUR Workshop Proceedings. CEUR-WS.org, 2016. 28, 107, 112, 118

[Anil *et al.* 2015] Kumar K.M Anil, Poojari Asmita and Kumari M Mohana. *Pattern based Approach for Mining Users Opinion from Kannada Web Documents*. Discovery, vol. 45, pages 138–143, 2015. 19

[Antony & Soman 2011] PJ Antony and KP Soman. *Parts of speech tagging for Indian languages: a literature survey*. International Journal of Computer Applications (0975-8887), vol. 34, no. 8, pages 22–29, 2011. 51

[Araque *et al.* 2017] Oscar Araque, Ignacio Corcuera-Platas, J Fernando Sánchez-Rada and Carlos A Iglesias. *Enhancing deep learning sentiment analysis with ensemble techniques in social applications*. Expert Systems with Applications, vol. 77, pages 236–246, 2017. 13

[Arjovsky & Bottou 2017] Martin Arjovsky and Léon Bottou. *Towards principled methods for training generative adversarial networks*. In NIPS 2016 Workshop on Adversarial Training. In review for ICLR, volume 2016, 2017. 141

[Arthur & Vassilvitskii 2007] David Arthur and Sergei Vassilvitskii. *k-means++: The advantages of careful seeding*. In Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007. 90

[Asiaee T *et al.* 2012] Amir Asiaee T, Mariano Tepper, Arindam Banerjee and Guillermo Sapiro. *If you are happy and you know it... tweet*. In Proceedings of the 21st ACM international conference on Information and knowledge management, pages 1602–1606. ACM, 2012. 12

[Aston *et al.* 2014] Nathan Aston, Jacob Liddle and Wei Hu. *Twitter sentiment in data streams with perceptron*. Journal of Computer and Communications, pages 11–16, 2014. 12

[Baccianella *et al.* 2010a] Stefano Baccianella, Andrea Esuli and Fabrizio Sebastiani. *SentiWordNet 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining.* In LREC, volume 10, pages 2200–2204, 2010. 12

[Baccianella *et al.* 2010b] Stefano Baccianella, Andrea Esuli and Fabrizio Sebastiani. *SentiWordNet 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining.* In LREC, volume 10, pages 2200–2204, 2010. 135

[Baccianella *et al.* 2010c] Stefano Baccianella, Andrea Esuli and Fabrizio Sebastiani. *SentiWordNet 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining.* In LREC, volume 10, pages 2200–2204, 2010. 244

[Bag & Harit 2011] Soumen Bag and Gaurav Harit. *Topographic Feature Extraction for Bengali and Hindi Character Images.* arXiv preprint arXiv:1107.2723, 2011. 76, 80

[Bahdanau *et al.* 2014] Dzmitry Bahdanau, Kyunghyun Cho and Yoshua Bengio. *Neural machine translation by jointly learning to align and translate.* arXiv preprint arXiv:1409.0473, 2014. 34

[Bahdanau *et al.* 2016] Dzmitry Bahdanau, Jan Chorowski, Dmitriy Serdyuk, Philemon Brakel and Yoshua Bengio. *End-to-end attention-based large vocabulary speech recognition*. In Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on, pages 4945–4949. IEEE, 2016. 34

[Balahur *et al.* 2014] Alexandra Balahur, Marco Turchi, Ralf Steinberger, José Manuel Perea Ortega, Guillaume Jacquet, Dilek Küçük, Vanni Zavarella and Adil El Ghali. *Resource Creation and Evaluation for Multilingual Sentiment Analysis in Social Media Texts.* In LREC, pages 4265–4269, 2014. 60

[Balamurali 2012] AR Balamurali. *Cross-lingual sentiment analysis for Indian languages using linked wordnets*. In In proceedings of COLING, pages 73–82. Citeseer, 2012. 20, 58, 60

[Balazs & Velásquez 2016] Jorge A Balazs and Juan D Velásquez. *Opinion mining and information fusion: a survey*. Information Fusion, vol. 27, pages 95–110, 2016. 61

[Bali *et al.* 2014] Kalika Bali, Jatin Sharma, Monojit Choudhury and Yogarshi Vyas. *" I am borrowing ya mixing?" An Analysis of English-Hindi Code Mixing in Facebook.* In Proceedings of the First Workshop on Computational Approaches to Code Switching, pages 116–126, 2014. 171

[Bamman & Smith 2015] David Bamman and Noah A Smith. *Contextualized sarcasm detection on twitter*. In Ninth International AAAI Conference on Web and Social Media, 2015. 46, 47

[Banerjee *et al.* 2015] Siddhartha Banerjee, Prasenjit Mitra and Kazunari Sugiyama. *Multi-document abstractive summarization using ilp based multi-sentence compression*. In Twenty-Fourth International Joint Conference on Artificial Intelligence, pages 1208–1214, 2015. 34

[Banerjee *et al.* 2016a] Somnath Banerjee, Kunal Chakma, Sudip Kumar Naskar, Amitava Das, Paolo Rosso, Sivaji Bandyopadhyay and Monojit Choudhury. *Overview of the Mixed Script Information Retrieval (MSIR) at FIRE*. In Working notes of FIRE 2016 - Forum for Information Retrieval Evaluation, Kolkata, India, December 7-10, 2016, volume 1737 of *CEUR Workshop Proceedings*, pages 94–99. CEUR-WS.org, 2016. 228, 232, 237

[Banerjee *et al.* 2016b] Somnath Banerjee, Sudip Kumar Naskar, Paolo Rosso and Sivaji Bandyopadhyay. *The first cross-script code-mixed question answering corpus*. In Proceedings of the workshop on Modeling, Learning and Mining for Cross/Multilinguality (MultiLingMine 2016), co-located with The 38th European Conference on Information Retrieval (ECIR), pages 56–65, 2016. 57, 225

[Banerjee *et al.* 2016c] Somnath Banerjee, Sudip Kumar Naskar, Paolo Rosso and Sivaji Bandyopadhyay. *The First Cross-Script Code-Mixed Question Answering Corpus.* In MultiLingMine@ ECIR, pages 56–65, 2016. 171

[Banu *et al.* 2007] M Banu, C Karthika, P Sudarmani and TV Geetha. *Tamil Document Summarization Using Semantic Graph Method*. In Conference on Computational Intelligence and Multimedia Applications, 2007. International Conference on, volume 2, pages 128–134. IEEE, 2007. 38, 40

[Barman *et al.* 2016] Utsab Barman, Joachim Wagner and Jennifer Foster. *Part-of-speech Tagging of Code-mixed Social Media Content: Pipeline, Stacking and Joint Modelling*. EMNLP 2016, pages 30–39, 2016. 55

[Barzilay *et al.* 1999] Regina Barzilay, Kathleen R McKeown and Michael Elhadad. *Information fusion in the context of multi-document summarization*. In Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics, pages 550–557. Association for Computational Linguistics, 1999. 29, 102

[Barzilay 2003] Regina Barzilay. *Information fusion for multidocument summarization: paraphrasing and generation*. PhD thesis, Columbia University, 2003. 103

[Baxendale 1958] Phyllis B Baxendale. *Machine-made index for technical literatureâĂŤan experiment*. IBM Journal of Research and Development, vol. 2, no. 4, pages 354–361, 1958. 30

[Becchetti *et al.* 2006] Luca Becchetti, Carlos Castillo, Debora Donato, Stefano Leonardi and Ricardo A Baeza-Yates. *Link-Based Characterization and Detection of Web Spam.* In AIRWeb, pages 1–8, 2006. 44

[Beegum & V.A 2016] Aleena Beegum and Noorjahan V.A. *Sentence level sentiment analysis in Malayalam*. International Journal of Science & Technoledge, vol. 4, pages 106–109, 2016. 19

[Benevenuto *et al.* 2010] Fabricio Benevenuto, Gabriel Magno, Tiago Rodrigues and Virgilio Almeida. *Detecting spammers on twitter*. In Collaboration, electronic messaging, anti-abuse and spam conference (CEAS), volume 6, pages 12–20, 2010. 44, 45

[Bengio *et al.* 2007] Yoshua Bengio, Yann LeCun*et al.* *Scaling learning algorithms towards AI*. Large-scale kernel machines, vol. 34, no. 5, pages 1–41, 2007. 83

[Berant & Liang 2014] Jonathan Berant and Percy Liang. *Semantic Parsing via Paraphrasing.* In ACL (1), pages 1415–1425, 2014. 103

[Bhargava *et al.* 2015] Rupal Bhargava, Yashvardhan Sharma, Shubham Sharma and Abhinav Baid. *Query Labelling for Indic Languages using a hybrid approach*. In Working notes of FIRE 2015 - Forum for Information Retrieval Evaluation, Gandhinagar, India, December, 2015, volume 1587 of *CEUR Workshop Proceedings*, pages 40–42. CEUR-WS.org, 2015. 50, 59, 248

[Bhargava *et al.* 2016a] Rupal Bhargava, Anushka Baoni, Harshit Jain and Yashvardhan Sharma. *Paraphrase Detection in Hindi Language using Syntactic Features of Phrase*. In Working notes of FIRE 2016 - Forum for Information Retrieval Evaluation, Kolkata, India, December 7-10, 2016, CEUR Workshop Proceedings, pages 239–243. CEUR-WS.org, 2016. 28

[Bhargava *et al.* 2016b] Rupal Bhargava, Shubham Khandelwal, Akshit Bhatia and Sharma Yashvardhan. *Modeling Classifier for Code Mixed Cross Script Questions*. In In workshop proceedings of Forum of Information Retrieval and Evaluation 2016, volume 1737 of *CEUR Workshop Proceedings*, pages 109–114. CEUR-WS.org, 2016. 56, 228, 239

[Bhargava *et al.* 2016c] Rupal Bhargava, Yashvardhan Sharma and Gargi Sharma. *ATSSI: Abstractive Text Summarization Using Sentiment Infusion*. Procedia Computer Science, vol. 89, pages 404–411, 2016. 30, 32

[Bhargava *et al.* 2016d] Rupal Bhargava, Yashvardhan Sharma and Shubham Sharma. *Sentiment analysis for mixed script Indic sentences*. In Advances in Computing, Communications and Informatics (ICACCI), 2016 International Conference on, pages 524–529. IEEE, 2016. 50

[Bharti *et al.* 2015] Santosh Kumar Bharti, Korra Sathya Babu and Sanjay Kumar Jena. *Parsing-based sarcasm sentiment recognition in Twitter data*. In Advances in Social Networks Analysis and Mining (ASONAM), 2015 IEEE/ACM International Conference on, pages 1373–1380. IEEE, 2015. 46, 48

[Bhat *et al.* 2016] Irshad Ahmad Bhat, Manish Shrivastava and Riyaz Ahmad Bhat. *Code Mixed Entity Extraction in Indian Languages using Neural Networks*. In Working notes of FIRE 2016 - Forum for Information Retrieval Evaluation, Kolkata, India, December 7-10, 2016, CEUR Workshop Proceedings, pages 296–297. CEUR-WS.org, 2016. 56

[Bhatia & Ritchie 2016] Tej Bhatia and William Ritchie. *Multilingual language mixing and creativity*. Languages, vol. 1, no. 1, page 6, 2016. 171

[Bhattacharjee & Bhattacharya 2016] Debjyoti Bhattacharjee and Paheli Bhattacharya. *Ensemble Classifier based approach for Code-Mixed Cross-Script Question Classification.* In Working notes of FIRE 2016 - Forum for Information Retrieval Evaluation, Kolkata, India, December 7-10, 2016, CEUR Workshop Proceedings, pages 119–121. CEUR-WS.org, 2016. 58, 235

[Bhoir & Gulati 2016] Arti S Bhoir and Archana Gulati. *Multi-document Hindi Text Summarization using Fuzzy Logic Method*. International Journal of Advance Research in Science and Engineering, vol. 4, no. 01, pages 468–476, 2016. 37, 39

[Bifet & Frank 2010] Albert Bifet and Eibe Frank. *Sentiment knowledge discovery in twitter streaming data*. In International Conference on Discovery Science, pages 1–15. Springer, 2010. 25

[Bing *et al.* 2015] Lidong Bing, Piji Li, Yi Liao, Wai Lam, Weiwei Guo and Rebecca J Passonneau. *Abstractive multi-document summarization via phrase selection and merging*. arXiv preprint arXiv:1506.01597, 2015. 34

[Bird & Loper 2004] Steven Bird and Edward Loper. *NLTK: the natural language toolkit*. In Proceedings of the ACL 2004 on Interactive poster and demonstration sessions, page 31. Association for Computational Linguistics, 2004. 163

[Bjørkelund & Burnett 2012] Eivind Bjørkelund and Thomas Hoberg Burnett. Temporal opinion mining. Master's thesis, NTNU-Trondheim Norwegian University of Science and Technology, 2012. 25

[Bock 2013] Zannie Bock. *Cyber socialising: Emerging genres and registers of intimacy among young South African students*. Language Matters, vol. 44, no. 2, pages 68–91, 2013. 49

[Bollen *et al.* 2011] Johan Bollen, Huina Mao and Alberto Pepe. *Modeling public mood and emotion: Twitter sentiment and socio-economic phenomena.* ICWSM, vol. 11, pages 450–453, 2011. 25

[Bosma *et al.* 2012] Maarten Bosma, Edgar Meij and Wouter Weerkamp. *A framework for unsupervised spam detection in social networking sites*. In European Conference on Information Retrieval, pages 364–375. Springer, 2012. 44

[Burstein *et al.* 2001] Jill Burstein, Claudia Leacock and Richard Swartz. *Automated evaluation of essays and short answers*. 2001. 43

[Buschmeier *et al.* 2014] Konstantin Buschmeier, Philipp Cimiano and Roman Klinger. *An impact analysis of features in a classification approach to irony detection in product reviews*. In Proceedings of the 5th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis, pages 42–49, 2014. xv, 47, 168, 169, 170

[Cao *et al.* 2012] Qiang Cao, Michael Sirivianos, Xiaowei Yang and Tiago Pregueiro. *Aiding the detection of fake accounts in large scale social online services*. In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, pages 15–15. USENIX Association, 2012. 44

[Carenini & Cheung 2008] Giuseppe Carenini and Jackie Chi Kit Cheung. *Extractive vs. NLG-based abstractive summarization of evaluative text: The effect of corpus controversiality*. In Proceedings of the Fifth International Natural Language Generation Conference, pages 33–41. Association for Computational Linguistics, 2008. 133

[Carvalho *et al.* 2009] Paula Carvalho, Luís Sarmento, Mário J Silva and Eugénio De Oliveira. *Clues for detecting irony in user-generated contents: oh...!! it's so easy;-*. In Proceedings of the 1st international CIKM workshop on Topic-sentiment analysis for mass opinion, pages 53–56. ACM, 2009. 171

[Castillo *et al.* 2007] Carlos Castillo, Debora Donato, Aristides Gionis, Vanessa Murdock and Fabrizio Silvestri. *Know your neighbors: Web spam detection using the web topology*. In Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, pages 423–430. ACM, 2007. 44

[Cavnar *et al.* 1994] William B Cavnar, John M Trenkle*et al.* *N-gram-based text categorization*. Ann Arbor MI, vol. 48113, no. 2, pages 161–175, 1994. 59

[Ceylan & Kim 2009] Hakan Ceylan and Yookyung Kim. *Language identification of search engine queries*. In Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2, pages 1066–1074, 2009. 59

[Chakraborty *et al.* 2016a] Manajit Chakraborty, Sukomal Pal, Rahul Pramanik and C Ravindranath Chowdary. *Recent developments in social spam detection and combating techniques: A survey*. Information Processing & Management, vol. 52, no. 6, pages 1053–1073, 2016. 4

[Chakraborty *et al.* 2016b] Manajit Chakraborty, Sukomal Pal, Rahul Pramanik and C Ravindranath Chowdary. *Recent developments in social spam detection and combating techniques: A survey*. Information Processing & Management, vol. 52, no. 6, pages 1053–1073, 2016. 4, 5, 44

[Chan 2006] Samuel WK Chan. *Beyond keyword and cue-phrase matching: A sentence-based abstraction technique for information extraction*. Decision Support Systems, vol. 42, no. 2, pages 759–777, 2006. 30

[Chen *et al.* 2015] Changge Chen, Hai Zhao and Yang Yang. *Deceptive Opinion Spam Detection Using Deep Level Linguistic Features*. In Natural Language Processing and Chinese Computing, pages 465–474. Springer, 2015. 152, 154

[Choi 2016] Jinho D Choi. *Dynamic feature induction: The last gist to the state-of-the-art*. In Proceedings of NAACL-HLT, pages 271–281, 2016. 51

[Chollet *et al.* 2015] François Chollet*et al.* *Keras*. `https://github.com/fchollet/keras`, 2015. 154

[Chopra *et al.* 2016] Sumit Chopra, Michael Auli, Alexander M Rush and SEAS Harvard. *Abstractive sentence summarization with attentive recurrent neural networks*. Proceedings of NAACL-HLT16, pages 93–98, 2016. 28

[Choudhury *et al.* 2007] Monojit Choudhury, Rahul Saraf, Vijit Jain, Animesh Mukherjee, Sudeshna Sarkar and Anupam Basu. *Investigation and modeling of the structure of texting language*. International Journal of Document Analysis and Recognition (IJDAR), vol. 10, no. 3-4, pages 157–174, 2007. 171

[Chowdhury & Chowdhury 2014] Shaika Chowdhury and Wasifa Chowdhury. *Performing sentiment analysis in Bangla microblog posts*. In Informatics, Electronics & Vision (ICIEV), 2014 International Conference on, pages 1–6. IEEE, 2014. 15

[Chu *et al.* 2010] Zi Chu, Steven Gianvecchio, Haining Wang and Sushil Jajodia. *Who is tweeting on Twitter: human, bot, or cyborg?* In Proceedings of the 26th annual computer security applications conference, pages 21–30. ACM, 2010. 45

[Chu *et al.* 2012] Zi Chu, Indra Widjaja and Haining Wang. *Detecting social spam campaigns on twitter*. In International Conference on Applied Cryptography and Network Security, pages 455–472. Springer, 2012. 44

[Clarke & Lapata 2006] James Clarke and Mirella Lapata. *Models for sentence compression: A comparison across domains, training requirements and evaluation measures*. In Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics, pages 377–384. Association for Computational Linguistics, 2006. 32

[Cliche 2014] Mathieu Cliche. *The sarcasm detector*. PhD thesis, Cornell University, 2014. 48

[Costa *et al.* 2013] Helen Costa, Fabricio Benevenuto and Luiz HC Merschmann. *Detecting tip spam in location-based social networks*. In Proceedings of the 28th Annual ACM Symposium on Applied Computing, pages 724–729. ACM, 2013. 44

[Da Silva *et al.* 2014] Nadia FF Da Silva, Eduardo R Hruschka and Estevam R Hruschka. *Tweet sentiment analysis with classifier ensembles*. Decision Support Systems, vol. 66, pages 170–179, 2014. 12

[Das & Bandyopadhyay 2009] Amitava Das and Sivaji Bandyopadhyay. *Subjectivity detection in english and bengali: A crf-based approach*. Proceeding of ICON, 2009. 58, 60

[Das & Bandyopadhyay 2010a] Amitava Das and Sivaji Bandyopadhyay. *Opinion-polarity identification in bengali*. In International Conference on Computer Processing of Oriental Languages, pages 169–182, 2010. 16

[Das & Bandyopadhyay 2010b] Amitava Das and Sivaji Bandyopadhyay. *SentiWordNet for Indian languages*. In 8th workshop on asian language resources, pages 56–63, 2010. 60, 243, 252

[Das & Bandyopadhyay 2010c] Amitava Das and Sivaji Bandyopadhyay. *Topic-based Bengali opinion summarization*. In Proceedings of the 23rd International Conference on Computational Linguistics: Posters, pages 232–240. Association for Computational Linguistics, 2010. 42

[Das & Bandyopadhyay 2011] Amitava Das and Sivaji Bandyopadhyay. *Dr Sentiment knows everything!* In Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies: systems demonstrations, pages 50–55. Association for Computational Linguistics, 2011. 60

[Das & Gambäck 2012] Amitava Das and Björn Gambäck. *Sentimantics: conceptual spaces for lexical sentiment polarity representation with contextuality*. In Proceedings of the 3rd Workshop in Computational Approaches to Subjectivity and Sentiment Analysis, pages 38–46. Association for Computational Linguistics, 2012. 60

[Das & Gambäck 2014] Amitava Das and Björn Gambäck. *Identifying languages at the word level in code-mixed indian social media text*. In Proceedings of the 11th International

Conference on Natural Language Processing, Goa, India, pages 169–178. Citeseer, 2014. 6

[Das & Gambäck 2015] Amitava Das and Björn Gambäck. *Code-Mixing in Social Media Text: The Last Language Identification Frontier?* TAL, vol. 54, pages 41–64, 2015. 6

[Das *et al.* 2011] Dipankar Das, Anup Kolya, Asif Ekbal and Sivaji Bandyopadhyay. *Temporal analysis of sentiment events–a visual realization and tracking*. Computational Linguistics and Intelligent Text Processing, pages 417–428, 2011. 25

[Dave *et al.* 2003] Kushal Dave, Steve Lawrence and David M Pennock. *Mining the peanut gallery: Opinion extraction and semantic classification of product reviews*. In Proceedings of the 12th international conference on World Wide Web, pages 519–528. ACM, 2003. 149

[Davidov *et al.* 2010] Dmitry Davidov, Oren Tsur and Ari Rappoport. *Enhanced sentiment learning using twitter hashtags and smileys*. In Proceedings of the 23rd international conference on computational linguistics: posters, pages 241–249. Association for Computational Linguistics, 2010. 12

[Deepali & Garg 2013] Deepali and Navneet Garg. *Movie Review Mining in Punjabi*. International Journal of Application or Innovation in Engineering & Management, vol. 2, pages 372–375, 2013. 20

[Desai & Shah 2016] Nikita Desai and Prachi Shah. *Automatic Text Summarization using Supervised using Supervised Machine Learning Technique for Hindi language*. International Journal of Research in Engineering and Technology, vol. 5, no. 6, pages 361–367, 2016. 37

[Dewaele 2010] J Dewaele. Emotions in multiple languages. Springer, 2010. 50

[Diale *et al.* 2016] Melvin Diale, Christiaan Van Der Walt, Turgay Celik and Abiodun Modupe. *Feature selection and support vector machine hyper-parameter optimisation for spam detection*. In Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference (PRASA-RobMech), 2016, pages 1–7. IEEE, 2016. 45

[Ding *et al.* 2012] Duo Ding, Florian Metze, Shourabh Rawat, Peter Franz Schulam, Susanne Burger, Ehsan Younessian, Lei Bao, Michael G Christel and Alexander Hauptmann. *Beyond audio and video retrieval: towards multimedia summarization*. In Proceedings of the 2nd ACM International Conference on Multimedia Retrieval, pages 2:1–2:8. ACM, 2012. 133

[Dixit & Agrawal 2013] Snehal Dixit and AJ Agrawal. *Survey on review spam detection*. International Journal of Computer & Communication Technology, vol. 4, pages 68–72, 2013. 4

[Dolan *et al.* 2004] Bill Dolan, Chris Quirk and Chris Brockett. *Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources*. In Proceedings of the 20th international conference on Computational Linguistics, pages 350:1–350:7. Association for Computational Linguistics, 2004. 112, 118

[Dos Santos & Gatti 2014] Cícero Nogueira Dos Santos and Maira Gatti. *Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts.* In COLING, pages 69–78, 2014. 69, 71

[Duboue & Chu-Carroll 2006] Pablo Ariel Duboue and Jennifer Chu-Carroll. *Answering the question you wish they had asked: The impact of paraphrasing for question answering*. In Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers, pages 33–36. Association for Computational Linguistics, 2006. 103

[Dunlavy *et al.* 2007] Daniel M Dunlavy, Dianne P OâĂŹLeary, John M Conroy and Judith D Schlesinger. *QCS: A system for querying, clustering and summarizing documents*. Information processing & management, vol. 43, no. 6, pages 1588–1605, 2007. 29

[Edmundson 1969] Harold P Edmundson. *New methods in automatic extracting*. Journal of the ACM (JACM), vol. 16, no. 2, pages 264–285, 1969. 30

[Egele *et al.* 2013] Manuel Egele, Gianluca Stringhini, Christopher Kruegel and Giovanni Vigna. *Compa: Detecting compromised accounts on social networks.* In NDSS, pages 1–17, 2013. 45

[Ekbal *et al.* 2009] Asif Ekbal, Mohammed Hasanuzzaman and Sivaji Bandyopadhyay. *Voted Approach for Part of Speech Tagging in Bengali.* In PACLIC, pages 120–129, 2009. 51

[El-Alfy *et al.* 2015] El-Sayed M El-Alfy, Radwan E Abdel-Aal, Wasfi G Al-Khatib and Faisal Alvi. *Boosting paraphrase detection through textual similarity metrics with abductive networks*. Applied Soft Computing, vol. 26, pages 444–453, 2015. 26

[Elman 1990] Jeffrey L Elman. *Finding structure in time*. Cognitive science, vol. 14, no. 2, pages 179–211, 1990. 63, 65, 69

[Ethiraj *et al.* 2015] Rampreeth Ethiraj, Sampath Shanmugam, Gowri Srinivasa and Navneet Sinha. *NELIS-Named Entity and Language Identification System: Shared Task System Description.* In FIRE Workshops, pages 43–46, 2015. 59

[Fader *et al.* 2013] Anthony Fader, Luke S Zettlemoyer and Oren Etzioni. *Paraphrase-Driven Learning for Open Question Answering.* In ACL (1), pages 1608–1618. Citeseer, 2013. 103

[Fakhraei *et al.* 2015] Shobeir Fakhraei, James Foulds, Madhusudana Shashanka and Lise Getoor. *Collective spammer detection in evolving multi-relational social networks*. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 1769–1778. ACM, 2015. 44

[Fang *et al.* 2015] Hanyin Fang, Weiming Lu, Fei Wu, Yin Zhang, Xindi Shang, Jian Shao and Yueting Zhuang. *Topic aspect-oriented summarization via group selection*. Neurocomputing, vol. 149, pages 1613–1619, 2015. 31

[Fattah 2014] Mohamed Abdel Fattah. *A hybrid machine learning model for multi-document summarization*. Applied intelligence, vol. 40, no. 4, pages 592–600, 2014. 31

[Fei *et al.* 2013] Geli Fei, Arjun Mukherjee, Bing Liu, Meichun Hsu, Malu Castellanos and Riddhiman Ghosh. *Exploiting Burstiness in Reviews for Review Spammer Detection.* ICWSM, vol. 13, pages 175–184, 2013. 44

[Feng *et al.* 2012a] Song Feng, Ritwik Banerjee and Yejin Choi. *Syntactic stylometry for deception detection*. In Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2, pages 171–175. Association for Computational Linguistics, 2012. 45

[Feng *et al.* 2012b] Song Feng, Longfei Xing, Anupam Gogar and Yejin Choi. *Distributional Footprints of Deceptive Product Reviews.* ICWSM, vol. 12, pages 98–105, 2012. 149

[Fernando & Stevenson 2008] Samuel Fernando and Mark Stevenson. *A semantic similarity approach to paraphrase detection*. In Proceedings of the 11th Annual Research Colloquium of the UK Special Interest Group for Computational Linguistics, pages 45–52. Citeseer, 2008. 27

[Ferrara *et al.* 2014] Emilio Ferrara, Onur Varol, Clayton Davis, Filippo Menczer and Alessandro Flammini. *The rise of social bots*. arXiv preprint arXiv:1407.5225, 2014. 45

[Ferreira *et al.* 2013] Rafael Ferreira, Luciano de Souza Cabral, Rafael Dueire Lins, Gabriel Pereira e Silva, Fred Freitas, George DC Cavalcanti, Rinaldo Lima, Steven J Simske and Luciano Favaro. *Assessing sentence scoring techniques for extractive text summarization*. Expert systems with applications, vol. 40, no. 14, pages 5755–5764, 2013. 31

[Filatova 2012] Elena Filatova. *Irony and Sarcasm: Corpus Generation and Analysis Using Crowdsourcing.* In LREC, pages 392–398, 2012. 47, 163, 172

[Fischer 2011] Eric Fischer. *Language communities of Twitter*, 2011. 50

[Fu *et al.* 2015] Hao Fu, Xing Xie and Yong Rui. *Leveraging careful microblog users for spammer detection*. In Proceedings of the 24th International Conference on World Wide Web, pages 419–429. ACM, 2015. 44

[Galavotti *et al.* 2000] Luigi Galavotti, Fabrizio Sebastiani and Maria Simi. *Experiments on the use of feature selection and negative evidence in automated text categorization*. In International Conference on Theory and Practice of Digital Libraries, pages 59–68. Springer, 2000. 40

[Gambäck & Das 2016] Björn Gambäck and Amitava Das. *Comparing the level of code-switching in corpora*. In Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016), pages 1850–1855, 2016. 52

[Ganesan *et al.* 2010a] Kavita Ganesan, ChengXiang Zhai and Jiawei Han. *Opinosis: a graph-based approach to abstractive summarization of highly redundant opinions*. In Proceedings of the 23rd international conference on computational linguistics, pages 340–348. Association for Computational Linguistics, 2010. xi, 32, 33, 137, 138, 139, 140, 144

[Ganesan *et al.* 2010b] Kavita Ganesan, ChengXiang Zhai and Jiawei Han. *Opinosis: a graph-based approach to abstractive summarization of highly redundant opinions*. In Proceedings of the 23rd international conference on computational linguistics, pages 340–348. Association for Computational Linguistics, 2010. 147

[Ganesh *et al.* 2016] Barathi HB Ganesh, Kumar M Anand and KP Soman. *Distributional semantic representation in health care text classification*. In Working notes of FIRE 2016 - Forum for Information Retrieval Evaluation, Kolkata, India, December 7-10, 2016, CEUR Workshop Proceedings, pages 126–130. CEUR-WS.org, 2016. 57

[Gao *et al.* 2014] Hongyu Gao, Yi Yang, Kai Bu, Yan Chen, Doug Downey, Kathy Lee and Alok Choudhary. *Spam ain't as diverse as it seems: throttling OSN spam with templates underneath*. In Proceedings of the 30th Annual Computer Security Applications Conference, pages 76–85. ACM, 2014. 45

[Gella *et al.* 2013] Spandana Gella, Jatin Sharma and Kalika Bali. *Query word labeling and back transliteration for indian languages: Shared task system description*. FIRE Working Notes, vol. 3, pages 1–6, 2013. 52

[Gella *et al.* 2014] Spandana Gella, Kalika Bali and Monojit Choudhury. *âĂIJye word kis lang ka hai bhai?âĂİ Testing the Limits of Word level Language Identification*. In Proceedings of the Eleventh International Conference on Natural Language Processing, pages 130–139, 2014. 60

[Genest & Lapalme 2012] Pierre-Etienne Genest and Guy Lapalme. *Fully abstractive approach to guided summarization*. In Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2, pages 354–358. Association for Computational Linguistics, 2012. 32

[Ghosh & Dutta 2014] Aanusha Ghosh and Indranil Dutta. *Real time Sentiment Analysis of Hindi Tweets*. In 35th Conference of the Linguistic Society of Nepal, pages 1–8, 2014. 18

[Ghosh & Veale 2016] Aniruddha Ghosh and Tony Veale. *Fracking sarcasm using neural network*. In Proceedings of NAACL-HLT, pages 161–169, 2016. 49

[Ghosh *et al.* 2012] Saptarshi Ghosh, Bimal Viswanath, Farshad Kooti, Naveen Kumar Sharma, Gautam Korlam, Fabricio Benevenuto, Niloy Ganguly and Krishna Phani Gummadi. *Understanding and combating link farming in the twitter social network*. In Proceedings of the 21st international conference on World Wide Web, pages 61–70. ACM, 2012. 44, 45

[Ghosh *et al.* 2016] Souvick Ghosh, Satanu Ghosh and Dipankar Das. *Part-of-speech Tagging of Code-Mixed Social Media Text*. Second Workshop on Computational Approaches to Code Switching, pages 90–97, 2016. 54

[Giachanou & Crestani 2016] Anastasia Giachanou and Fabio Crestani. *Like it or not: A survey of twitter sentiment analysis methods*. ACM Computing Surveys (CSUR), vol. 49, no. 2, pages 28:1–28:41, 2016. 12

[Go *et al.* 2009] Alec Go, Richa Bhayani and Lei Huang. *Twitter sentiment classification using distant supervision*. CS224N Project Report, Stanford, vol. 1, no. 12, pages 1–6, 2009. 12

[Goldbarg 2009] Rosalyn Negrón Goldbarg. *Spanish English code switching in Email communication*. Language@ internet, vol. 6, no. 3, pages 1–21, 2009. 49

[Gong & Liu 2001] Yihong Gong and Xin Liu. *Generic text summarization using relevance measure and latent semantic analysis*. In Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval, pages 19–25. ACM, 2001. 29

[Gong *et al.* 2014] Neil Zhenqiang Gong, Mario Frank and Prateek Mittal. *Sybilbelief: A semi-supervised learning approach for structure-based sybil detection*. IEEE Transactions on Information Forensics and Security, vol. 9, no. 6, pages 976–987, 2014. 44

[González-Ibánez *et al.* 2011] Roberto González-Ibánez, Smaranda Muresan and Nina Wacholder. *Identifying sarcasm in Twitter: a closer look*. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers-Volume 2, pages 581–586. Association for Computational Linguistics, 2011. 5, 46, 47

[Goodfellow *et al.* 2014] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville and Yoshua Bengio. *Genera-*

tive adversarial nets. In Advances in neural information processing systems, pages 2672–2680, 2014. 141

[Graham 2002] Paul Graham. *A plan for spam*. http://paulgraham. com/spam. html, 2002. 44

[Graves 2013] Alex Graves. *Generating sequences with recurrent neural networks*. arXiv preprint arXiv:1308.0850, 2013. 74, 76

[Grefenstette 1995] Gregory Grefenstette. *COMPARING TWO LANGUAGE IDENTIFICATION SCHEM ES*. In 3rd International conference on statistical analysis of textual data, pages 1–6, 1995. 59

[Gupta & Kaur 2016] Vishal Gupta and Narvinder Kaur. *A novel hybrid text summarization system for Punjabi text*. Cognitive Computation, vol. 8, no. 2, pages 261–277, 2016. 37

[Gupta *et al.* 2012] Kanika Gupta, Monojit Choudhury and Kalika Bali. *Mining Hindi-English Transliteration Pairs from Online Hindi Lyrics.* In LREC, pages 2459–2465, 2012. 171

[Gupta *et al.* 2014] Parth Gupta, Kalika Bali, Rafael E Banchs, Monojit Choudhury and Paolo Rosso. *Query expansion for mixed-script information retrieval*. In Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval, pages 677–686. ACM, 2014. 50, 60

[Gupta *et al.* 2016a] Deepak Gupta, Shubham Tripathi, Asif Ekbal and Pushpak Bhattacharyya. *A Hybrid Approach for Entity Extraction in Code-Mixed Social Media Data*. In FIRE 2016, pages 298–303. CEUR-WS, 2016. 56

[Gupta *et al.* 2016b] Sakshi Gupta, Piyush Bansal and Radhika Mamidi. *Resource creation for hindi-english code mixed social media text*. In The 4th International Workshop on Natural Language Processing for Social Media in the 25th International Joint Conference on Artificial Intelligence, 2016. 171

[Gupta *et al.* 2017] Deepak Gupta, Shubham Tripathi, Asif Ekbal and Pushpak Bhattacharyya. *SMPOST: Parts of Speech Tagger for Code-Mixed Indic Social Media Text*. arXiv preprint arXiv:1702.00167, 2017. 54

[Gupta 2013] Vishal Gupta. *Hybrid algorithm for multilingual summarization of Hindi and Punjabi documents*. In Mining Intelligence and Knowledge Exploration, pages 717–727. Springer, 2013. 41

[Gyöngyi *et al.* 2004] Zoltán Gyöngyi, Hector Garcia-Molina and Jan Pedersen. *Combating web spam with trustrank*. In Proceedings of the Thirtieth international conference on Very large data bases-Volume 30, pages 576–587. VLDB Endowment, 2004. 44

[Hakkani-Tür *et al.* 2016] Dilek Hakkani-Tür, Gökhan Tür, Asli Celikyilmaz, Yun-Nung Chen, Jianfeng Gao, Li Deng and Ye-Yi Wang. *Multi-Domain Joint Semantic Frame Parsing Using Bi-Directional RNN-LSTM.* In INTERSPEECH, pages 715–719, 2016. 74

[Halácsy *et al.* 2007] Péter Halácsy, András Kornai and Csaba Oravecz. *HunPos: an open source trigram tagger*. In Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions, pages 209–212. Association for Computational Linguistics, 2007. 54

[Hamdan *et al.* 2013] Hussam Hamdan, Frederic Béchet and Patrice Bellot. *Experiments with DBpedia, WordNet and SentiWordNet as resources for sentiment analysis in micro-blogging*. In Second Joint Conference on Lexical and Computational Semantics (* SEM), volume 2, pages 455–459, 2013. 12

[Hao *et al.* 2011] Ming Hao, Christian Rohrdantz, Halldór Janetzko, Umeshwar Dayal, Daniel A Keim, Lars-Erik Haug and Mei-Chun Hsu. *Visual sentiment analysis on twitter data streams*. In Visual Analytics Science and Technology (VAST), 2011 IEEE Conference on, pages 277–278. IEEE, 2011. 25

[Harabagiu & Lacatusu 2005] Sanda Harabagiu and Finley Lacatusu. *Topic themes for multi-document summarization*. In Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval, pages 202–209. ACM, 2005. 30

[Hasan *et al.* 2014] KM Azharul Hasan, Mosiur Rahman*et al.* *Sentiment detection from Bangla text using contextual valency analysis*. In Computer and Information Technology (ICCIT), 2014 17th International Conference on, pages 292–295. IEEE, 2014. 15

[Hassan & Mahmood 2017] Abdalraouf Hassan and Ausif Mahmood. *Deep Learning approach for sentiment analysis of short texts*. In Control, Automation and Robotics (ICCAR), 2017 3rd International Conference on, pages 705–710. IEEE, 2017. 13

[Hassan *et al.* 2013] Ammar Hassan, Ahmed Abbasi and Daniel Zeng. *Twitter sentiment analysis: A bootstrap ensemble framework*. In Social Computing (SocialCom), 2013 International Conference on, pages 357–364. IEEE, 2013. 12

[He *et al.* 2015] Hua He, Kevin Gimpel and Jimmy Lin. *Multi-Perspective Sentence Similarity Modeling with Convolutional Neural Networks*. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 1576–1586, Lisbon, Portugal, September 2015. Association for Computational Linguistics. 27, 104

[Heu *et al.* 2015] Jee-Uk Heu, Iqbal Qasim and Dong-Ho Lee. *FoDoSu: Multi-document summarization exploiting semantic analysis based on social Folksonomy*. Information Processing & Management, vol. 51, no. 1, pages 212–225, 2015. 31

[Hochreiter & Schmidhuber 1997] Sepp Hochreiter and Jürgen Schmidhuber. *Long short-term memory*. Neural computation, vol. 9, no. 8, pages 1735–1780, 1997. 63, 65, 69

[Hovold 2005] Johan Hovold. *Naive Bayes Spam Filtering Using Word-Position-Based Attributes.* In CEAS, pages 41–48, 2005. 44

[Hu & Liu 2004] Minqing Hu and Bing Liu. *Mining and summarizing customer reviews*. In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 168–177. ACM, 2004. 2, 163

[Hu *et al.* 2013] Xia Hu, Jiliang Tang, Yanchao Zhang and Huan Liu. *Social spammer detection in microblogging*. In Twenty-Third International Joint Conference on Artificial Intelligence, pages 2633–2639, 2013. 44

[Huang *et al.* 2008] Zhiheng Huang, Marcus Thint and Zengchang Qin. *Question classification using head words and their hypernyms*. In Proceedings of the Conference on Empirical Methods in Natural Language Processing, pages 927–936. Association for Computational Linguistics, 2008. 56

[Huang 2011a] Eric Huang. *Paraphrase detection using recursive autoencoder*. Source:[http://nlp. stanford. edu/courses/cs224n/2011/reports/ehhuang. pdf], pages 1–8, 2011. 26

[Huang 2011b] Eric Huang. *Paraphrase detection using recursive autoencoder*. Source:[http://nlp. stanford. edu/courses/cs224n/2011/reports/ehhuang. pdf], 2011. 149

[Ingle 1976] C Ingle Norman. A language identification table. The Incorporated Linguist 15(4), 1976. 59

[Islam & Inkpen 2008] Aminul Islam and Diana Inkpen. *Semantic text similarity using corpus-based word similarity and string similarity*. ACM Transactions on Knowledge Discovery from Data (TKDD), vol. 2, no. 2, pages 10:1–10:25, 2008. 26

[Islam & Masum 2004] T Islam and SMA Masum. *Bhasa: A Corpus Based Information Retrieval and Summarizer for Bengali Text*. Macquarie University, 2004. 42

[Jain 2015] Devanshu Jain. *DA-IICT in FIRE 2015 Shared Task on Mixed Script Information Retrieval.* In FIRE Workshops, pages 51–54, 2015. 59

[Jamatia & Das 2014] Anupam Jamatia and Amitava Das. *Part-of-speech tagging system for Indian social media text on Twitter*. In Proceedings Workshop on Language Technologies For Indian Social Media (SOCIAL-INDIA), pages 21–28, 2014. 51

[Jamatia & Das 2016] Anupam Jamatia and Amitava Das. *TASK REPORT: TOOL CONTEST ON POS TAGGING FOR CODE-MIXED INDIAN SOCIAL MEDIA (FACEBOOK, TWITTER, AND WHATSAPP) TEXT @ ICON 2016*. In Proceedings of 13th International Conference on Natural Language Processing,ICON 2016, 2016. xiii, 194, 201, 203, 204

[Jamatia *et al.* 2015] Anupam Jamatia, Björn Gambäck and Amitava Das. *Part-of-Speech Tagging for Code-Mixed English-Hindi Twitter and Facebook Chat Messages.* In RANLP, pages 239–248, 2015. 52

[Jayashree *et al.* 2011] R Jayashree, Srikanta K Murthy and K Sunny. *Keyword Extraction Based Summarization of Categorized Kannada Text Documents*. International Journal on Soft Computing, vol. 2, no. 4, pages 81–93, 2011. 37, 40

[Jeong *et al.* 2016] Sihyun Jeong, Giseop Noh, Hayoung Oh and Chong-kwon Kim. *Follow spam detection based on cascaded social information*. Information Sciences, vol. 369, pages 481–499, 2016. 44, 45

[Jiang *et al.* 2011] Long Jiang, Mo Yu, Ming Zhou, Xiaohua Liu and Tiejun Zhao. *Target-dependent twitter sentiment classification*. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1, pages 151–160. Association for Computational Linguistics, 2011. 12, 13

[Jiang *et al.* 2014] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos and Shiqiang Yang. *Catchsync: catching synchronized behavior in large directed graphs*. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 941–950. ACM, 2014. 44, 45

[Jin *et al.* 2011] Xin Jin, C Lin, Jiebo Luo and Jiawei Han. *A data mining-based spam detection system for social media networks*. Proceedings of the VLDB Endowment, vol. 4, no. 12, pages 1458–1461, 2011. 44

[Jindal & Liu 2008] Nitin Jindal and Bing Liu. *Opinion spam and analysis*. In Proceedings of the 2008 International Conference on Web Search and Data Mining, pages 219–230. ACM, 2008. 45, 148

[Jones *et al.* 1999] K Sparck Jones*et al.* *Automatic summarizing: factors and directions*. Advances in automatic text summarization, pages 1–12, 1999. 3

[Joshi *et al.* 2010] Aditya Joshi, AR Balamurali and Pushpak Bhattacharyya. *A fall-back strategy for sentiment analysis in hindi: a case study*. Proceedings of ICON, pages 1–6, 2010. 58, 59

[Joshi *et al.* 2016a] Aditya Joshi, Pushpak Bhattacharyya and Mark James Carman. *Automatic sarcasm detection: A survey*. arXiv preprint arXiv:1602.03426, 2016. 162

[Joshi *et al.* 2016b] Aditya Joshi, Vaibhav Tripathi, Pushpak Bhattacharyya and Mark Carman. *Harnessing sequence labeling for sarcasm detection in dialogue from tv series 'friends'*. CoNLL 2016, pages 146–155, 2016. 48

[Joshi *et al.* 2016c] Aditya Joshi, Vaibhav Tripathi, Kevin Patel, Pushpak Bhattacharyya and Mark Carman. *Are Word Embedding-based Features Useful for Sarcasm Detection?* arXiv preprint arXiv:1610.00883, 2016. 48

[Kadlec *et al.* 2015] Rudolf Kadlec, Martin Schmid and Jan Kleindienst. *Improved deep learning baselines for ubuntu corpus dialogs*. arXiv preprint arXiv:1510.03753, 2015. 71

[Kalchbrenner & Blunsom 2013] Nal Kalchbrenner and Phil Blunsom. *Recurrent convolutional neural networks for discourse compositionality*. arXiv preprint arXiv:1306.3584, 2013. 57

[Kalchbrenner *et al.* 2014] Nal Kalchbrenner, Edward Grefenstette and Phil Blunsom. *A convolutional neural network for modelling sentences*. arXiv preprint arXiv:1404.2188, 2014. 27

[Kallimani *et al.* 2010] Jagadish S Kallimani, KG Srinivasa*et al.* *Information retrieval by text summarization for an Indian regional language*. In Natural Language Processing and Knowledge Engineering (NLP-KE), 2010 International Conference on, pages 1–4. IEEE, 2010. 39

[Kallimani *et al.* 2016] Jagadish S Kallimani, KG Srinivasa and B Eswara Reddy. *Statistical and analytical study of guided abstractive text summarization*. CURRENT SCIENCE, vol. 110, no. 1, pages 69–72, 2016. 37

[Kantchelian *et al.* 2012] Alex Kantchelian, Justin Ma, Ling Huang, Sadia Afroz, Anthony Joseph and JD Tygar. *Robust detection of comment spam using entropy rate*. In Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence, pages 59–70. ACM, 2012. 44

[Kaur & Gupta 2014] Amandeep Kaur and Vishal Gupta. *Proposed algorithm of sentiment analysis for punjabi text*. Journal of Emerging Technologies in Web Intelligence, vol. 6, no. 2, pages 180–184, 2014. 21

[Kausikaa & V 2016] N Kausikaa and Uma V. *Sentiment Analysis of English and Tamil Tweets using Path Length Similarity based Word Sense Disambiguation*. IOSR Journals (IOSR Journal of Computer Engineering), vol. 1, pages 82–89, 2016. 22

[Kayes *et al.* 2015] Imrul Kayes, Nicolas Kourtellis, Daniele Quercia, Adriana Iamnitchi and Francesco Bonchi. *The social world of content abusers in community question answering*. In Proceedings of the 24th International Conference on World Wide Web, pages 570–580. ACM, 2015. 44

[Keyan & Srinivasagan 2012] M Karthi Keyan and KG Srinivasagan. *Multi-Document and Multi-Lingual Summarization using Neural Networks*. In Proceedings of International Conference on Recent Trends in Computational Methods, Communication and Controls, pages 11–14, 2012. 38, 41

[Khan & Baharum 2011] Aurangzeb Khan and Baharudin Baharum. *Sentiment classification by sentence level semantic orientation using sentiwordnet from online reviews and Blogs*. International Journal of Computer Science & Emerging Technologies, vol. 2, no. 4, pages 539–552, 2011. 18, 19

[Khan *et al.* 2015] Atif Khan, Naomie Salim and Yogan Jaya Kumar. *A framework for multi-document abstractive summarization based on semantic role labelling*. Applied Soft Computing, vol. 30, pages 737–747, 2015. 33

[Kim *et al.* 2016] Jong Myoung Kim, Zae Myung Kim and Kwangjo Kim. *An approach to spam comment detection through domain-independent features*. In Big Data and Smart Computing (BigComp), 2016 International Conference on, pages 273–276. IEEE, 2016. 46

[Kim 2014] Yoon Kim. *Convolutional neural networks for sentence classification*. arXiv preprint arXiv:1408.5882, 2014. 57, 63, 65, 69

[Knight & Marcu 2002] Kevin Knight and Daniel Marcu. *Summarization beyond sentence extraction: A probabilistic approach to sentence compression*. Artificial Intelligence, vol. 139, no. 1, pages 91–107, 2002. 32

[Ko & Seo 2008] Youngjoong Ko and Jungyun Seo. *An effective sentence-extraction technique using contextual information and statistical approaches for text summarization*. Pattern Recognition Letters, vol. 29, no. 9, pages 1366–1371, 2008. 30

[Kong *et al.* 2016] Leilei Kong, Kaisheng Chen, Liuyang Tian, Zhenyuan Hao, Zhongyuan Han and Haoliang Qi. *HIT2016@ DPIL-FIRE2016: Detecting Paraphrases in Indian Languages based on Gradient Tree Boosting.* In Working notes of FIRE 2016 - Forum for Information Retrieval Evaluation, Kolkata, India, December 7-10, 2016, CEUR Workshop Proceedings, pages 260–265. CEUR-WS.org, 2016. 28

[Krishnan & Raj 2006] Vijay Krishnan and Rashmi Raj. *Web spam detection with anti-trust rank.* In AIRWeb, volume 6, pages 37–40, 2006. 44

[Krueger *et al.* 2015] Tammo Krueger, Danny Panknin and Mikio Braun. *Fast cross-validation via sequential testing*. The Journal of Machine Learning Research, vol. 16, no. 1, pages 1103–1155, 2015. 82

[Kumar & Yadav 2015] K Vimal Kumar and Divakar Yadav. *An Improvised Extractive Approach to Hindi Text Summarization.* In Information Systems Design and Intelligent Applications, pages 291–300. Springer, 2015. 39

[Kumar *et al.* 2013] Niraj Kumar, Kannan Srinathan and Vasudeva Varma. *A knowledge induced graph-theoretical model for extract and abstract single document summarization*. In International Conference on Intelligent Text Processing and Computational Linguistics, pages 408–423. Springer, 2013. 133

[Kumar *et al.* 2015a] Ayush Kumar, Sarah Kohail, Asif Ekbal and Chris Biemann. *IIT-TUDA: System for sentiment analysis in indian languages using lexical acquisition*. In International Conference on Mining Intelligence and Knowledge Exploration, pages 684–693. Springer, 2015. 82, 83

[Kumar *et al.* 2015b] KM Anil Kumar, N Rajasimha, Manovikas Reddy, A Rajanarayana and Kewal Nadgir. *Analysis of users' Sentiments from Kannada Web Documents*. Procedia Computer Science, vol. 54, pages 247–256, 2015. 18, 19, 58, 60

[Kumar *et al.* 2015c] M Anand Kumar, S Rajendran and KP Soman. *Cross-Lingual Preposition Disambiguation for Machine Translation*. Procedia Computer Science, vol. 54, pages 291–300, 2015. 58, 60

[Kumar *et al.* 2015d] Rahul Venkatesh Kumar, M Anand Kumar and KP Soman. *AmritaCEN_NLP@ FIRE 2015 Language Identification for Indian Languages in Social Media Text.* In FIRE Workshops, pages 26–28, 2015. 59

[Lau *et al.* 2011] Raymond YK Lau, SY Liao, Ron Chi-Wai Kwok, Kaiquan Xu, Yunqing Xia and Yuefeng Li. *Text mining and probabilistic language modeling for online review spam detection*. ACM Transactions on Management Information Systems (TMIS), vol. 2, no. 4, pages 25:1–25:30, 2011. 4

[Leacock & Chodorow 1998] Claudia Leacock and Martin Chodorow. *Combining local context and WordNet similarity for word sense identification*. WordNet: An electronic lexical database, vol. 49, no. 2, pages 265–283, 1998. 104

[Lee *et al.* 2009] Ju-Hong Lee, Sun Park, Chan-Min Ahn and Daeho Kim. *Automatic generic document summarization based on non-negative matrix factorization*. Information Processing & Management, vol. 45, no. 1, pages 20–34, 2009. 31

[Lee *et al.* 2011] Kyumin Lee, Brian David Eoff and James Caverlee. *Seven Months with the Devils: A Long-Term Study of Content Polluters on Twitter.* In ICWSM, pages 185–192, 2011. 45

[Lei *et al.* 2015] Tao Lei, Regina Barzilay and Tommi Jaakkola. *Molding cnns for text: non-linear, non-consecutive convolutions.* arXiv preprint arXiv:1508.04112, 2015. 71

[Li & Roth 2002] Xin Li and Dan Roth. *Learning question classifiers.* In Proceedings of the 19th international conference on Computational linguistics-Volume 1, pages 1–7. Association for Computational Linguistics, 2002. 57, 225

[Li *et al.* 2014] Jiwei Li, Myle Ott, Claire Cardie and Eduard H Hovy. *Towards a General Rule for Identifying Deceptive Opinion Spam.* In ACL (1), pages 1566–1576. Citeseer, 2014. 149, 152

[Li *et al.* 2015] Huayi Li, Zhiyuan Chen, Arjun Mukherjee, Bing Liu and Jidong Shao. *Analyzing and Detecting Opinion Spam on a Large-scale Dataset via Temporal and Spatial Patterns.* In ICWSM, pages 634–637, 2015. 149

[Li *et al.* 2017a] Huayi Li, Geli Fei, Shuai Wang, Bing Liu, Weixiang Shao, Arjun Mukherjee and Jidong Shao. *Bimodal distribution and co-bursting in review spam detection.* In Proceedings of the 26th International Conference on World Wide Web, pages 1063–1072. International World Wide Web Conferences Steering Committee, 2017. 46

[Li *et al.* 2017b] Lingxiao Li, Shaozi Li, Donglin Cao and Dazhen Lin. *SentiNet: Mining Visual Sentiment from Scratch.* In Advances in Computational Intelligence Systems, pages 309–317. Springer, 2017. 164

[Li *et al.* 2017c] Luyang Li, Bing Qin, Wenjing Ren and Ting Liu. *Document representation and feature combination for deceptive spam review detection.* Neurocomputing, pages 33–41, 2017. 46

[Li 2000] David Li. *Cantonese-English code-switching research in Hong Kong: a Y2K review.* World Englishes, vol. 19, no. 3, pages 305–322, 2000. 49

[Lin & Hovy 2003] Chin-Yew Lin and Eduard Hovy. *Automatic evaluation of summaries using n-gram co-occurrence statistics.* In Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1, pages 71–78. Association for Computational Linguistics, 2003. 137

[Lin & Kolcz 2012] Jimmy Lin and Alek Kolcz. *Large-scale machine learning at twitter.* In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, pages 793–804. ACM, 2012. 12

[Lin *et al.* 2014] Yuming Lin, Tao Zhu, Hao Wu, Jingwei Zhang, Xiaoling Wang and Aoying Zhou. *Towards online anti-opinion spam: Spotting fake reviews from the review sequence*. In Advances in Social Networks Analysis and Mining (ASONAM), 2014 IEEE/ACM International Conference on, pages 261–264. IEEE, 2014. 44

[Lin 2004] Chin-Yew Lin. *Rouge: A package for automatic evaluation of summaries*. In Text summarization branches out: Proceedings of the ACL-04 workshop, volume 8, pages 1–8. Barcelona, Spain, 2004. 41, 137, 138, 146

[Ling *et al.* 2013] Wang Ling, Guang Xiang, Chris Dyer, Alan W Black and Isabel Trancoso. *Microblogs as Parallel Corpora*. In ACL (1), pages 176–186, 2013. 6

[Litvak & Last 2008] Marina Litvak and Mark Last. *Graph-based keyword extraction for single-document summarization*. In Proceedings of the workshop on Multi-source Multilingual Information Extraction and Summarization, pages 17–24. Association for Computational Linguistics, 2008. 29, 102

[Liu & Liu 2009] Fei Liu and Yang Liu. *From extractive to abstractive meeting summaries: Can it be done by sentence compression?* In Proceedings of the ACL-IJCNLP 2009 Conference Short Papers, pages 261–264. Association for Computational Linguistics, 2009. 139

[Liu & Zhang 2012] Bing Liu and Lei Zhang. *A survey of opinion mining and sentiment analysis*. In Mining text data, pages 415–463. Springer, 2012. 11

[Liu *et al.* 2014] Peng Liu, Wei Chen, Gaoyan Ou, Tengjiao Wang, Dongqing Yang and Kai Lei. *Sarcasm detection in social media based on imbalanced classification*. In International Conference on Web-Age Information Management, pages 459–471. Springer, 2014. 48

[Liu *et al.* 2015] Fei Liu, Jeffrey Flanigan, Sam Thomson, Norman Sadeh and Noah A Smith. *Toward abstractive summarization using semantic representations*. 2015. 32, 133

[Liu *et al.* 2016] Linqing Liu, Yao Lu, Ye Luo, Renxian Zhang, Laurent Itti and Jianwei Lu. *Detecting" Smart" Spammers On Social Network: A Topic Model Approach*. arXiv preprint arXiv:1604.08504, 2016. 45

[Liu 2012] Bing Liu. *Sentiment analysis and opinion mining*. Synthesis lectures on human language technologies, vol. 5, no. 1, pages 1–167, 2012. 1

[Lloret & Palomar 2011] Elena Lloret and Manuel Palomar. *Analyzing the use of word graphs for abstractive text summarization*. In Proceedings of the First International Conference on Advances in Information Mining and Management, Barcelona, Spain, pages 61–6, 2011. 32, 33, 138, 139, 147

[Lloret & Palomar 2013] Elena Lloret and Manuel Palomar. *Tackling redundancy in text summarization through different levels of language analysis*. Computer Standards & Interfaces, vol. 35, no. 5, pages 507–518, 2013. 31

[Lloret *et al.* 2013] Elena Lloret, María Teresa Romá-Ferri and Manuel Palomar. *COMPENDIUM: A text summarization system for generating abstracts of research papers*. Data & Knowledge Engineering, vol. 88, pages 164–175, 2013. 33

[Lloret 2008] Elena Lloret. *Text summarization: an overview*. Paper supported by the Spanish Government under the project TEXT-MESS (TIN2006-15265-C06-01), 2008. 3

[Lopez & Kalita 2017] Marc Moreno Lopez and Jugal Kalita. *Deep Learning applied to NLP*. arXiv preprint arXiv:1703.03091, 2017. 13

[Lopyrev 2015] Konstantin Lopyrev. *Generating news headlines with recurrent neural networks*. arXiv preprint arXiv:1512.01712, 2015. 34, 102

[Luhn 1958] Hans Peter Luhn. *The automatic creation of literature abstracts*. IBM Journal of research and development, vol. 2, no. 2, pages 159–165, 1958. 28, 30

[Lumezanu & Feamster 2012] Cristian Lumezanu and Nick Feamster. *Observing common spam in Twitter and email*. In Proceedings of the 2012 ACM conference on Internet measurement conference, pages 461–466. ACM, 2012. 44

[Luong *et al.* 2015] Thang Luong, Hieu Pham and Christopher D Manning. *Bilingual word representations with monolingual quality in mind*. In Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing, pages 151–159, 2015. 178

[M & Soman 2016] Anand Kumar M and KP Soman. *Amrita_CEN@ MSIR-FIRE2016: Code-Mixed Question Classification using BoWs and RNN Embeddings*. In Working notes of FIRE 2016 - Forum for Information Retrieval Evaluation, Kolkata, India, December 7-10, 2016, CEUR Workshop Proceedings, pages 122–125. CEUR-WS.org, 2016. 57

[Maas *et al.* 2011] Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng and Christopher Potts. *Learning word vectors for sentiment analysis*. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1, pages 142–150. Association for Computational Linguistics, 2011. 69

[Majumder & Pakray 2016] Goutam Majumder and Partha Pakray. *NLP-NITMZ@ MSIR 2016 System for Code-Mixed Cross-Script Question Classification.* In Working notes of FIRE 2016 - Forum for Information Retrieval Evaluation, Kolkata, India, December 7-10, 2016, CEUR Workshop Proceedings, pages 100–104. CEUR-WS.org, 2016. 58, 237

[Malakasiotis 2009] Prodromos Malakasiotis. *Paraphrase recognition using machine learning to combine similarity measures*. In Proceedings of the ACL-IJCNLP 2009 Student Research Workshop, pages 27–35. Association for Computational Linguistics, 2009. 26, 104

[Malarkodi *et al.* 2012] CS Malarkodi, RK Pattabhi and Lalitha Devi Sobha. *Tamil ner–coping with real time challenges*. In 24th International Conference on Computational Linguistics, pages 23–37, 2012. 55

[Manchala *et al.* 2012] Sadanandam Manchala, D Chandra Mohan and A Nagesh. *Word and Sentence Level Emotion Analyzation in Telugu Blog and News*. International Journal of Computer Science, Engineering and Applications, vol. 2, pages 183–197, 2012. 23

[Mani & Maybury 1999] Inderjeet Mani and Mark T Maybury. Advances in automatic text summarization, volume 293. MIT Press, 1999. 29

[Mani *et al.* 1999] Inderjeet Mani, David House, Gary Klein, Lynette Hirschman, Therese Firmin and Beth Sundheim. *The TIPSTER SUMMAC text summarization evaluation*. In Proceedings of the ninth conference on European chapter of the Association for Computational Linguistics, pages 77–85. Association for Computational Linguistics, 1999. 3

[Mani *et al.* 2002] Inderjeet Mani, Gary Klein, David House, Lynette Hirschman, Therese Firmin and Beth Sundheim. *SUMMAC: a text summarization evaluation*. Natural Language Engineering, vol. 8, no. 01, pages 43–68, 2002. 42

[Märkle-Huß *et al.* 2017] Joscha Märkle-Huß, Stefan Feuerriegel and Helmut Prendinger. *Improving Sentiment Analysis with Document-Level Semantic Relationships from Rhetoric Discourse Structures*. In Proceedings of the 50th Hawaii International Conference on System Sciences, pages 1142–1151, 2017. 13

[Martinez-Romo & Araujo 2013] Juan Martinez-Romo and Lourdes Araujo. *Detecting malicious tweets in trending topics using a statistical analysis of language*. Expert Systems with Applications, vol. 40, no. 8, pages 2992–3000, 2013. 44, 45

[Maynard & Greenwood 2014] Diana Maynard and Mark A Greenwood. *Who cares about Sarcastic Tweets? Investigating the Impact of Sarcasm on Sentiment Analysis.* In LREC, pages 4238–4243, 2014. 46, 48, 171

[Medhat *et al.* 2014] Walaa Medhat, Ahmed Hassan and Hoda Korashy. *Sentiment analysis algorithms and applications: A survey*. Ain Shams Engineering Journal, vol. 5, no. 4, pages 1093–1113, 2014. 1

[Merriam-Webster 1983] Inc Merriam-Webster. Webster's ninth new collegiate dictionary. Merriam-Webster, 1983. 161

[Mi *et al.* 2015] Guyue Mi, Yang Gao and Ying Tan. *Apply stacked auto-encoder to spam detection*. In International Conference in Swarm Intelligence, pages 3–15. Springer, 2015. 45

[Mihalcea *et al.* 2006] Rada Mihalcea, Courtney Corley, Carlo Strapparava*et al. Corpus-based and knowledge-based measures of text semantic similarity*. In AAAI, volume 6, pages 775–780, 2006. 26, 104

[Mihalcea *et al.* 2007] Rada Mihalcea, Carmen Banea and Janyce Wiebe. *Learning multilingual subjective language via cross-lingual projections*. In Annual meeting-association for computational linguistics, volume 45, pages 976–983, 2007. 60

[Mikolov *et al.* 2013a] Tomas Mikolov, Kai Chen, Greg Corrado and Jeffrey Dean. *Efficient estimation of word representations in vector space*. arXiv preprint arXiv:1301.3781, 2013. 74, 175

[Mikolov *et al.* 2013b] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado and Jeff Dean. *Distributed representations of words and phrases and their compositionality*. In Advances in neural information processing systems, pages 3111–3119, 2013. 113, 117, 124

[Mishra *et al.* 2017] Abhijit Mishra, Diptesh Kanojia, Seema Nagar, Kuntal Dey and Pushpak Bhattacharyya. *Harnessing Cognitive Features for Sarcasm Detection*. arXiv preprint arXiv:1701.05574, 2017. 46, 49

[Mittal *et al.* 2013] Namita Mittal, Basant Agarwal, Garvit Chouhan, Nitin Bania and Prateek Pareek. *Sentiment analysis of hindi review based on negation and discourse relation*. In proceedings of International Joint Conference on Natural Language Processing, pages 45–50, 2013. 17

[Moawad & Aref 2012] Ibrahim F Moawad and Mostafa Aref. *Semantic graph reduction approach for abstractive Text Summarization*. In Computer Engineering & Systems (ICCES), 2012 Seventh International Conference on, pages 132–138. IEEE, 2012. 33

[Mohanty 1998] SK Mohanty. *The formulation of parameters for type design of Indian scripts based on calligraphic studies*. In Electronic Publishing, Artistic Imaging, and Digital Typography, pages 157–166. Springer, 1998. 79, 80

[Moradi 2014] Hamzeh Moradi. *A Survey on Code-Mixing, Code Switching, Language Alteration and Interference*. International Journal of Applied Research, vol. 4, no. 10, pages 1–3, 2014. 6

[Muhammad *et al.* 2016] Mahmudun Nabi Muhammad, Tanzir Altaf Md. and Sabir Ismail. *Detecting Sentiment from Bangla Text using Machine Learning Technique and Feature Analysis*. International Journal of Computer Applications, vol. 153, pages 28–34, 2016. 16

[Mukherjee *et al.* 2013a] Arjun Mukherjee, Abhinav Kumar, Bing Liu, Junhui Wang, Meichun Hsu, Malu Castellanos and Riddhiman Ghosh. *Spotting opinion spammers using behavioral footprints*. In Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 632–640. ACM, 2013. 45

[Mukherjee *et al.* 2013b] Arjun Mukherjee, Vivek Venkataraman, Bing Liu and Natalie Glance. *Fake review detection: Classification and analysis of real and pseudo reviews*. Technical Report UIC-CS-2013–03, University of Illinois at Chicago, Tech. Rep., pages 1–11, 2013. 44

[Mukku *et al.* 2016] Sandeep Sricharan Mukku, Nurendra Choudhary and Radhika Mamidi. *Enhanced Sentiment Classification of Telugu Text using ML Techniques.* In SAAIP@ IJCAI, pages 29–34, 2016. 23

[Nallapati *et al.* 2016] Ramesh Nallapati, Bing Xiang and Bowen Zhou. *TEXT SUMMARIZATION*. arXiv preprint arXiv:1602.06023, 2016. 34, 102

[Nelakuditi *et al.* 2016] Kovida Nelakuditi, Divya Sai Jitta Jitta and Radhika Mamidi. *Part-of-Speech Tagging for Code mixed English-Telugu Social media data*. In 17th International Conference on Intelligent Text Processing and Computational Linguistics (CICLING 2016), 2016. 54

[Nguyen *et al.* 2014] Dat Quoc Nguyen, Dang Duc Pham Dai Quoc Nguyen and Son Bao Pham. *RDRPOSTagger: A ripple down rules-based part-of-speech tagger*. In Proceedings of the Demonstrations at the 14th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2014, pages 17–20. Citeseer, 2014. 105, 110

[O'Callaghan *et al.* 2012] Derek O'Callaghan, Martin Harrigan, Joe Carthy and Pádraig Cunningham. *Network analysis of recurring youtube spam campaigns*. arXiv preprint arXiv:1201.3783, 2012. 44, 45

[O'Connor *et al.* 2010] Brendan O'Connor, Ramnath Balasubramanyan, Bryan R Routledge and Noah A Smith. *From tweets to polls: Linking text sentiment to public opinion time series.* ICWSM, vol. 11, no. 122-129, pages 1–2, 2010. 25

[Oraby *et al.* 2016] Shereen Oraby, Vrindavan Harrison, Lena Reed, Ernesto Hernandez, Ellen Riloff and Marilyn Walker. *Creating and Characterizing a Diverse Corpus of Sarcasm in Dialogue*. In 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue, pages 31–41, 2016. 172

[Ott *et al.* 2011] Myle Ott, Yejin Choi, Claire Cardie and Jeffrey T Hancock. *Finding deceptive opinion spam by any stretch of the imagination*. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language

Technologies-Volume 1, pages 309–319. Association for Computational Linguistics, 2011. 45, 149, 154

[Ouyang *et al.* 2011] You Ouyang, Wenjie Li, Sujian Li and Qin Lu. *Applying regression models to query-focused multi-document summarization*. Information Processing & Management, vol. 47, no. 2, pages 227–237, 2011. 29, 31

[Ouyang *et al.* 2013] You Ouyang, Wenjie Li, Renxian Zhang, Sujian Li and Qin Lu. *A progressive sentence selection strategy for document summarization*. Information Processing & Management, vol. 49, no. 1, pages 213–221, 2013. 31

[Owoputi *et al.* 2013] Olutobi Owoputi, Brendan O'Connor, Chris Dyer, Kevin Gimpel, Nathan Schneider and Noah A Smith. *Improved part-of-speech tagging for online conversational text with word clusters*. pages 380–391. Association for Computational Linguistics, 2013. 52, 248

[Pak & Paroubek 2010] Alexander Pak and Patrick Paroubek. *Twitter as a Corpus for Sentiment Analysis and Opinion Mining.* In LREc, volume 10, 2010. 12

[Pallavi *et al.* 2015] KP Pallavi, K Srividhya, Rexiline Ragini John Victor and MM Ramya. *HITS@ FIRE task 2015: Twitter based Named Entity Recognizer for Indian Languages.* In FIRE Workshops, pages 81–84, 2015. 56

[Pang *et al.* 2002] Bo Pang, Lillian Lee and Shivakumar Vaithyanathan. *Thumbs up?: sentiment classification using machine learning techniques*. In Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10, pages 79–86. Association for Computational Linguistics, 2002. 12

[Pang *et al.* 2008] Bo Pang, Lillian Lee*et al.* *Opinion mining and sentiment analysis*. Foundations and Trends® in Information Retrieval, vol. 2, no. 1–2, pages 1–135, 2008. 11

[Park *et al.* 2006] Sun Park, Ju-Hong Lee, Chan-Min Ahn, Jun Hong and Seok-Ju Chun. *Query based summarization using non-negative matrix factorization*. In Knowledge-Based Intelligent Information and Engineering Systems, pages 84–89. Springer, 2006. 29

[Patel *et al.* 2007] Alkesh Patel, Tanveer Siddiqui and Uma Shanker Tiwary. *A language independent approach to multilingual text summarization*. In Large scale semantic access to content (text, image, video, and sound), pages 123–132. LE CENTRE DE HAUTES ETUDES INTERNATIONALES D'INFORMATIQUE DOCUMENTAIRE, 2007. 35, 38

[Patel *et al.* 2016] Raj Nath Patel, Prakash B Pimpale and M Sasikumat. *Recurrent Neural Network based Part-of-Speech Tagger for Code-Mixed Social Media Text*. arXiv preprint arXiv:1611.04989, 2016. 54

[Patra *et al.* 2015] Braja Gopal Patra, Dipankar Das, Amitava Das and Rajendra Prasath. *Shared task on sentiment analysis in indian languages (sail) tweets-an overview*. In International Conference on Mining Intelligence and Knowledge Exploration, pages 650–655. Springer, 2015. 50, 68, 82

[Pedersen *et al.* 2004] Ted Pedersen, Siddharth Patwardhan and Jason Michelizzi. *WordNet:: Similarity: measuring the relatedness of concepts*. In Demonstration papers at HLT-NAACL 2004, pages 38–41. Association for Computational Linguistics, 2004. 12

[Pedregosa *et al.* 2011a] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay. *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, vol. 12, pages 2825–2830, 2011. 154

[Pedregosa *et al.* 2011b] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg*et al.* *Scikit-learn: Machine learning in Python*. Journal of Machine Learning Research, vol. 12, no. Oct, pages 2825–2830, 2011. 109, 152, 153, 195, 199

[Pennington *et al.* 2014] Jeffrey Pennington, Richard Socher and Christopher D Manning. *Glove: Global Vectors for Word Representation.* In EMNLP, volume 14, pages 1532–1543, 2014. 85, 86

[Phani *et al.* 2016] Shanta Phani, Shibpur IIEST, Shibamouli Lahiri and Arindam Biswas. *Sentiment Analysis of Tweets in Three Indian Languages*. WSSANLP 2016, vol. 1001, pages 93–102, 2016. 22

[Pimpale & Patel 2016] Prakash B Pimpale and Raj Nath Patel. *Experiments with POS Tagging Code-mixed Indian Social Media Text*. arXiv preprint arXiv:1610.09799, 2016. 53

[Plank *et al.* 2016] Barbara Plank, Anders Søgaard and Yoav Goldberg. *Multilingual Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Models and Auxiliary Loss*. arXiv preprint arXiv:1604.05529, 2016. 53

[Pontiki *et al.* 2016] Maria Pontiki, Dimitris Galanis, John Pavlopoulos, Harris Papageorgiou, Ion Androutsopoulos and Suresh Manandhar. *SemEval-2016 task 5: Aspect based sentiment analysis*. Proceedings of SemEval,Association for Computational Linguistics, pages 19–30, 2016. 85

[Poorgholami *et al.* 2013] Maryam Poorgholami, Mehrdad Jalali, Saeed Rahati and Taha Asgari. *Spam detection in social bookmarking websites*. In Software Engineering and

Service Science (ICSESS), 2013 4th IEEE International Conference on, pages 56–59. IEEE, 2013. 44

[Popov 2016] Alexander Popov. *Deep Learning Architecture for Part-of-Speech Tagging with Word and Suffix Embeddings*. In International Conference on Artificial Intelligence: Methodology, Systems, and Applications, pages 68–77. Springer, 2016. 149

[Poria *et al.* 2016a] Soujanya Poria, Erik Cambria, Devamanyu Hazarika and Prateek Vij. *A deeper look into sarcastic Tweets using deep convolutional neural networks*. arXiv preprint arXiv:1610.08815, 2016. 175

[Poria *et al.* 2016b] Soujanya Poria, Erik Cambria, Newton Howard, Guang-Bin Huang and Amir Hussain. *Fusing audio, visual and textual clues for sentiment analysis from multimodal content*. Neurocomputing, vol. 174, pages 50–59, 2016. 13

[Prabhu *et al.* 2016] Ameya Prabhu, Aditya Joshi, Manish Shrivastava and Vasudeva Varma. *Towards Sub-Word Level Compositions for Sentiment Analysis of Hindi-English Code Mixed Text*. arXiv preprint arXiv:1611.00472, 2016. 61, 171

[Pundlik *et al.* 2016] Sumitra Pundlik, Prasad Dasare, Prachi Kasbekar, Akshay Gawade, Gajanan Gaikwad and Purushottam Pundlik. *Multiclass classification and class based sentiment analysis for Hindi language*. In Advances in Computing, Communications and Informatics (ICACCI), 2016 International Conference on, pages 512–518. IEEE, 2016. 39

[Radev *et al.* 2002] Dragomir R Radev, Eduard Hovy and Kathleen McKeown. *Introduction to the special issue on summarization*. Computational linguistics, vol. 28, no. 4, pages 399–408, 2002. 29, 30

[Radev *et al.* 2004] Dragomir R Radev, Timothy Allison, Sasha Blair-Goldensohn, John Blitzer, Arda Celebi, Stanko Dimitrov, Elliott Drabek, Ali Hakim, Wai Lam, Danyu Liu*et al.* *MEAD-A Platform for Multidocument Multilingual Text Summarization*. In LREC, 2004. 29, 102

[Rădulescu *et al.* 2014] Cristina Rădulescu, Mihaela Dinsoreanu and Rodica Potolea. *Identification of spam comments using natural language processing techniques*. In Intelligent Computer Communication and Processing (ICCP), 2014 IEEE International Conference on, pages 29–35. IEEE, 2014. 44

[Ramanathan & Rao 2003] Ananthakrishnan Ramanathan and Durgesh D Rao. *A lightweight stemmer for Hindi*. In the Proceedings of EACL, 2003. 105

[RamaSree & Kusuma Kumari 2007] RJ RamaSree and P Kusuma Kumari. *Combining pos taggers for improved accuracy to create telugu annotated texts for information retrieval*. Dept. of Telugu Studies, Tirupathi, India, pages 1–12, 2007. 51

[Rao & Devi 2016] Pattabhi RK Rao and Sobha Lalitha Devi. *CMEE-IL: Code Mix Entity Extraction in Indian Languages from Social Media Text@ FIRE 2016–An Overview.* pages 289–295, 2016. 56

[Rao *et al.* 2015] Pattabhi RK Rao, CS Malarkodi, R Vijay Sundar Ram and Sobha Lalitha Devi. *ESM-IL: Entity Extraction from Social Media Text for Indian Languages@ FIRE 2015-An Overview.* In FIRE Workshops, pages 74–80, 2015. 55

[Read 2005] Jonathon Read. *Using emoticons to reduce dependency in machine learning techniques for sentiment classification.* In Proceedings of the ACL student research workshop, pages 43–48. Association for Computational Linguistics, 2005. 12

[Reed *et al.* 2016] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele and Honglak Lee. *Generative adversarial text to image synthesis.* In Proceedings of The 33rd International Conference on Machine Learning, volume 3, pages 1060–1069, 2016. 141

[Řehůřek & Kolkus 2009] Radim Řehůřek and Milan Kolkus. Language identification on the web: Extending the dictionary method, pages 357–368. Berlin, Heidelberg, 2009. 59

[Reyes & Rosso 2012] Antonio Reyes and Paolo Rosso. *Making objective decisions from subjective data: Detecting irony in customer reviews.* Decision Support Systems, vol. 53, no. 4, pages 754–760, 2012. 47

[Reyes & Rosso 2014] Antonio Reyes and Paolo Rosso. *On the difficulty of automatically detecting irony: beyond a simple case of negation.* Knowledge and Information Systems, vol. 40, no. 3, pages 595–614, 2014. 47

[Riedhammer *et al.* 2010] Korbinian Riedhammer, Benoit Favre and Dilek Hakkani-Tür. *Long story short–global unsupervised models for keyphrase based meeting summarization.* Speech Communication, vol. 52, no. 10, pages 801–815, 2010. 29

[Riloff *et al.* 2013] Ellen Riloff, Ashequl Qadir, Prafulla Surve, Lalindra De Silva, Nathan Gilbert and Ruihong Huang. *Sarcasm as Contrast between a Positive Sentiment and Negative Situation.* In EMNLP, volume 13, pages 704–714, 2013. 5, 46, 48

[Ruscoe 2009] Tony Ruscoe. *Google Translator Toolkit*, 2009. 250

[Rush *et al.* 2015] Alexander M Rush, Sumit Chopra and Jason Weston. *A neural attention model for abstractive sentence summarization.* arXiv preprint arXiv:1509.00685, 2015. 28, 34

[Saha *et al.* 2008] Sujan Kumar Saha, Sanjay Chatterji, Sandipan Dandapat, Sudeshna Sarkar and Pabitra Mitra. *A hybrid approach for named entity recognition in indian languages.* In Proceedings of the IJCNLP-08 Workshop on NER for South and South East Asian languages, pages 17–24, 2008. 55

[Saini 2016a] Anuj Saini. *Anuj@ DPIL-FIRE2016: A Novel Paraphrase Detection Method in Hindi Language using Machine Learning.* In Working notes of FIRE 2016 - Forum for Information Retrieval Evaluation, Kolkata, India, December 7-10, 2016, CEUR Workshop Proceedings, pages 270–274. CEUR-WS.org, 2016. 28

[Saini 2016b] Anuj Saini. *Code Mixed Cross Script Question Classification.* In Working notes of FIRE 2016 - Forum for Information Retrieval Evaluation, Kolkata, India, December 7-10, 2016, CEUR Workshop Proceedings, pages 115–118. CEUR-WS.org, 2016. 57

[San 2009] Hong Ka San. *Chinese-English code-switching in blogs by Macao young people.* 2009. 6, 49

[Sang 2002] Erik F Tjong Kim Sang. *Introduction to the CoNLL-2002 shared task: language-independent named entity recognition, proceedings of the 6th conference on Natural language learning.* August, vol. 31, pages 1–4, 2002. 55

[Sankar *et al.* 2011] Sankar, Sundar Vijay and Sobha Lalitha Devi. *Text Extraction for an Agglutinative Language.* Language in India, vol. 11, no. 5, pages 56–59, 2011. 37, 38, 41

[Sankarasubramaniam *et al.* 2014] Yogesh Sankarasubramaniam, Krishnan Ramanathan and Subhankar Ghosh. *Text summarization using Wikipedia.* Information Processing & Management, vol. 50, no. 3, pages 443–461, 2014. 32

[Sapkal & Shrawankar 2016] Krutika Sapkal and Urmila Shrawankar. *Transliteration of Secured SMS to Indian Regional Language.* Procedia Computer Science, vol. 78, pages 748–755, 2016. 58, 60

[Sarkar 2012] Kamal Sarkar. *An approach to summarizing Bengali news documents.* In proceedings of the International Conference on Advances in Computing, Communications and Informatics, pages 857–862. ACM, 2012. 37, 42

[Sarkar 2015] Kamal Sarkar. *A hidden markov model based system for entity extraction from social media english text at fire 2015.* arXiv preprint arXiv:1512.03950, 2015. 56

[Sarkar 2016a] Kamal Sarkar. *A CRF Based POS Tagger for Code-mixed Indian Social Media Text.* arXiv preprint arXiv:1612.07956, 2016. 54

[Sarkar 2016b] Kamal Sarkar. *Part-of-speech tagging for code-mixed indian social media text at ICON 2015.* arXiv preprint arXiv:1601.01195, 2016. 53

[Schuster & Paliwal 1997] Mike Schuster and Kuldip K Paliwal. *Bidirectional recurrent neural networks.* IEEE Transactions on Signal Processing, vol. 45, no. 11, pages 2673–2681, 1997. 72, 80

[Sculley & Wachman 2007] David Sculley and Gabriel M Wachman. *Relaxed online SVMs for spam filtering*. In Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, pages 415–422. ACM, 2007. 44

[Se *et al.* 2015] Shriya Se, R Vinayakumar, M Anand Kumar and KP Soman. *AMRITA-CEN@ SAIL2015: Sentiment Analysis in Indian Languages*. In International Conference on Mining Intelligence and Knowledge Exploration, pages 703–710. Springer, 2015. 56

[Se *et al.* 2016] Shriya Se, R Vinayakumar, M Anand Kumar and KP Soman. *Predicting the Sentimental Reviews in Tamil Movie using Machine Learning Algorithms*. Indian Journal of Science and Technology, vol. 9, no. 45, 2016. 21

[Sequiera *et al.* 2015a] Royal Sequiera, Monojit Choudhury and Kalika Bali. *Pos tagging of hindi-english code mixed text from social media: Some machine learning experiments*. In Proceedings of International Conference on NLP. NLPAI, December, pages 233–242, 2015. 53

[Sequiera *et al.* 2015b] Royal Sequiera, Monojit Choudhury, Parth Gupta, Paolo Rosso, Shubham Kumar, Somnath Banerjee, Sudip Kumar Naskar, Sivaji Bandyopadhyay, Gokul Chittaranjan, Amitava Das and Kunal Chakma. *Overview of FIRE-2015 Shared Task on Mixed Script Information Retrieval*. In Working notes of FIRE 2015 - Forum for Information Retrieval Evaluation, Gandhinagar, India, December, 2015, volume 1587 of *CEUR Workshop Proceedings*, pages 19–25. CEUR-WS.org, 2015. 50, 59, 227, 260, 263

[Shafie & Nayan 2013] Latisha Asmaak Shafie and Surina Nayan. *Languages, code-switching practice and primary functions of Facebook among university students*. Studies in English Language Teaching, vol. 1, no. 1, pages 187–199, 2013. 49

[Shao *et al.* 2017] Yeqin Shao, Marcello Trovati, Quan Shi, Olga Angelopoulou, Eleana Asimakopoulou and Nik Bessis. *A hybrid spam detection method based on unstructured datasets*. Soft Computing, vol. 21, no. 1, pages 233–243, 2017. 46

[Sharma & Lin 2013] Kuldeep Sharma and King-Ip Lin. *Review spam detector with rating consistency check*. In Proceedings of the 51st ACM Southeast Conference, pages 34:1–34:6. ACM, 2013. 44

[Sharma *et al.* 2014] Richa Sharma, Shweta Nigam and Rekha Jain. *Polarity detection movie reviews in hindi language*. arXiv preprint arXiv:1409.3942, 2014. 17

[Sharma *et al.* 2015a] Shashank Sharma, PYKL Srinivas and Rakesh Chandra Balabantaray. *Sentiment analysis of code-mix script*. In Computing and Network Communications (CoCoNet), 2015 International Conference on, pages 530–534. IEEE, 2015. 61

[Sharma *et al.* 2015b] Shashank Sharma, PYKL Srinivas and Rakesh Chandra Balabantaray. *Text normalization of code mix and sentiment analysis*. In Advances in Computing, Communications and Informatics (ICACCI), 2015 International Conference on, pages 1468–1473. IEEE, 2015. 58, 60

[Sharma *et al.* 2015c] Yakshi Sharma, Veenu Mangat and Mandeep Kaur. *A practical approach to Sentiment Analysis of hindi tweets*. In Next Generation Computing Technologies (NGCT), 2015 1st International Conference on, pages 677–680. IEEE, 2015. 18

[Sharma *et al.* 2016] Arnav Sharma, Sakshi Gupta, Raveesh Motlani, Piyush Bansal, Manish Srivastava, Radhika Mamidi and Dipti M Sharma. *Shallow Parsing Pipeline for Hindi-English Code-Mixed Social Media Text*. arXiv preprint arXiv:1604.03136, 2016. 54

[Sharma 2014] Anu Sharma. *Sentiment Analyzer using Punjabi Language*. International Journal of Innovative Research in Computer and Communication Engineering, vol. 2, pages 5904–5909, 2014. 20

[Shen *et al.* 2014] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng and Grégoire Mesnil. *A latent semantic model with convolutional-pooling structure for information retrieval*. In Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, pages 101–110. ACM, 2014. 57

[Shi *et al.* 2013] Lu Shi, Shucheng Yu, Wenjing Lou and Y Thomas Hou. *Sybilshield: An agent-aided social network-based sybil defense among multiple communities*. In INFOCOM, 2013 Proceedings IEEE, pages 1034–1042. IEEE, 2013. 44

[Shirani-Mehr 2014] Houshmand Shirani-Mehr. *Applications of Deep Learning to Sentiment Analysis of Movie Reviews*, 2014. 13

[Singh *et al.* 2006] Smriti Singh, Kuhoo Gupta, Manish Shrivastava and Pushpak Bhattacharyya. *Morphological richness offsets resource demand-experiences in constructing a POS tagger for Hindi*. In Proceedings of the COLING/ACL on Main conference poster sessions, pages 779–786. Association for Computational Linguistics, 2006. 51

[Singh *et al.* 2016] Meghna Singh, Rajita Shukla, Jaya Saraswati, Laxmi Kashyap, Diptesh Kanojia and Pushpak Bhattacharyya. *Mapping it differently: A solution to the linking challenges*. pages 406–413, 2016. 244

[Sirivianos *et al.* 2011] Michael Sirivianos, Kyungbaek Kim and Xiaowei Yang. *Socialfilter: Introducing social trust to collaborative spam mitigation*. In INFOCOM, 2011 Proceedings IEEE, pages 2300–2308. IEEE, 2011. 44

[Socher *et al.* 2011] Richard Socher, Eric H Huang, Jeffrey Pennington, Andrew Y Ng and Christopher D Manning. *Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection.* In NIPS, volume 24, pages 801–809, 2011. 27, 104

[Socher *et al.* 2013] Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, Christopher Potts*et al. Recursive deep models for semantic compositionality over a sentiment treebank*. In Proceedings of the conference on empirical methods in natural language processing (EMNLP), pages 1631–1642, 2013. 47, 149

[Sokolova & Lapalme 2009] Marina Sokolova and Guy Lapalme. *A systematic analysis of performance measures for classification tasks*. Information Processing & Management, vol. 45, no. 4, pages 427–437, 2009. 71

[Solorio & Liu 2008] Thamar Solorio and Yang Liu. *Learning to predict code-switching points*. In Proceedings of the Conference on Empirical Methods in Natural Language Processing, pages 973–981. Association for Computational Linguistics, 2008. 51, 55

[Song *et al.* 2017] Long Song, Raymond Yiu Keung Lau, Ron Chi-Wai Kwok, Kristijan Mirkovski and Wenyu Dou. *Who are the spoilers in social media marketing? Incremental learning of latent semantics for social spam detection*. Electronic Commerce Research, vol. 17, no. 1, pages 51–81, 2017. 46

[Sotillo 2012] Susanna Sotillo. *Ehhhh utede hacen plane sin mi???:@ im feeling left out:(Form, Function and Type of Code Switching in SMS Texting*. In ICAME, volume 33, pages 309–310, 2012. 49

[Steinberger *et al.* 2007] Josef Steinberger, Massimo Poesio, Mijail A Kabadjov and Karel Ježek. *Two uses of anaphora resolution in summarization*. Information Processing & Management, vol. 43, no. 6, pages 1663–1680, 2007. 131

[Stringhini *et al.* 2013] Gianluca Stringhini, Gang Wang, Manuel Egele, Christopher Kruegel, Giovanni Vigna, Haitao Zheng and Ben Y Zhao. *Follow the green: growth and dynamics in twitter follower markets*. In Proceedings of the 2013 conference on Internet measurement conference, pages 163–176. ACM, 2013. 44, 45

[Subramaniam & Dalal 2015] Manjula Subramaniam and Vipul Dalal. *Test Model for Rich Semantic Graph Representation for Hindi Text using Abstractive Method.* International Research Journal of Engineering and Technology (IRJET), vol. 2, no. 2, pages 113–116, 2015. 38, 39

[Sun *et al.* 2013] Huan Sun, Alex Morales and Xifeng Yan. *Synthetic review spamming and defense*. In Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 1088–1096. ACM, 2013. 44

[Sun *et al.* 2016] Xiao Sun, Chengcheng Li and Fuji Ren. *Sentiment analysis for Chinese microblog based on deep neural networks with convolutional extension features*. Neurocomputing, vol. 210, pages 227–236, 2016. 63

[Sundaram *et al.* 2009] Mahalakshmi Shanmuga Sundaram, Kumar Madasamy and Soman Kotti Padannayil. *: Paraphrase Detection for Twitter using Unsupervised Feature Learning with Recursive Autoencoders*. In Workshop Proceedings of the International Workshop on Semantic Evaluation 2015 (Sem Eval-2015), Denver, Colorado, US, pages 45–50. Citeseer, 2009. 26, 103

[Sureka 2011] Ashish Sureka. *Mining user comment activity for detecting forum spammers in youtube*. arXiv preprint arXiv:1103.5044, 2011. 44

[Sutskever *et al.* 2014] Ilya Sutskever, Oriol Vinyals and Quoc V Le. *Sequence to sequence learning with neural networks*. In Advances in neural information processing systems, pages 3104–3112, 2014. 149

[Tan *et al.* 2012] Enhua Tan, Lei Guo, Songqing Chen, Xiaodong Zhang and Yihong Zhao. *Spammer behavior analysis and detection in user generated content on social networks*. In Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on, pages 305–314. IEEE, 2012. 44

[Tan *et al.* 2013] Enhua Tan, Lei Guo, Songqing Chen, Xiaodong Zhang and Yihong Zhao. *Unik: Unsupervised social network spam detection*. In Proceedings of the 22nd ACM international conference on Information & Knowledge Management, pages 479–488. ACM, 2013. 44

[Tang *et al.* 2009] Huifeng Tang, Songbo Tan and Xueqi Cheng. *A survey on sentiment detection of reviews*. Expert Systems with Applications, vol. 36, no. 7, pages 10760–10773, 2009. 11

[Tang *et al.* 2014] Duyu Tang, Furu Wei, Bing Qin, Ting Liu and Ming Zhou. *Coooolll: A deep learning system for twitter sentiment classification*. In Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014), pages 208–212, 2014. 13

[Tepperman *et al.* 2006] Joseph Tepperman, David R Traum and Shrikanth Narayanan. *" yeah right": sarcasm recognition for spoken dialogue systems*. In Ninth International Conference on Spoken Language Processing, pages 1838–1841, 2006. 46, 162

[Thaokar & Malik 2013] Chetana Thaokar and Latesh Malik. *Test model for summarizing hindi text using extraction method*. In Information & Communication Technologies (ICT), 2013 IEEE Conference on, pages 1138–1143. IEEE, 2013. 37, 39

[Tjong Kim Sang & De Meulder 2003] Erik F Tjong Kim Sang and Fien De Meulder. *Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition*. In Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4, pages 142–147. Association for Computational Linguistics, 2003. 55

[Toutanova *et al.* 2003] Kristina Toutanova, Dan Klein, Christopher D Manning and Yoram Singer. *Feature-rich part-of-speech tagging with a cyclic dependency network*. In Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1, pages 173–180. Association for Computational Linguistics, 2003. 52, 54

[Tsur *et al.* 2010] Oren Tsur, Dmitry Davidov and Ari Rappoport. *ICWSM-A Great Catchy Name: Semi-Supervised Recognition of Sarcastic Sentences in Online Product Reviews.* In ICWSM, pages 162–169, 2010. 46, 47, 162, 171

[Tsytsarau & Palpanas 2012] Mikalai Tsytsarau and Themis Palpanas. *Survey on mining subjective data on the web*. Data Mining and Knowledge Discovery, vol. 24, no. 3, pages 478–514, 2012. 5, 11

[Tumasjan *et al.* 2010] Andranik Tumasjan, Timm Oliver Sprenger, Philipp G Sandner and Isabell M Welpe. *Predicting elections with twitter: What 140 characters reveal about political sentiment.* ICWSM, vol. 10, no. 1, pages 178–185, 2010. 2

[Turney 2002] Peter D Turney. *Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews*. In Proceedings of the 40th annual meeting on association for computational linguistics, pages 417–424. Association for Computational Linguistics, 2002. 18, 19

[Vaswani *et al.* 2016] Ashish Vaswani, Yonatan Bisk, Kenji Sagae and Ryan Musa. *Supertagging with LSTMs*. In Proceedings of the Human Language Technology Conference of the NAACL, pages 232–237, 2016. 53

[Velásquez & Gonzalez 2010] Juan D Velásquez and Pablo Gonzalez. *Expanding the possibilities of deliberation: The use of data mining for strengthening democracy with an application to education reform*. The Information Society, vol. 26, no. 1, pages 1–16, 2010. 2

[Venugopalan *et al.* 2015] Subhashini Venugopalan, Marcus Rohrbach, Jeffrey Donahue, Raymond Mooney, Trevor Darrell and Kate Saenko. *Sequence to sequence-video to text*. In Proceedings of the IEEE International Conference on Computer Vision, pages 4534–4542, 2015. 34

[Vilares *et al.* 2015] David Vilares, Miguel A Alonso and Carlos Gómez-Rodrıguez. *Sentiment analysis on monolingual, multilingual and code-switching twitter corpora*. In 6TH

WORKSHOP ON COMPUTATIONAL APPROACHES TO SUBJECTIVITY, SENTIMENT AND SOCIAL MEDIA ANALYSIS WASSA 2015, pages 2–8, 2015. 60

[Vilares *et al.* 2017] David Vilares, Miguel A Alonso and Carlos Gómez-Rodríguez. *Supervised sentiment analysis in multilingual environments*. Information Processing & Management, vol. 53, no. 3, pages 595–607, 2017. 61

[Vishal & Gurpreet 2012] Gupta Vishal and Singh Lehal Gurpreet. *Automatic Punjabi text extractive summarization system*. In 24th International Conference on Computational Linguistics, pages 191–198. Citeseer, 2012. 41

[Viswanath *et al.* 2014] Bimal Viswanath, Muhammad Ahmad Bashir, Mark Crovella, Saikat Guha, Krishna P Gummadi, Balachander Krishnamurthy and Alan Mislove. *Towards Detecting Anomalous User Behavior in Online Social Networks.* In USENIX Security Symposium, pages 223–238, 2014. 45

[Vo & Zhang 2015] Duy-Tin Vo and Yue Zhang. *Target-Dependent Twitter Sentiment Classification with Rich Automatic Features.* In IJCAI, pages 1347–1353, 2015. 13

[Vo *et al.* 2015] Ngoc Phuoc An Vo, Simone Magnolini and Octavian Popescu. *Paraphrase identification and semantic similarity in twitter with simple features*. In The 3rd International Workshop on Natural Language Processing for Social Media, pages 10–19, 2015. 26, 103

[Vyas *et al.* 2014] Yogarshi Vyas, Spandana Gella, Jatin Sharma, Kalika Bali and Monojit Choudhury. *POS Tagging of English-Hindi Code-Mixed Social Media Content.* In EMNLP, volume 14, pages 974–979, 2014. 51, 55

[Wallace *et al.* 2014] Byron C Wallace, Laura Kertz Do Kook Choe, Laura Kertz and Eugene Charniak. *Humans Require Context to Infer Ironic Intent (so Computers Probably do, too).* In ACL (2), pages 512–516, 2014. 46

[Wallace *et al.* 2015] Byron C Wallace, Do Kook Choe and Eugene Charniak. *Sparse, Contextually Informed Models for Irony Detection: Exploiting User Communities, Entities and Sentiment.* In ACL (1), pages 1035–1044, 2015. 46

[Wan 2008] Xiaojun Wan. *Using only cross-document relationships for both generic and topic-focused multi-document summarizations*. Information Retrieval, vol. 11, no. 1, pages 25–49, 2008. 29

[Wang & Liu 2015] Bo Wang and Min Liu. *Deep Learning for Aspect-Based Sentiment Analysis*, 2015. x, 89

[Wang *et al.* 2008] Dingding Wang, Tao Li, Shenghuo Zhu and Chris Ding. *Multi-document summarization via sentence-level semantic analysis and symmetric matrix factorization*. In Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval, pages 307–314. ACM, 2008. 32

[Wang *et al.* 2011] De Wang, Danesh Irani and Calton Pu. *A social-spam detection framework.* In Proceedings of the 8th Annual Collaboration, Electronic Messaging, Anti-Abuse and Spam Conference, pages 46–54. ACM, 2011. 44

[Wang *et al.* 2015a] Bo Wang, Arkaitz Zubiaga, Maria Liakata and Rob Procter. *Making the most of tweet-inherent features for social spam detection on Twitter.* arXiv preprint arXiv:1503.07405, 2015. 45

[Wang *et al.* 2015b] Peilu Wang, Yao Qian, Frank K Soong, Lei He and Hai Zhao. *Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Recurrent Neural Network.* arXiv preprint arXiv:1510.06168, 2015. 52

[Wang *et al.* 2015c] Zelin Wang, Zhijian Wu, Ruimin Wang and Yafeng Ren. *Twitter sarcasm detection exploiting a context-based model.* In International Conference on Web Information Systems Engineering, pages 77–91. Springer, 2015. 48

[Wang *et al.* 2016] Jin Wang, Liang-Chih Yu, K Robert Lai and Xuejie Zhang. *Dimensional sentiment analysis using a regional cnn-lstm model.* In The 54th Annual Meeting of the Association for Computational Linguistics, pages 225–230, 2016. 74, 75

[Wang 2010] Alex Hai Wang. *Don't follow me: Spam detection in twitter.* In Security and Cryptography (SECRYPT), Proceedings of the 2010 International Conference on, pages 1–10. IEEE, 2010. 44, 45

[Wiatowski & Bölcskei 2017] Thomas Wiatowski and Helmut Bölcskei. *A mathematical theory of deep convolutional neural networks for feature extraction.* IEEE Transactions on Information Theory, 2017. 65, 70

[Wiebe *et al.* 2005] Janyce Wiebe, Theresa Wilson and Claire Cardie. *Annotating expressions of opinions and emotions in language.* Language resources and evaluation, vol. 39, no. 2, pages 165–210, 2005. 25

[Wu & Davison 2005] Baoning Wu and Brian D Davison. *Identifying link farm spam pages.* In Special interest tracks and posters of the 14th international conference on World Wide Web, pages 820–829. ACM, 2005. 45

[Xia & Zhiwen 2016] Jianhong Cecilia Xia and S Zhiwen. *Spatial and Temporal Sentiment Analysis of Twitter data.* In European Handbook of Crowdsourced Geographic Information, pages 205–221. London: Ubiquity Press, 2016. 26

[Xun *et al.* 2017] Guangxu Xun, Yaliang Li, Wayne Xin Zhao, Jing Gao and Aidong Zhang. *A correlated topic model using word embeddings.* In Proceedings of the 26th International Joint Conference on Artificial Intelligence.[doi> 10.24963/ijcai. 2017/588], 2017. 63

[Yang & Lee 2011] Hsin-Chang Yang and Chung-Hong Lee. *Post-level spam detection for social bookmarking web sites*. In Advances in Social Networks Analysis and Mining (ASONAM), 2011 International Conference on, pages 180–185. IEEE, 2011. 44

[Yang & Lee 2014] Hsin-Chang Yang and Chung-Hong Lee. *Detecting tag spams for social bookmarking Websites using a text mining approach*. International Journal of Information Technology & Decision Making, vol. 13, no. 02, pages 387–406, 2014. 44

[Yang *et al.* 2011] Chao Yang, Robert Chandler Harkreader and Guofei Gu. *Die free or live hard? empirical evaluation and new design for fighting evolving twitter spammers*. In International Workshop on Recent Advances in Intrusion Detection, pages 318–337. Springer, 2011. 45

[Yang *et al.* 2012] Chao Yang, Robert Harkreader, Jialong Zhang, Seungwon Shin and Guofei Gu. *Analyzing spammers' social networks for fun and profit: a case study of cyber criminal ecosystem on twitter*. In Proceedings of the 21st international conference on World Wide Web, pages 71–80. ACM, 2012. 44, 45

[Yang *et al.* 2014] Libin Yang, Xiaoyan Cai, Yang Zhang and Peng Shi. *Enhancing sentence-level clustering with ranking-based clustering framework for theme-based summarization*. Information sciences, vol. 260, pages 37–50, 2014. 31

[Yardi *et al.* 2009] Sarita Yardi, Daniel Romero, Grant Schoenebeck*et al.* *Detecting spam in a twitter network*. First Monday, vol. 15, no. 1, pages 1–9, 2009. 45

[Yeh *et al.* 2005] Jen-Yuan Yeh, Hao-Ren Ke, Wei-Pang Yang and I-Heng Meng. *Text summarization using a trainable summarizer and latent semantic analysis*. Information processing & management, vol. 41, no. 1, pages 75–95, 2005. 30

[Yih *et al.* 2014] Wen-tau Yih, Xiaodong He and Christopher Meek. *Semantic Parsing for Single-Relation Question Answering*. In ACL (2), pages 643–648. Citeseer, 2014. 57

[Yin & Schütze 2015] Wenpeng Yin and Hinrich Schütze. *Convolutional Neural Network for Paraphrase Identification*. In HLT-NAACL, pages 901–911, 2015. 27, 103, 104

[Zajic *et al.* 2007] David Zajic, Bonnie J Dorr, Jimmy Lin and Richard Schwartz. *Multi-candidate reduction: Sentence compression as a tool for document summarization tasks*. Information Processing & Management, vol. 43, no. 6, pages 1549–1570, 2007. 32

[Zhang & Lee 2003] Dell Zhang and Wee Sun Lee. *Question classification using support vector machines*. In Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval, pages 26–32. ACM, 2003. 57

[Zhang & Skiena 2010] Wenbin Zhang and Steven Skiena. *Trading Strategies to Exploit Blog and News Sentiment*. In ICWSM, pages 375–378, 2010. 2

[Zhang *et al.* 2015] Xiang Zhang, Junbo Zhao and Yann LeCun. *Character-level convolutional networks for text classification*. In Advances in Neural Information Processing Systems, pages 649–657, 2015. 57

[Zheng *et al.* 2016] Xianghan Zheng, Xueying Zhang, Yuanlong Yu, Tahar Kechadi and Chunming Rong. *ELM-based spammer detection in social networks*. The Journal of Supercomputing, vol. 72, no. 8, pages 2991–3005, 2016. 44

[Zhou *et al.* 2014] Bing Zhou, Yiyu Yao and Jigang Luo. *Cost-sensitive three-way email spam filtering*. Journal of Intelligent Information Systems, vol. 42, no. 1, pages 19–45, 2014. 44

[Zhu *et al.* 2011] Linhong Zhu, Aixin Sun and Byron Choi. *Detecting spam blogs from blog search results*. Information Processing & Management, vol. 47, no. 2, pages 246–262, 2011. 44

[Zimbra *et al.* 2016] David Zimbra, Manoochehr Ghiassi and Sean Lee. *Brand-related Twitter sentiment analysis using feature engineering and the dynamic architecture for artificial neural networks*. In System Sciences (HICSS), 2016 49th Hawaii International Conference on, pages 1930–1938. IEEE, 2016. 13

[Zisiadis *et al.* 2011] Dimitris Zisiadis, Spyros Kopsidas, Argyris Varalis and Leandros Tassiulas. *Mailbook: A social network against spamming*. In Internet Technology and Secured Transactions (ICITST), 2011 International Conference for, pages 245–249. IEEE, 2011. 44

Publications

Conference Papers

1. Rupal Bhargava, Bapiraju Vamsi Tadikonda, Yashvardhan Sharma, "Named Entity Recognition for Code Mixed SentencesâĂİ, In proceedings of International conference on Machine learning, Image processing, Network security and Data Sciences(MIND), March 2019. (accepted)

2. Rupal Bhargava, Yashvardhan Sharma, Ayushi Agarwal, "FAID: Feature Aftermath for Irony Discernment", In proceedings of International conference on Cloud Computing, Data Science and Engineering, CONFLUENCE 2019, IEEE, January 2018.

3. Rupal Bhargava, Gargi Sharma, Yashvardhan Sharma, "Deep Paraphrase Detection in Indian Languages", In proceedings of International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2017,IEEE/ACM, August 2017, pp. 1152-1159 .

4. Rupal Bhargava, Yashvardhan Sharma, "MSATS: Multilingual Sentiment Analysis via Text Summarization", In proceedings of Seventh International Conference on Cloud Computing, Data Science & Engineering (CONFLUENCE-2017), IEEE, January 2017, pp.71-76.

5. Rupal Bhargava, Yashvardhan Sharma, Shubham Sharma, "Sentiment Analysis for Mixed Script Indic Sentences", In proceedings of Fifth International Conference on Advances in Computing, Communications & Informatics, IEEE,September 2016, pp.524-529.

6. Rupal Bhargava, Yashvardhan Sharma, Gargi Sharma, "ATSSI: Abstractive Text Summarization using Sentiment Infusion", In proceedings of Twelfth International Multi Conference on Information Processing, Procedia Computer Science, Vol 89C, August 2016, pp. 404-411.

7. Rupal Bhargava, Yashvardhan Sharma, "Analyzing Sentiments through Text Summarization", In Proceedings of International Conference on Evidence Based Management, ISBN: 978-93-84935-18-4, Vol II, March 2015, pp 769 -772.

8. Rupal Bhargava, Gargi Sharma, Yashvardhan Sharma, "BITS _PILANI _TEAM3@POS Tagging for Code Mix Indian Social Media", In NLP Tool Contest on POS Tagging for Code Mixed Indian Social Media (Facebook, Twitter, and WhatsApp) TEXT @ ICON 2016, International Conference on Natural Language Processing, December 2016.

9. Rupal Bhargava, Bapiraju Vamsi Tadikonda, Yashvardhan Sharma, "BITS_PILANI_TEAM2@POS Tagging for Code Mix Indian Social Media", In NLP Tool Contest on POS Tagging for Code Mixed Indian Social Media (Facebook, Twitter, and WhatsApp) TEXT @ ICON 2016, International Conference on Natural Language Processing, December 2016.

10. Rupal Bhargava, Raghav Bhartia, Indrajeet Mishra, Yashvardhan Sharma, "BITS _PILANI _ TEAM1@POS Tagging for Code Mix Indian Social Media", In NLP Tool Contest on POS Tagging for Code Mixed Indian Social Media (Facebook, Twitter, and WhatsApp) TEXT @ ICON 2016, International Conference on Natural Language Processing, December 2016.

11. Rupal Bhargava, Anushka Baoni, Harshit Jain, Yashvardhan Sharma, "BITS _PILANI@DPIL -FIRE2016: Paraphrase Detection in Hindi Language Using Syntactic Features of Phrase". In workshop proceedings of Forum of Information Retrieval and Evaluation 2016, CEUR Workshop Proceedings, Vol-1737, pp. 239-243, December 2016.

12. Rupal Bhargava, Shubham Khandelwal, Akshit Bhatia, Yashvardhan Sharma, "Modeling Classifier for Code Mixed Cross Script Questions". In workshop proceedings of Forum of Information Retrieval and Evaluation 2016, CEUR Workshop Proceedings, Vol-1737, pp. 109-114, December 2016.

13. Rupal Bhargava, Bapiraju Vamsi Tadikona, Yashvardhan Sharma, "Named Entity Recognition for Code Mixing in Indian Languages using Hybrid Approach". In workshop proceedings of Forum of Information Retrieval and Evaluation 2016, CEUR Workshop Proceedings, Vol-1737, pp. 313-317, December 2016.

14. Rupal Bhargava, Yashvardhan Sharma, Shubham Sharma, Abhinav Baid, "Query Labelling for Indic Languages using a hybrid approach". In workshop proceedings of Forum of Information Retrieval and Evaluation 2015, CEUR Workshop Proceedings, Vol 1587, pp. 40-42, December 2015.

15. Rupal Bhargava, Yashvardhan Sharma, "User Modeled Aspect Based Sentiment Analysis", In proceedings of National Conference on Emerging Trends of Research in Applied Sciences, Experimental and Computational Techniques, IJERT, February 2015, pp 21-24.

16. Rupal Bhargava, Gargi Sharma, Yashvardhan Sharma, "Deep Text Summarization using Generative Adversarial Networks in Indian Languages", International Conference on Computational Science, 2019, Portugal, (Communicated)

Journal Papers

1. Rupal Bhargava, Shivangi Arora, Yashvardhan Sharma, "NASAIL: Neural networks based architecture for Sentiment Analysis in Indian Languages", Journal of Intelligent Systems (In press).

2. Rupal Bhargava, Anushka Baoni, Yashvardhan Sharma, "Composite Sequential Modelling for Identifying Fake Reviews", In International Journal of Intelligent Systems (IJISys) (In press).

3. Rupal Bhargava, Yashvardhan Sharma, Bapiraju Vamsi Tadikonda, Shubham Khandelwal, " Identification of Indian Languages and Sentiments in Code Mixed Text", IJCSE, Inderscience (Communicated).

4. Rupal Bhargava, Gargi Sharma, Yashvardhan Sharma, "Strategies for Modelling Code Mixed Cross Script Question Classification", IJAISC (Communicated).

Biographies

Brief Biography of the Supervisor

Dr. Yashvardhan Sharma is currently Associate Professor in the Department of Computer Science and Information Systems at BITS, Pilani. He is also Faculty-Incharge at Information Processing Center, Software Development and Educational Technology, BITS, Pilani. Dr. Sharma obtained his doctorate in Computer Science from BITS, Pilani in 2008. Previously he completed his M.E. in Software Systems in 2001. At BITS, Pilani he has been involved in teaching, research and administration. He has published more than 40 research papers in national and international conferences/ journals in Software Engineering, databases, data warehousing and data mining. He has teaching experience of more than 19 years at BITS, Pilani. He presently teaches courses on Object Oriented Analysis and Design, Database systems, Data warehousing, and Data mining. He has successfully completed three major research projects sponsored by UGC, DST and BITS, Pilani over a worth of one crore. His current research interests include Natural Language Processing (NLP), Data Mining, Machine Learning, Sentiment Analysis, Social Media Analytics, NLP for Indian Languages.

Brief Biography of the Candidate

Rupal Bhargava is currently a full time research scholar in the Department of Computer Science at Birla Institute of Technology & Science, Pilani, Rajasthan, India. She is pursuing her Ph.D under supervision of Dr. Yashvardhan Sharma. She has completed her M.Tech in Computer Science from Bansathali Vidyapeeth, Rajasthan in 2013. She had been associated as a Junior Research Fellow with DST funded project -"Design and Development of Opinion Mining Framework" (PI: Dr. Yashvardhan Sharma) during tenure of August 2014-January 2017. Her current research interests are in areas of Natural Language Processing, Data Mining, Information Retrieval, Machine Learning and Deep Learning. She has various publications in International Conferences and Journals.