WORD LEVEL LANGUAGE IDENTIFICATION OF CODE-MIXING TEXT IN SOCIAL MEDIA USING NLP

Kasthuri Shanmugalingam

168287D

Degree of Master of Science in Artificial Intelligence

Department of Computational Mathematics

University of Moratuwa

Sri Lanka

April 2019

WORD LEVEL LANGUAGE IDENTIFICATION OF CODE-MIXING TEXT IN SOCIAL MEDIA USING NLP

Kasthuri Shanmugalingam

168287D

Thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Artificial Intelligence

Department of Computational Mathematics

University of Moratuwa

Sri Lanka

April 2019

Declaration

I declare that this is my own work and this thesis does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my thesis/dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Name of Student

Kasthuri Shanmugalingam

Signature of Student: Date:

The above candidate has carried out research for the Master's Dissertation under my supervision.

Name of Supervisor

Dr. Sagara Sumathipala

Signature of Supervisor: Date:

Acknowledgements

I would like to make this opportunity to express my sincere gratitude to my supervisor Dr. Sagara Sumathipala for his valuable guidance extended throughout the research. This research would not have been completed to success without his immense support and guidance. Further, I would like to thank prof. Asoka Karunananda to gave valuable guidance to the documentation of the work during the lectures. And I would like to thank all academic staff of the Department of Computational Mathematics to gave sufficient knowledge to complete this research.

I wish to express my sincere thanks to my family members and colleagues who stood beside me whenever I need and helped me always with support, advice, and encouragement to complete my research work successfully.

Abstract

Automatic analyzing and extracting useful information from the noisy social media content are currently getting more attention from the research community. Recent days people easily mixing their native language along with the English language together to express their thoughts in social media, using the Unicode characters written in Roman Scripts. Thus these types of noisy code-mixed text are characterized by a high percentage of spelling mistakes with phonetic typing, wordplay, creative spelling, abbreviations, Meta tags, and so on. Identification of languages at word level become as necessary part for analyzing the noisy content in social media. It would be used as an intimidate language identifier for chatbot application by using the native languages.

For this study used Tamil-English and Sinhala-English code-mixed text from social media. Natural Language Processing (NLP) and Machine Learning (ML) technologies used to identify the language tags at the word level. A novel approach proposed for this system implemented as machine learning classifier based on features such as Tamil Unicode characters in Roman scripts, dictionaries, double consonant, and term frequency used for Tamil-English code-mixed text and features such as Sinhala Unicode characters written in Roman scripts, dictionaries, and term frequency used for Sinhala-English code-mixed text.

Different machine learning classifiers such as Support Vector Machines (SVM), Naive Bayes, Logistic Regression, Random Forest and Decision Trees used in the model evaluation process. Ten-fold cross-validation used to evaluate the performance based on language tags at the word level. Among that the highest accuracy of 89.46% was obtained in SVM classifier and 90.5% was obtained in Random Forest classifier for Tamil-English (Tanglish) and Sinhala-English (Singlish) code-mixed text respectively.

In the testing process of Tanglish model with SVM and Singlish model with Random Forest gave accuracy as 93.87% and 95.83% respectively for the testing unseen data. Tanglish model with SVM gave F-Measure for ‘tam’ and ‘eng’ tags were 0.965 and 0.894 respectively. Singlish model with Random Forest gave F-Measure for ‘sin’ and ‘eng’ tags were 0.975 and 0.929 respectively. So this the evidence that most of the times the Tanglish model with SVM and Singlish model with Random Forest predict the language labels correctly at word level.

Table of Contents

Declaration i

Acknowledgements ii

Abstract iii

List of Figures vii

List of Tables viii

Chapter 1 Introduction 1

1.1 Prolegomena 1

1.2 Objectives 2

1.3 Background and Motivation 2

1.4 Code Mixing Problem in Brief 3

1.5 Proposed Solution 4

1.6 Resource Requirements 4

1.7 Structure of the Thesis 4

1.8 Summary 5

Chapter 2 Code Mixing in Social Media – Practices and Issues 6

2.1 Introduction 6

2.2 State of the art of language identification of code-mixed text in social media 6

2.2.1 Code Mixing 6

2.2.2 Language Identification 8

2.3 Future Trends 12

2.4 Summary 13

Chapter 3 Natural Language Processing and Machine Learning 14

3.1 Introduction 14

3.2 Artificial Intelligence 14

3.3 Natural Language Processing 15

3.3.1 Natural Language Tool Kit (NLTK) 16

3.3.2 Pandas 16

3.3.3 Numpy 17

3.4 Machine Learning 17

3.4.1 Supervised Learning 17

3.4.2 Weka and Classifiers used in development of models 18

3.4.2.1 Support Vector Machine (SVM) 19
3.4.2.2 Logistic Regression 20
3.4.2.3 Naïve Bayes 20
3.4.2.4 Decision Tree 22
3.4.2.5 Random Forest 23
3.5 Summary 24
Chapter 4 Novel Approach to Language Identification of Code-Mixing Text 25
4.1 Introduction 25
4.2 Hypothesis 25
4.3 Process 25
4.4 Input 26
4.5 Output 27
4.6 Users 27
4.7 Summary 27
Chapter 5 Design 28
5.1 Introduction 28
5.2 Language Identification System for Tamil-English Code-Mixed Text 28
5.2.1 Dataset Description 29
5.2.2 Preprocessing 29
5.2.3 Feature identification of Tamil-English code-mixed text 29
5.2.3.1 Tamil Unicode characters in Roman scripts 29
5.2.3.2 Language-specific dictionaries 31
5.2.3.3 Double consonants 31
5.2.3.4 Term Frequency 31
5.3 Language Identification System for Sinhala-English Code-Mixed Text 31
5.3.1 Dataset Description 32
5.3.2 Preprocessing 33
5.3.3 Feature identification of Sinhala-English code-mixed text 33
5.3.3.1 Sinhala Unicode Characters in Roman Scripts 33
5.3.3.2 Language-Specific Dictionaries 35
5.3.3.3 Term Frequency 35
5.4 Summary 35

Chapter 6 Implementation **36**
6.1 Introduction 36
6.2 Language Identification System for Tamil-English Code-Mixed Text 36
6.2.1 Preprocessing 36
6.2.2 Feature identification of Tamil-English code-mixed text 38
6.2.3 Model Development for Tamil-English code-mixed text 40
6.3 Language Identification System for Sinhala-English Code-Mixed Text 41
6.3.1 Preprocessing 41
6.3.2 Feature identification of Sinhala-English code-mixed text 43
6.3.3 Model Development for Sinhala-English code-mixed text 45
6.4 Summary 45
Chapter 7 Evaluation **46**
7.1 Introduction 46
7.2 Experimental design 46
7.2.1 Experimental design for Model Evaluation 46
7.2.1.1 Evaluation Strategy for predictive Models 47
7.2.2 Experimental design for Testing of Models 48
7.2.2.1 Evaluation Strategy for Testing Models 49
7.3 Experimental Results 49
7.3.1 Experiment Results for Model Evaluation 49
7.3.2 Experiment Results for Model Testing 53
7.4 Summary 53
Chapter 8 Conclusion and Future Work **54**
8.1 Introduction 54
8.2 Concluding remarks 54
8.3 Limitation and Future work 55
8.4 Summary 56
References **57**
Appendix A **60**
Appendix B **67**

List of Figures

Figure 2.1: The selection of optimal hyperplane with linear SVM 10

Figure 2.2: Bags of Words Example 12

Figure 3.1: Workflow of the Supervised Learning 18

Figure 3.2: Structure of a Decision Tree 22

Figure 4.1: Design Diagram of Language Identification System for Code-Mixed Text 26

Figure 5.2: Design Diagram of Language Identification System for Tamil-English Code-Mixed Text 28

Figure 5.3: Design Diagram of Language Identification System for Sinhala-English Code-Mixed Text 32

Figure 6.1: Sample of Tamil-English Code-Mixed Data 36

Figure 6.2: Sample of Tamil-English Code-Mixed words 37

Figure 6.3: Sample of Unique Tamil-English Code-Mixed words with Term Frequency 37

Figure 6.4: Sample of Annotated Tamil-English Code-Mixed Words with Language Tags 38

Figure 6.5: Sample Tamil word with features 39

Figure 6.6: Sample Tamil word with identified features with values 39

Figure 6.7: Sample of Sinhala-English Code-Mixed Data 41

Figure 6.8: Sample of Sinhala-English Code-Mixed words 42

Figure 6.9: Sample of Unique Sinhala-English Code-Mixed words with Term Frequency 42

Figure 6.10: Sample of Annotated Sinhala-English Code-Mixed Words with Language Tags 43

Figure 6.11: Sample of Sinhala word with features 44

Figure 6.12: Sample of Sinhala word with identified features with values 44

Figure 7.1: The knowledge flow of experiment process for model evaluation 46

Figure 7.2: The User Interface for testing finalized model for language identification of code-mixed text 48

Figure 7.3: Evaluation of features impotency for Tamil-English code-mixed text 52

Figure 7.4: Evaluation of features impotency for Sinhala-English code-mixed text 52

List of Tables

Table 1.1: Example Comment and language-tagged sentence for Tamil-English code-mixed text 3

Table 1.2: Example Comment and language-tagged sentence for Sinhala-English code-mixed text 3

Table 5.1: Statistics of Tamil- English Code-Mixed Dataset 29

Table 5.2: Tamil Unicode Characters 30

Table 5.3: Statistics of Sinhala-English Code-Mixed Dataset 32

Table 5.4: Sinhala Unicode Characters 34

Table 6.1: The parameter used in model development by different classifiers for Tamil-English code-mixed text 40

Table 6.2: The parameter used in model development by different classifiers for Sinhala-English code-mixed text 45

Table 7.1: The statistics of the dataset used for testing of models 49

Table 7.2: Overall results obtained from different classifiers for Tamil-English code-mixed text 50

Table 7.3: Overall results obtained from different classifiers for Sinhala-English code-mixed text 51

Table 7.4: Testing results obtained from Tanglish model and Singlish model 53

Chapter 1

Introduction

1.1 Prolegomena

Automatic analyzing and extracting useful information from the noisy social media content are currently getting more attention from the research community for NLP [1]. The type of noisy social media text is characterized by a high percentage of spelling mistakes with phonetic typing ("how is your amma and appa"; since 'amma', 'appa' are Tamil words), wordplay ('helloooo' for 'hello'), creative spelling ('Gud 9t' for 'good night'), abbreviations ('TC' for 'Take Care'), Meta tags (URLs), and so on. Currently, bilingual speakers use Unicode characters or Unicode characters written in Roman scripts to write in their own language and use phonetic typing, frequently add English elements through the combination of native languages to express their thoughts. These type of text is called code-mixed text.

Make automatic language identification a precondition for the complete process of text analysis on social media. Although language identification is considered an almost solved problem in different applications [2], language detectors fail within the context of social media such as code-mixing, phonetic typing and lexical borrowing [3]. For this reason, the complexity of the analysis and understanding of information will increase within the context of social media. The most reason for this limitation is due to the correct corpus acquisition. Automatically detect the boundaries of the language in a code-mixed social media text, for English-Bengali and English-Hindi has been proposed [3], [4].

This study focuses on effectively detect the language boundaries at word level of Tamil-English and Sinhala-English code-mixed noisy text. Natural language processing and machine learning techniques proposed to provide a solution. This study proposed a novel approach by adding Tamil Unicode characters in Roman scripts as new feature including dictionaries, double consonant, and term frequency as other features for Tamil-English code-mixed text and Sinhala Unicode characters written in Roman scripts as new features including dictionaries, and term frequency as other features for Sinhala-English code-mixed text.

1.2 Objectives

The objectives of this study are listed below:

- Critical study to understand the characteristics of code-mixed languages in the social media context
- Critical study of existing approaches in automatic language identification
- Design and develop a system for automatic language identification
- Evaluation of the proposed system

1.3 Background and Motivation

New forms of communication have greatly changed the types of traditional spoken and written languages [5]. These new forms are the result of the internet and social media in particular - Facebook, Twitter, etc. Typically, written languages tends towards a formal and defined structure, while spoken language is more casual and context dependent. With the advancement of technology in social media, the distinction between written and spoken languages has faded.

Scholars observed that the language across the internet, especially in synchronous communication resembled spoken communication. It was observed to be less formal, simpler and very similar to speech. This is the result of the conversion of casual vernacular to written form. People use words and symbols to express emotions, which results in inconsistent language generation across users since there is no defined structure for this usage [4].

In the 1940s and 1950s, code-mixing was often considered a sub-standard use of language. However, since the 1980s it has generally been recognized as a natural part of bilingual and multilingual language use. In social media communication, more than half of the users are bilingual speakers who often switch from one language to another to express their thoughts, especially in messages and comments on Facebook [6]. This language interchanging involves complex grammar complications in social media data itself. Code-mixing refers to the use of linguistic units from different languages in a single utterance or sentence.

Information retrieval deals with the issues of storing and retrieving information from all types of resources including social media which is very tough with regard to tokenizing and text processing. The impact of code mixing, creative spelling, phonetic typing, wordplay, abbreviations, Meta tags, and so on, social media contents become noisy. So automatic language identification become as necessary part for analyzing the content in social media [3].

As social media contains valuable information, due to the presence of above-mentioned problems, the complexity in analyzing the data increases. Even today there are no proper tools that deal with this type of data. So language identification at word level for code-mixing text became an as necessary and challenging task. The language identification system at word level of code-mixing text can be used as intimidate for chatbot application where the text written in native languages by using Roman scripts.

1.4 Code Mixing Problem in Brief

In social media bilingual speakers often mixed two languages to express their thoughts. When we consider the Sri Lanka local languages such as Sinhala and Tamil, most of the native speakers often mixed English with native languages. Most of the time people find it easily chat with other people in native language written in Roman scripts with the English word. The example Tamil-English and Sinhala- English code-mixed comment is shown in Table 1 and Table 2 respectively. For this reason, language identification at word level become a challenging task.

Table 1.1 Example Comment and language-tagged sentence for Tamil-English code-mixed text

Example Tamil-English Code-Mixed Sentence								
Text	super	anna	romba	days	ku	piraku	ellarajum	parkkuram
Tags	eng	tam	tam	eng	tam	tam	tam	tam

Table 1.2 Example Comment and language-tagged sentence for Sinhala-English code-mixed text

Example Sinhala-English Code-Mixed Comment							
Text	screen	eke	pena	okkoma	photo	ekata	wadinawa
Tags	eng	sin	sin	sin	eng	sin	sin

Also code-mixed noisy text is characterized by a high percentage of spelling mistakes with creative spelling (‘2morrw’ for ‘tomorrow’), phonetic typing (“how is your amma and appa”; since ‘amma’, ‘appa’ are Tamil words), wordplay (‘wooooow’ for ‘wow’), abbreviations (‘H R U’ for ‘How Are You’) and Meta tags (URLs). Because of the spelling mistakes the complexity in identifying language tags of code-mixed text at word level increases.

1.5 Proposed Solution

Effectively detect the language boundaries at word level of Tamil-English and Sinhala-English code-mixed noisy text identified as a research problem. Natural language processing and machine learning techniques proposed to provide a solution. Tamil-English and Sinhala-English code-mixed data from popular social media posts and comments took as input for this study. This study includes (a) Feature identification and feature selection of language identification system for Tamil-English and Sinhala-English code mixed data at word level; (b) Creation of annotated data set for collected Tamil-English and Sinhala-English code-mixed data; (c) implementing the language identification system for the Tamil-English and Sinhala-English code-mixed data, and (d) Evaluation of language identification with Tamil-English and Sinhala-English code-mixed data.

1.6 Resource Requirements

- Tamil-English and Sinhala-English code-mixed data from popular social media
- PC/Laptop with minimum 8GB of RAM and Intel i5 or i7 Processor
- The software is expected to run on platforms above Microsoft Windows 7

1.7 Structure of the Thesis

This thesis is divided into 8 chapters. Chapter 1 gave an overall introduction to the project. Chapter 2 critically reviews critically reviews the work done on code-mixing and language identification with background information and defining the research problem and identification of technologies. Chapter 3 is on technology adapted to building the natural language processing and machine learning based solution for language identification of code-mixing text in social media. Chapter 4 presents our approach to for language identification of code-mixing text in social media in terms of input, output, process, users and features of the system. Chapter 5 demonstrates the detailed design of the system. Chapter 6 contains the implementation of the components of modules given in the design stage. Chapter 7 reports on evaluation strategy with respect to the objectives of the project, experimental setup, and experimental results. Finally, Chapter 8 concludes the thesis with a note on the possible further work.

1.8 Summary

This chapter provided an introduction to the entire project. For this purpose, we have presented our research problem, objectives, technology adapted, proposed solution and resource requirements. Next chapter provides a detailed critical review of the work done on code-mixing and language identification with background information.

Chapter 2

Code Mixing in Social Media – Practices and Issues

2.1 Introduction

Chapter 1 gave an introduction to the overall project. To start with the research, it is important to find out the current state of the research in the world by reviewing other's work. This chapter presents a critical review of Language identification of code mixing text in social media with background information. This chapter formulate research problem and highlight the technology adopted towards a solution. In doing so, this chapter has been structured with two main sections, namely, State of the art of language identification of code-mixed text in social media and future trends.

2.2 State of the art of language identification of code-mixed text in social media

Code-mixing being a relatively new phenomenon has only attracted the attention of researchers in the last two decades. In the context of code-mixed social media data, new complications have been added to the Language Identification process. This section is described overall idea of code-mixing and critically review the work done on language identification of code-mixed text.

2.2.1 Code Mixing

Social media creates social interaction among people in which they share information and ideas in virtual communities and networks. One of social media features that are updated any time by users is status. Through status, the user can inform all activity, news, opinions, exchange ideas, business, and so on. In addition, they also are able to comment or respond to the latest status of their fellow social media users. The user of the social media sometimes mixes and uses several languages to update their status or comment to their friends' status, for example when they chat with other people at Facebook or web chat. The sociological and conversational needs behind the code-mixing and its linguistic nature were mainly focused on the linguistic efforts in the field [7].

Code-mixing is a process in which lexical items and grammatical features of two or more languages exists in the same sentence [8]. Spolsky commented that, it is common that people develop some knowledge and ability in a second language and so become bilingual [9]. The

simplest definition of a bilingual is a person who has some functional ability in a second language. This may vary from a limited ability in one or more domains, to very strong command of both languages.

The researcher concluded that the abbreviation like ‘CD’, ’DVD’, ’SMS’, ’VIP’ were used to make sentences simple and easy to understand. Furthermore, some adjectives like hot, cool, high etc. were used to produce stylish effect in sentences. Whereas duplication of words like ‘cute cute’, ‘high high’ pattern, it is not allowed in English grammar [10].

Two types of code-mixing can be categorized, namely Intra-sentential code-mixing and Inter-sentential code-mixing [4]. The code-mixing that takes place within sentence with no superficial change in topic is Intra-sentential code-mixing. An Intra-sentential code-mixing can occur in three processes: - Firstly, noun insertion: This code mixing involves the mixing of noun in one code in a sentence which is another code. Secondly, Verb insertion: This code mixing involves the mixing of verb in a sentence from one code and remaining parts are from another code. And finally, Clause and sentence insertion: complex sentence with different clause of another code reflects this code-mixing. The code-mixing that takes place when switching between native and second language to explain an incident is known as Inter-sentential code-mixing [11]. It was observed that word level code mixing occurred most of the times and at phrasal level it’s happened rarely. Also noted that the nouns were code-mixed mostly [12].

The users of Facebook have a tendency to use inter-sentential code mixing over intra-sentential code-mixing, and reported that 45% of the code mixing was initiated by real lexical needs, 40% was to talk about a particular topic, and 5% to clarify the content [13].

The major reasons for code mixing in Facebook explain here:

- 45% : Real lexical needs

  For instance someone is thinking of some object but is not able to recall the word in the language he/she is using already, then he/she will tend to switch to a language where he/she knows the appropriate word.

- 40% : Talking about a particular topic

  People tend to talk about some topics in their native language (like food) and generally while discussing science people tend to switch to English.

- 5% : For content clarification

  While explaining something, for better clarification of the audience, to make the audience more clear about the topic, code mixing is used.

Problems of storing and retrieving information from all types of data including social media, which is very difficult for tokenization and text processing are dealt by information retrievals [11]. Generally it is difficult to understand and analyze texts written in multiple languages. An evaluation metric was proposed to determine the complexity which occurs due to code-mixed social media texts which get developed rapidly due to multilingual interference [14].

Code-mixing for online data focused on the use of English and Arabic in e-mail communication by a group of young professionals and concluded that English was used more frequently for search on the Web. It was also discovered that a Romanized version of Egyptian-Arabic was used more often in informal e-mails, conversations and even to express personal content as opposed to classical Arabic [15]. Most of the people used Romanized version of native language script to express their thoughts in social media.

2.2.2 Language Identification

The first work on code-mixing processing was done by Joshi more than 30 years ago, in 1982 [16], while efforts to develop automatic language identification tools began earlier [17]. In its standard formulation, language identification assumes monolingual documents and attempts to classify each document according to its language from some closed set of known languages. Nevertheless, the solving of the problem of applying the language identification process to texts mixed in several languages, has only recently begun.

Automatic identification of word-level boundaries of different languages used in social media texts, illustrated by mixed English-Bengali and English-Hindi Facebook messages with standard techniques such as n-gram characters, dictionaries and, SVM classifiers[1]. Also, some different techniques were used, including a simple unsupervised dictionary-based approach, supervised word-level classification with and without contextual clues, and sequence labeling using Conditional Random Fields. The dictionaries-based approach is surpassed by supervised classification and sequence labeling, and it is important to consider contextual clues [3]. Classification of Hindi-English code-mixed data was performed to categorize the data into English,

Hindi, Mixed, Named Entity, Acronym, Universal, and undefined tags. Two types of embedding features were considered; character-based embedding features and word-based embedding features with the addition of context information. Support Vector Machine was used to train and test the system [18].

A word-based language identification system on mixed Turkish-Dutch messages randomly sampled from an online forum by comparing dictionary-based methods with language models and with logistic regression and conditional random fields with linear chain. This system achieved a high level of accuracy at the word level (97.6%), but with significantly lower accuracy at the post level (89. 5%), although 83% of the messages were actually monolingual [19]. Similarly, using a bilingual case using Spanish-English Twitter messages, uses only the ratio of the probability of words as a source of information and obtain good results, with a 96.9% accuracy at the word-level. However, the corpora are almost monolingual, so the result was obtained with a baseline of up to 92.3% [20].

The use of the most frequent word dictionaries is an established method in language identification. In this method, efficient and automatic segmentation of the input text in individual language blocks, in the case of multi-language documents [21]. But this method has a number of challenges, especially for social media, the text is full of noise. The general trend in dictionary-based methods is to keep only high-frequency words is for longer texts, and for the code-mixing situations it cannot be applicable. Because most of the times code-mixing text are short texts. In this situation avoid the most-frequent word lists and instead uses of the full-length dictionaries are better.

The probably most well-known language detection system is TextCat, which utilizes character-based n-gram models [22]. The method generates language specific n-gram profiles from the training corpus sorted by their frequency. A similar text profile is created from the text to be classified, and a cumulative “out-of-place” measure between the text profile and each language profile is calculated. The measure determines how far an n-gram in one profile is from its place in the other profile. Based on that distance value, a threshold is calculated automatically to decide the language of a given text. This approach has been widely used and is well established in language identification.

Word-level language detection from code-mixed text can be defined as a classification problem. Support vector machines (SVMs) are one of the most popular methods for text classification, largely because of the automatic weighing of a large number of functions [23]. SVM is currently the most successful machine learning technique across multiple domains [24].

A support vector machine (SVM) is a supervised learning technique which incorporates a learning algorithm for like gradient descent and is used for tasks like classification, pattern recognition and regression. The objective of the support vector machine algorithm is to find a hyperplane in an n-dimensional space (n- number of features) that distinctly classifies the data points. To separate the two classes of data points, there are many possible hyperplanes that could be chosen. The objective is to find a plane that has the maximum margin (the maximum distance between data points of both classes). Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence. Figure 2.1 shown the selection of optimal hyperplane with linear SVM.

Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, able to maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help to build the SVM.

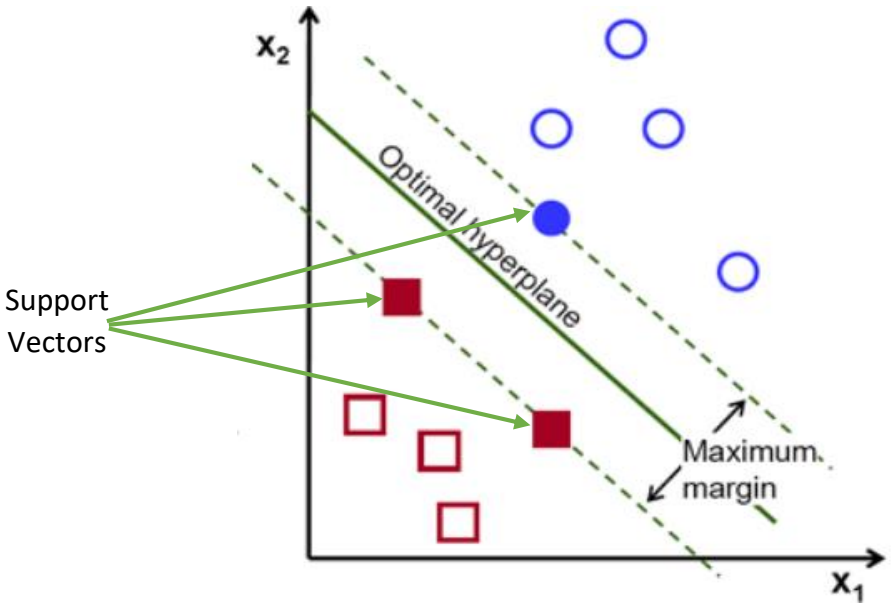


Figure 2.1 The selection of optimal hyperplane with linear SVM

Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds three.

Weakly semi-supervised methods are used to build the word level language identifier which considered the problem sequence labeling where monolingual text is used as samples data. The model which was performed well for this approach is continuous random field [25]. This classifier was used by Vyas in English-Hindi code-mixed text to calculate the confidence probability for each word [26].

By using a large Swahili-English internet corpus word-level language identification and prediction of code switch points for Swahili-English code-mixed done. Features such as tagging the language of words using char n-grams and capitalization set are taken and combined with label probabilities on the adjacent words [27].

Recognizing language from English, French, Dutch, German and Spanish microblog posts work was accomplished. This features like the content of attached links, language of blogger etc. and so forth was demonstrated to develop the accuracy of the system [28].

Language identification for four Twitter code-mixed languages such as Spanish-English, Nepali-English, Mandarin-English, and Modern Standard Arabic-Dialectal Arabic was organized as a shared task [29]. A CRF-based approach which uses stylometric features like word length, the presence of numbers, index, punctuations etc. was implemented by a team for the same shared task for which were used [30].

A feature-based system which obtained accuracy of 76% had the features like punctuations, prefixes, suffixes and so on along with SVM classifier was implemented with MSIR 2015 shared task data [31].

Bag of Words is a method to extract features from text documents. These features can be used for training machine learning algorithms. It creates a vocabulary of all the unique words occurring in all the documents in the training set [32]. A bag of words example shown in figure 2.2.

Figure 2.2 Bags of Words Example

2.3 Future Trends

Currently some of the researches focused on some language pairs like English-Bengali, Hindi-English, Telugu-English and so on, language identification of code mixed content in social media [1], [3], [4], [18]. But it is reasonable to experiment with other language pair like Tamil-English, Sinhala-English and so on with different types of social media content such as tweets, Facebook posts and messages. Also possible to do the sentiment analysis in code mixed content by using different language pairs. For the language identification incorporating other techniques and information sources are obvious targets for future work. In particular, to look at other machine learning methods, for example, to use a sequence learning method such as Conditional Random Fields and deep learning approaches [1], [4] . Also could be focus on cases with more than two languages, and languages that are typologically less distinct from each other or dialects [19].

2.4 Summary

This chapter provides a detailed critical review of analyzing the meaning of code mixing text in online social media with background information. Also formulate research problem and highlight the technology adopted towards a solution. Next chapter provides technology adapted to building the natural language processing based solution for analyzing the meaning of Tamil-English code mixing text in online social media.

Chapter 3

Natural Language Processing and Machine Learning

3.1 Introduction

Chapter 2 provides a detailed critical review of language identification of code mixing text in social media with background information. This chapter provides detail description of technology adapted to building the natural language processing and machine learning based solution for language identification of code mixing text in social media.

3.2 Artificial Intelligence

Artificial intelligence (AI) is an area of computer science that emphasizes the creation of intelligent machines that perform and react like humans which is concerned with automation of intelligent behavior. It is the simulation of intelligent behavior in computers and imitate intelligent human behavior. AI is accomplished by studying how human brain thinks, and how humans learn, decide, and work while trying to solve a problem [33]. It has become an essential part of the technology industry.

AI strives to build intelligent entities as well as understand them. AI can be defined in four ways. Those are as a system that: thinks like humans; acts like humans; thinks rationally; and acts rationally. In thinking like humans, it is the effort of making computers to think; machines with minds. In acting like humans, AI is the art of creating machines that perform functions that require intelligence when performed by people. AI as the study of the computations that make it possible to perceive, reason, and act is defined as thinking rationally. AI acting rationally is seeking to explain and emulate intelligent behavior in terms of computational processes [33].

Artificial Intelligence may be subdivided into two main branches or aspects. One aspect is cognitive science which has a strong affiliation with psychology. The aim of cognitive science is to construct programs for testing theories that describe and explain human intelligence. The other is machine intelligence which is more computer oriented and studies how to make computers behave intelligently [33].

Some of the core applications that artificial intelligence with computers are designed for include

- Knowledge
- Reasoning
- Problem solving
- Perception
- Learning
- Planning
- Ability to manipulate and move objects

Major areas of artificial intelligence categorize as

- Expert systems
- Neural networks
- Fuzzy logic
- Genetic algorithms
- Case-based reasoning
- Natural language processing
- Machine learning
- Computer vision
- Robotics
- Multi agent system

In this project two major areas of AI such as natural language processing and machine learning were used in development process. The description of natural language processing and machine learning technologies is explained in below sections.

3.3 Natural Language Processing

Natural language processing (NLP) is a technique which can be said as interaction between computers and human language. The ultimate goal of NLP is to enable computers to understand human language (English, Tamil, Sinhala, and etc.). NLP began in the 1950s as the intersection of artificial intelligence and linguistics. NLP was originally distinct from text information retrieval (IR), which employs highly scalable statistics-based techniques to index and search large volumes of text efficiently [34].

The two terms, Computational Linguistics (CL) and NLP, have often been used interchangeably. The difference is that CL tends more towards Linguistics, and answers linguistic questions using computational tools. Natural Language Processing involves applications that process language and tends more towards Computer Science. NLP is the art of solving engineering problems that need

to analyze the natural language. NLP is mainly used to help people navigate and digest large quantities of information that already exist in text form. It is also used to produce better user interfaces so that humans can better communicate with computers and with other humans. NLP is the driving force behind things like virtual assistants, speech recognition, language identification, sentiment analysis, automatic text summarization, machine translation and much more.

This mechanism of NLP comprises three processes which are natural language understanding, natural language generation and natural language interaction [34]. Natural Language Understanding (NLU) is a process which endeavors to understand the meaning of given text, nature and structure of each word by trying to resolve the ambiguity present in natural language and the meaning of each word is understood by using lexicons (vocabulary) and a set of grammatical rules. In the natural language generation process, the text is produced automatically from structured data in a readable format with meaningful phrases and sentences. The problem of natural language generation is tough to deal with.

Natural language interaction is somewhat a conflation of the technologies in which users communicate and evoke responses from systems via natural language. Humans are able to give a command either by typing it or speaking it. An automated voice back or a typed response from the computer or machine is the interaction piece that is generated in natural language interaction. In this research mainly focus on natural language interaction and identify the mixing of natural languages at word level which used to communicate in the social media.

3.3.1 Natural Language Tool Kit (NLTK)

NLTK is a leading platform for building Python programs to work with human language data. It provides text processing libraries for classification, tokenization, stemming, tagging, parsing, semantic reasoning and etc. [35]. In the implementation process nltk.word_tokenize( ) used to split the word from the input sentences.

3.3.2 Pandas

Pandas is an open source library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language [35]. In the implementation process pandas DataFrame structure used to store the two-dimensional labeled data structures with columns of potentially different types.

3.3.3 Numpy

This is a scientific computing library with support for multidimensional arrays and linear algebra, required for certain probability, tagging, clustering and classification tasks [35]. In the implementation process numpy savetxt() function was used to save the feature matrix values into a text file.

3.4 Machine Learning

Machine learning is a core part of AI which focused on algorithms that "learn" from data to construct models that can be used to make predictions and decisions. Data plays a vital role in machine learning and the learning algorithm is used to discover and learn properties of data. The quality and quantity of the training dataset will affect the learning and prediction performance on unseen data samples (test data) or future events.

Machine learning is generally composed of two important factors, modelling and optimization. Modelling means how to model the separating boundary or probability distribution of the given training set, and then the optimization techniques are used to seek the best parameters of the chosen model. Machine learning is also related to other disciplines such as artificial neural networks, pattern recognition, information retrieval, artificial intelligence, data mining, and function approximation, etc. [36].

There are mainly three categories of machine learning based on the problem and the dataset. Those categories are supervised learning, unsupervised learning and reinforcement learning. In this project focused on supervised learning method.

3.4.1 Supervised Learning

Supervised learning is learning with adequate supervision. The training set given for supervised learning is a labelled dataset. Supervised learning tries to find the relationships between the feature set and the label set, which represent the knowledge and properties of the dataset. The supervised learning work flow shown in figure 3.1. Classification and regression are the two types of supervised learning. Classification determines the category an object belongs to and regression deals with obtaining a set of numerical input or output examples, thereby discovering functions enabling the generation of suitable outputs from respective inputs [36].

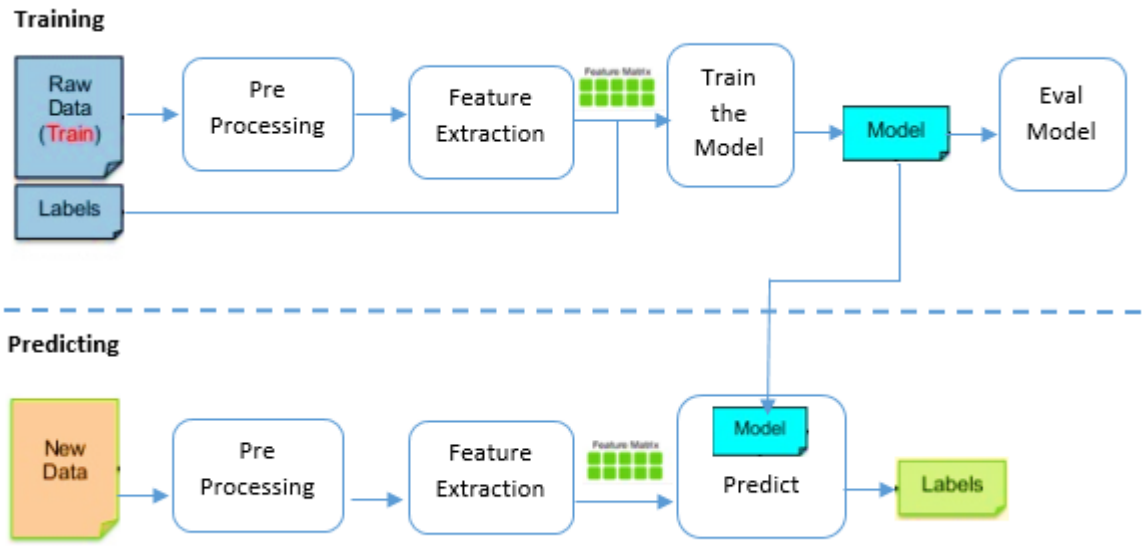


Figure 3.1 Workflow of the Supervised Learning

3.4.2 Weka and Classifiers used in development of models

Weka was developed at the University of Waikato in New Zealand; the name stands for Waikato Environment for Knowledge Analysis The system is written in Java and distributed under the terms of the GNU General Public License [37]. It provides a uniform interface to many different learning algorithms, along with methods for pre and post processing and for evaluating the result of learning schemes on any given dataset. Weka provides implementations of learning algorithms that can be easily apply to dataset.

Weka supports several standard data mining tasks, more specifically, data preprocessing, clustering, classification, regression, visualization, and feature selection. All of Weka's techniques are predicated on the assumption that the data is available as a single flat file or relation, where each data point is described by a fixed number of attributes.

In the terminology of machine learning, classification is considered an instance of supervised learning, learning where a training set of correctly identified observations is available. An algorithm that implements classification, especially in a concrete implementation, is known as a classifier. In this research different machine learning Classifiers such as Support Vector Machine, Naïve Bayes, Logistic Regression, Random Forest and Decision Tree used to evaluate the performance of models.

3.4.2.1 Support Vector Machine (SVM)

In Weka support vector machine implements the sequential minimal optimization (SMO) algorithm for training a support vector classifier [38]. SMO is an algorithm for solving the quadratic programming (QP) problem that arises during the training of support vector machines (SVM). SMO breaks this large QP problem into a series of smallest possible QP problems. These small QP problems are solved analytically, which avoids using a time-consuming numerical QP optimization as an inner loop. The amount of memory required for SMO is linear in the training set size, which allows SMO to handle very large training sets. SMO is fastest for linear SVMs and sparse data sets. This implementation globally replaces all missing values and transforms nominal attributes into binary ones. It also normalizes all attributes by default. SMO solved multi-class problems using pairwise classification.

Consider a binary classification problem with a dataset (x1, y1), ..., (xn, yn), where xi is an input vector and yi ∈ {-1, +1} is a binary label corresponding to it. A soft-margin support vector machine is trained by solving a quadratic programming problem, which is expressed in the dual form as follows:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j K(x_i, x_j) \alpha_i \alpha_j,$$

subject to:

$$0 \leq \alpha_i \leq C, \quad \text{for } i = 1, 2, \ldots, n,$$

$$\sum_{i=1}^{n} y_i \alpha_i = 0$$

Where C is an SVM hyperparameter and $K(x_i, x_j)$ is the kernel function, and the variables α_i are Lagrange multipliers.

SMO is an iterative algorithm for solving the optimization problem. SMO breaks the problem into a series of smallest possible sub-problems, which are then solved analytically. Because of the linear equality constraint involving the Lagrange multipliers α_i, the smallest possible problem involves two such multipliers. Then, for any two multipliers α_1 and α_2, the constraints are reduced to:

$$0 \leq \alpha_1, \alpha_2 \leq C,$$
$$y_1\alpha_1 + y_2\alpha_2 = k,$$

And this reduced problem can be solved analytically: one needs to find a minimum of a one-dimensional quadratic function. k is the negative of the sum over the rest of terms in the equality constraint, which is fixed in each iteration.

So in the SMO algorithm the complexity parameter(c) and the kernel parameter are the important parameters.

3.4.2.2 Logistic Regression

Logistic regression using a multinomial logistic regression model with a ridge estimator [39]. Logistic regression capable to deals with binary class, missing class values and nominal class.

If there are k classes for n instances with m attributes, the parameter matrix B to be calculated will be an m*(k-1) matrix. The probability for class j with the exception of the last class is

Pj(Xi) = exp(XiBj)/((sum[j=1..(k-1)]exp(Xi*Bj))+1)

The last class has probability

1-(sum[j=1..(k-1)]Pj(Xi)) = 1/((sum[j=1..(k-1)]exp(Xi*Bj))+1)

The (negative) multinomial log-likelihood is thus:

```
L = -sum[i=1..n]{
                    sum[j=1..(k-1)](Yij * ln(Pj(Xi)))
                    +(1 - (sum[j=1..(k-1)]Yij))
                    * ln(1 - sum[j=1..(k-1)]Pj(Xi))
                    } + ridge * (B^2)
```

So in the logistic regression ridge parameter is the most important parameter, which uses log-likelihood to assign the ridge value.

3.4.2.3 Naïve Bayes

Naive Bayes is a simple, yet effective and commonly-used, machine learning classifier. Naive Bayes classifier using estimator classes. Numeric estimator precision values are chosen based on analysis of the training data [40]. Naïve Bayes implements the probabilistic Naïve Bayesian

classifier based on applying Bayes' theorem with strong (naive) independence assumptions between the features [41].

Bayes' theorem states the following relationship, given class variable y and dependent feature vector x_1 through x_n:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots x_n \mid y)}{P(x_1, \ldots, x_n)}$$

Using the naive conditional independence assumption that

$$P(x_i|y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i|y)$$

For all i, this relationship is simplified to

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \ldots, x_n)}$$

Since P(x1,…,xn) is constant given the input, can able to use the following classification rule:

$$P(y \mid x_1, \ldots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

$$\Downarrow$$

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y),$$

This is referred to as the Maximum A Posteriori decision rule. It can be used to estimate P(y) and $P(x_i|y)$, the former is then the relative frequency of class y in the training set.

3.4.2.4 Decision Tree

The C4.5 algorithm is used as a Decision Tree Classifier which can be employed to generate a decision, based on a certain sample of data [42]. This is a standard algorithm that is widely used for practical machine learning. Part is a more recent scheme for producing sets of rules called "decision lists"; it works by forming partial decision trees and immediately converting them into the corresponding rule. C4.5 uses Information gain. A decision tree consists of four parts named, root, internal nodes, branches and leaf nodes as represented in Figure 3.2.

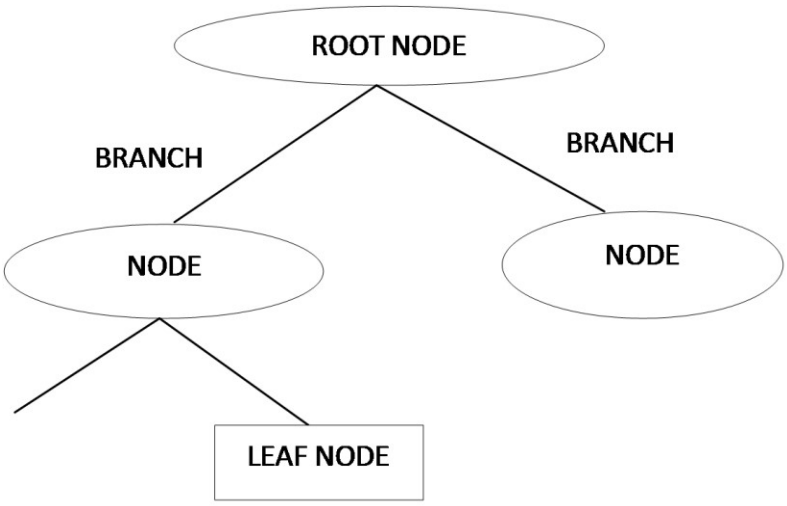


Figure 3.2 Structure of a Decision Tree

Decision tree use a divide-and-conquer strategy in order to partition the instance space into decision regions. A decision tree is drawn upside down with its root at the top. The root of the tree asks one of the feature questions to splits datasets into multiple branches based on the answer of the parent question. If all the subsets of data in the branch have the same label, that branch will. It will not grow from that branch any further. Otherwise, the constructor algorithm can split the data further with new conditions from other features of the data sets. The algorithm will recursively try to create new branches of the tree based on the features of datasets. Basically algorithm try to understand which features effect the decision for labelling. So, from the tree structure the importance of the features and relations in between them can be clearly and easily viewed.

3.4.2.5 Random Forest

Random forest is a popular type of ensemble machine learning algorithm related with decision tree algorithm which can be used for both classification and regression problems. Ensemble classifier is a combination of weak classifiers in order to produce a strong classifier with the idea that the predictions from multiple machine learning algorithms together to make more accurate predictions than any individual model [43]. In this research, random forest algorithm was used as a supervised classification algorithm. Same as, more trees in the forest makes the forest looks more robust, the random tree classifier gives higher accuracy when there is higher number of trees. Random Forests grows many classification trees. To classify a new object from an input vector, put the input vector down each of the trees in the forest. Each tree gives a classification, and it is said to be tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

In random forest algorithm, it starts with randomly selecting ‘k’ number of features out of the total ‘m’ features in the data set provided. In the next stage, find the root node using the selected ‘k’ features using the best split approach. Because randomly selecting the feature for the root node may give the bad result with low accuracy. As the third stage, find the daughter nodes using the same best fit approach. These first three stages are repeatedly done until a target value is obtained as the leaf node. In the final stage, the above four stages are repeatedly done to create ‘n’ number of randomly created trees hence it forms the random forest.

Information gain is a popular measures used in best fit approach when finding the most relevant attributes for each nodes while constructing the tree in the RandomForest classifier in WEKA.

Information gain is how much information can be gained by doing the split using a particular feature. Before calculating the Information Gain, entropy should be calculated. Entropy is a measure of the uncertainty or impurity associated with a random variable.

For given a set of examples D, the original entropy of the dataset is computed as:

$$H[D] = -\sum_{j=1}^{c} P(c_j) log_2 P(c_j)$$

where c is the set of desired class.

If calculate entropy for attribute Ai, with v values thinking to put the Ai feature for the root of the current tree, this will partition D into v subsets D1, D2 …Dv. The expected entropy if Ai is used as the current root is given by

$$H_{A_i}[D] = \sum_{j=1}^{v} \frac{|D_j|}{|D|} H[D_j]$$

Therefore information gained by selecting attribute Ai to branch or to partition the data is given by

$$IG(D, A_i) = H[D] - H_{Ai}[D]$$

Likewise, after calculating information gain for all the attributes that are remaining in the hand, the attribute with the highest gain is chosen to branch/split the current tree.

When do prediction from a random forest algorithm, it takes the input features and send through every decision tree in the random forest while storing the output from each tree. Then votes to different output will be calculated and this is called majority voting concept. Predicted output with the highest votes will be the final prediction of the whole algorithm.

3.5 Summary

This chapter briefly discussed the technologies adopted in the development of word level language identification of code-mixing text in social media. Natural language processing and machine learning are the key technical factors that handled. Next chapter reveals about the novel approach of word level language identification of code-mixing text in social media with the use of NLP and ML technologies.

Chapter 4

Novel Approach to Language Identification of Code-Mixing Text

4.1 Introduction

Chapter 3 presented a broad overview of the technologies that have been used for language identification for Tamil-English and Sinhala-English code mixed text at word level in social media. This chapter provides the novel approach to language identification of Tamil-English and Sinhala-English code-mixing text at the word level in social media with the use of NLP and ML technologies. As such the chapter structure with subsections namely, hypothesis, process, input, output, and users of the solution. In the description, the section on process gives the overall functionality of the system together with relevant technologies.

4.2 Hypothesis

This research hypothesized that the issue of effectively detect the language boundaries of Tamil-English and Sinhala-English code-mixed noisy text at the word level, addressed by the use of NLP and ML technologies.

4.3 Process

In the proposed solution firstly, raw Tamil-English and Sinhala-English code-mixed sentences led to preprocessing steps. The preprocessing steps include tokenization, convert the word in lowercase, removing wordplay characters, symbol removal and calculating the term frequency of each word.

Then the pre-process data took as input for feature identification module for extracting features from words. Feature identification plays a significant part of this study. This section analyzed the pattern of the characters appears in the words from the Tamil- English and Sinhala-English code-mixed dataset and identified relevant features among it. Tamil Unicode characters in Roman scripts, dictionaries, double consonant, and term frequency used as features for Tamil-English code-mixed text. Sinhala Unicode characters written in Roman scripts, dictionaries, and term frequency used as features for Sinhala-English code-mixed text.

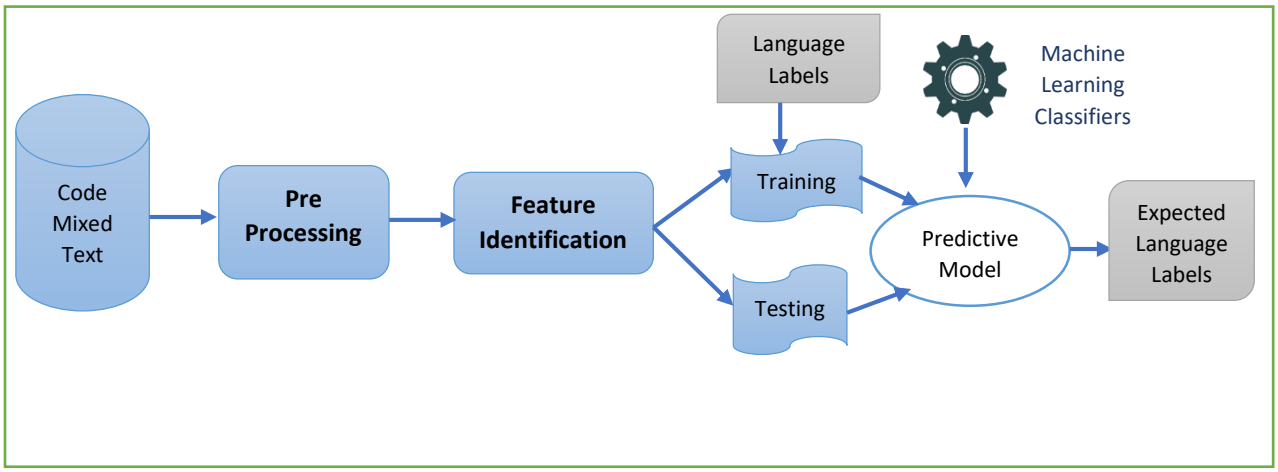


Figure 4.1 Design Diagram of Language Identification System for Code-Mixed Text

Then, the feature-based embedding model was created according to selected classifiers in Weka with identified features. Different machine learning classifiers such as Support Vector Machine, Naïve Bayes, Logistic Regression, Random Forest and Decision Tree used to create the models. Ten-fold cross-validation used to evaluate the performance based on language tags at the word level. The output was recorded based on accuracy and the F-Measure value of each tag. Based on the accuracy and F-Measure values suitable machine learning models were selected for Tamil-English and Sinhala-English code-mixed dataset. These suitable models were used to identify the language tags at word level for the new Tamil-English and Sinhala-English code-mixed sentences. The design diagram of language identification system for code-mixed text shown in figure 4.1.

Jupiter notebook 5.5.0 with Python 3.6.4 with Pandas, nltk and numpy libraries were used in preprocessing and feature identification process. For the training, testing and model creation process Weka 3.6 software was used. Java jdk 1.8, NetBeans IDE 8.0.2 and weka-stable-3.8.0 library were used to predict the language labels of new words for Tamil-English and Sinhala-English sentences at word level.

4.4 Input

In the proposed system, Facebook comments and post with Tamil-English code-mixed text and Sinhala-English code-mixed text were taken as separate inputs. All the posts and comments broke down into sentences. Among that 2,000 Tamil-English code mixed sentences taken to the Tamil-English code mixed dataset and 1,000 Sinhala-English code mixed sentences taken to the Sinhala-

English code mixed dataset. There were three language tags used to annotate the Tamil-English code-mixed dataset. They are ‘tam’ for Tamil words, ‘eng’ for English words and ‘rest’ for other words. Likewise for the Sinhala-English code-mixed dataset annotated with three language tags ‘sin’ for Sinhala words, ‘eng’ for English words and ‘rest’ for other words.

4.5 Output

The output of the proposed system is language identified sentences at word level both in Tamil and English for Tamil-English code-mixed text and, language identified sentences at word level both in Sinhala and English for Sinhala-English code-mixed text.

4.6 Users

Machine learning researchers who used Tamil-English and Sinhala-English code-mixed data for researches and developers who use Tamil and English native languages text is written in the Roman script as an input for chatbot application are the users of this system.

4.7 Summary

This chapter focuses on the hypothesis, the approach with input and output along with the process, which will be carried upon. Users of the system also discussed. Next chapter discusses the system design of the proposed system with a significant focus on elaborating the process mentioned in this chapter.

Chapter 5

Design

5.1 Introduction

Chapter 4 focused on the novel approach to language identification of Tamil-English and Sinhala-English code-mixing text at the word level in social media with the use of NLP and ML technologies. The design of language identification system for Tamil-English code-mixed text and Sinhala-English code-mixed text is explained in this chapter. The design diagram mainly focus on preprocessing, feature identification, training and testing of predictive model with expected language labels. The supervised learning used as machine learning methodology.

5.2 Language Identification System for Tamil-English Code-Mixed Text

The design diagram of language identification system for Tamil-English code-mixed text shown in Figure 5.2. In this system raw Tamil-English code-mixed sentences led to preprocessing steps. Then the pre-process data took as input for feature identification module for identifying features from words. After that the predictive model created according to selected classifiers in Weka. Ten-fold cross-validation used to evaluate the performance based on language tags at word level.

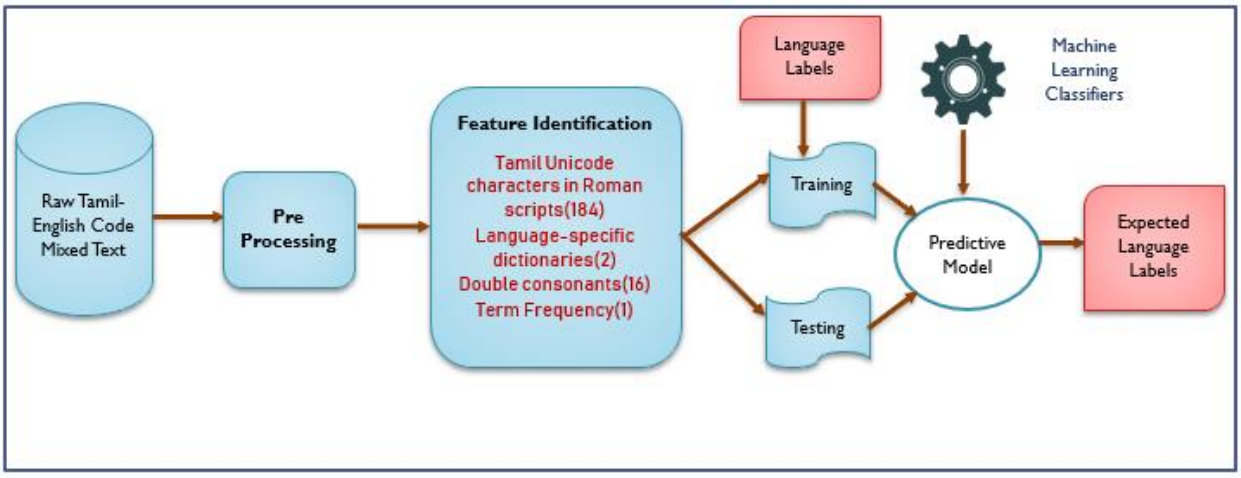


Figure 5.2 Design Diagram of Language Identification System for Tamil-English Code-Mixed Text

5.2.1 Dataset Description

In the proposed system, Facebook comments and post with Tamil-English code mixed text taken as input. All the posts and comments broke down into sentences. Among this 2,000 Tamil-English code mixed sentences taken as training dataset. There were 17,603 tokens identified from the sentences. Among that 8,607 unique words were identified in the training dataset. The statistics of Tamil-English code-mixed dataset used for training shown in Table 5.1.

Table 5.1 Statistics of Tamil- English Code-Mixed Dataset

Tamil-English code-mixed data	**Train Data**
Sentences	2,000
Tokens	17,603
Words	8,607

The Tamil-English code-mixed dataset annotated with three main language tags, “tam” for Tamil words, “eng” for English words and “rest” for all other words. The “rest” tag includes Named Entities, Acronyms, Universal, mixed and other language tags.

5.2.2 Preprocessing

In preprocessing raw Tamil-English code-mixed sentences led to preprocessing steps. The preprocessing steps include tokenization, convert the word in lowercase, removing wordplay characters, symbol removal and calculating the term frequency of each word.

5.2.3 Feature identification of Tamil-English code-mixed text

The below features identified for language identification system for Tamil-English code-mixed text.

5.2.3.1 Tamil Unicode characters in Roman scripts

Tamil characters written in Roman scripts was taken as important features to identify Tamil words. For example “க” in Tamil Unicode character written as “ka” in Roman scripts. There is totally 247 Unicode letters in the Tamil language. The Tamil Unicode characters like “ல” and “ள” written same as “la” in Roman scripts. Among 247 Unicode characters repeating 39 characters with same Roman script were remove and 208 characters were taken as features.

In addition to that there are some characters like “ஜ”, “ஷ”, “ஹ”, “ஸ்ரீ” used in the Tamil language which are derived from other languages. Among that 53 Unicode characters also taken as features. All together 261 Unicode characters were taken as 261 features. The features are taken from Tamil Unicode characters shown in Table 5.2.

Table 5.2 Tamil Unicode Characters

TAMIL ALPHABETS - தமிழ் எழுத்துக்கள்												
அ a	ஆ aa	இ i	ஈ ii	உ u	ஊ uu	எ e	ஏ ae	ஐ ai	ஒ o	ஓ oa	ஔ ow	ஃ q
க ka	கா kaa	கி ki	கீ kii	கு ku	கூ kuu	கெ ke	கே kae	கை kai	கொ ko	கோ koa	கௌ kow	க் k
ங nGa	ஙா nGaa	ஙி nGi	ஙீ nGii	ஙு nGu	ஙூ nGuu	ஙெ nGe	ஙே nGae	ஙை nGai	ஙொ nGo	ஙோ nGoa	ஙௌ nGow	ங் nG
ச sa	சா saa	சி si	சீ sii	சு su	சூ suu	செ se	சே sae	சை sai	சொ so	சோ soa	சௌ sow	ச் s
ஞ Gna	ஞா Gnaa	ஞி Gni	ஞீ Gnii	ஞு Gnu	ஞூ Gnuu	ஞெ Gne	ஞே Gnae	ஞை Gnai	ஞொ Gno	ஞோ Gnoa	ஞௌ Gnow	ஞ் Gn
ட ta	டா taa	டி ti	டீ tii	டு tu	டூ tuu	டெ te	டே tae	டை tai	டொ to	டோ toa	டௌ tow	ட் t
ண Na	ணா Naa	ணி Ni	ணீ Nii	ணு Nu	ணூ Nuu	ணெ Ne	ணே Nae	ணை Nai	ணொ No	ணோ Noa	ணௌ Now	ண் N
த tha	தா thaa	தி thi	தீ thii	து thu	தூ thuu	தெ the	தே thae	தை thai	தொ tho	தோ thoa	தௌ thow	த் th
ந nha	நா nhaa	நி nhi	நீ nhii	நு nhu	நூ nhuu	நெ nhe	நே nhae	நை nhai	நொ nho	நோ nhoa	நௌ nhow	ந் nh
ப pa	பா paa	பி pi	பீ pii	பு pu	பூ puu	பெ pe	பே pae	பை pai	பொ po	போ poa	பௌ pow	ப் p
ம ma	மா maa	மி mi	மீ mii	மு mu	மூ muu	மெ me	மே mae	மை mai	மொ mo	மோ moa	மௌ mow	ம் m
ய ya	யா yaa	யி yi	யீ yii	யு yu	யூ yuu	யெ ye	யே yae	யை yai	யொ yo	யோ yoa	யௌ yow	ய் y
ர ra	ரா raa	ரி ri	ரீ rii	ரு ru	ரூ ruu	ரெ re	ரே rae	ரை rai	ரொ ro	ரோ roa	ரௌ row	ர் r
ல la	லா laa	லி li	லீ lii	லு lu	லூ luu	லெ le	லே lae	லை lai	லொ lo	லோ loa	லௌ low	ல் l
வ va	வா vaa	வி vi	வீ vii	வு vu	வூ vuu	வெ ve	வே vae	வை vai	வொ vo	வோ voa	வௌ vow	வ் v
ழ za	ழா zaa	ழி zi	ழீ zii	ழு zu	ழூ zuu	ழெ ze	ழே zae	ழை zai	ழொ zo	ழோ zoa	ழௌ zow	ழ் z
Grantham (Sanskrit Characters) - கிரந்த எழுத்துக்கள்												
ஜ ja	ஜா jaa	ஜி ji	ஜீ jii	ஜு ju	ஜூ juu	ஜெ je	ஜே jae	ஜை jai	ஜொ jo	ஜோ joa	ஜௌ jow	ஜ் j
ஷ sha	ஷா shaa	ஷி shi	ஷீ shii	ஷு shu	ஷூ shuu	ஷெ she	ஷே shae	ஷை shai	ஷொ sho	ஷோ shoa	ஷௌ show	ஷ் sh
ஹ ha	ஹா haa	ஹி hi	ஹீ hii	ஹு hu	ஹூ huu	ஹெ he	ஹே hae	ஹை hai	ஹொ ho	ஹோ hoa	ஹௌ how	ஹ் h
க்ஷ ksha	க்ஷா kshaa	க்ஷி kshi	க்ஷீ kshii	க்ஷு kshu	க்ஷூ kshuu	க்ஷெ kshe	க்ஷே kshae	க்ஷை kshai	க்ஷொ ksho	க்ஷோ kshoa	க்ஷௌ kshow	க்ஷ் ksh
												ஸ்ரீ sri

Source: Tamil Alphabets - Tamilcube “http://tamilcube.com/learn-tamil/tamil-alphabets-chart.aspx”

5.2.3.2 Language-specific dictionaries

In order to identify the English words, two corpora were used as dictionaries. The corpora are British National Corpus (BNC) and LEXNORM corpus. Existence of a word in these two corpora were taken as two features.

- BNC: A computer corpus of 100 million words of British English, written and spoken [44].
- LEXNORM: A lexical normalization dataset released by Han et al. (2012) [45] . This dataset used to identify the spelling variations are expected in social media data.

5.2.3.3 Double consonants

By analyzed the words in collected Tamil-English code-mixed dataset some of the double consonant pattern letter was identified as features. The 16 identified double consonants were 'nn', 'ee', 'll', 'kk', 'tt', 'pp', 'mm', 'yy', 'rr', 'oo', 'cc', 'ss', 'dd', 'ff', 'bb', 'gg'.

5.2.3.4 Term Frequency

The frequency of each unique word occurs in the Tamil-English code-mixed dataset was taken as a feature.

Eventually, 280 features were taken for language identification system for Tamil-English code-mixed text.

5.3 Language Identification System for Sinhala-English Code-Mixed Text

The design diagram of language identification system for Sinhala-English code-mixed text shown in Figure 5.3. In this system raw Sinhala-English code-mixed sentences led to preprocessing steps. Then the pre-process data took as input for feature identification module for identifying features from words. After that the predictive model created according to selected classifiers in Weka. Ten-fold cross-validation used to evaluate the performance based on language tags at word level.

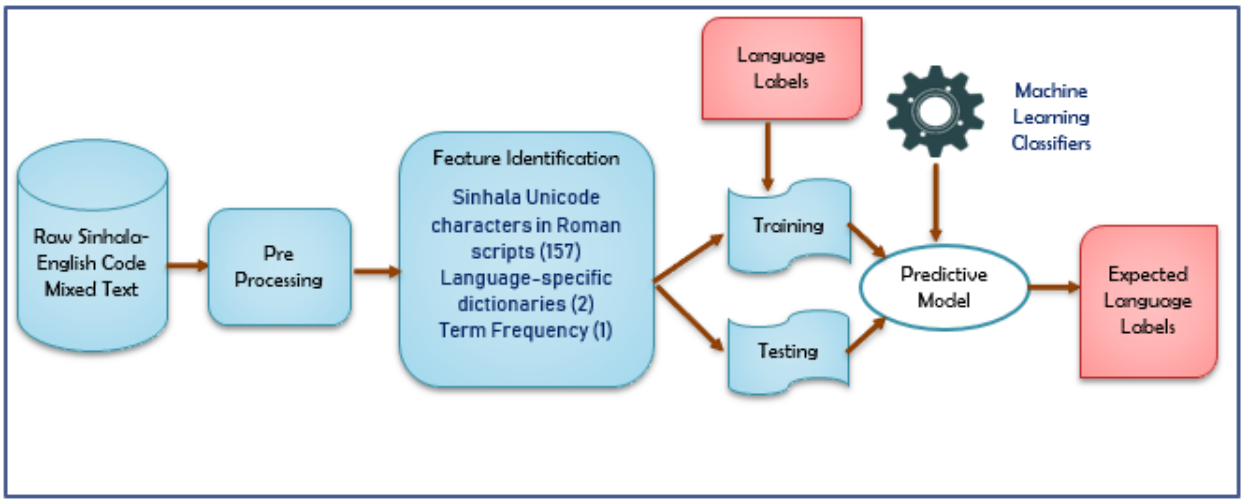


Figure 5.3 Design Diagram of Language Identification System for Sinhala-English Code-Mixed Text

5.3.1 Dataset Description

In the proposed system, Facebook comments and post with Sinhala-English code mixed text was taken as input. All the posts and comments broke down into sentences. Among this 1,000 Sinhala-English code mixed sentences was taken as training set. There were 6,640 tokens identified from the sentences. Among that 2,896 unique words were identified in the training dataset. The statistics of Sinhala-English code-mixed dataset used for training shown in Table 5.3.

Table 5.3 Statistics of Sinhala-English Code-Mixed Dataset

Sinhala-English code-mixed data	Train Data
Sentences	1,000
Tokens	6,640
Words	2,896

The Sinhala-English code-mixed dataset annotated with three main language tags, “sin” for Sinhala words, and “eng” for English words and “rest” for all other words. The “rest” tag includes Named Entities, Acronyms, Universal, mixed and other language tags.

5.3.2 Preprocessing

In preprocessing raw Sinhala-English code-mixed sentences led to preprocessing steps. The preprocessing steps include tokenization, convert the word in lowercase, removing wordplay characters, symbols removal and calculating the term frequency of each word.

5.3.3 Feature identification of Sinhala-English code-mixed text

The below features identified for language identification system for Sinhala-English code-mixed text.

5.3.3.1 Sinhala Unicode Characters in Roman Scripts

Sinhala characters written in Roman scripts was taken as important features to identify Sinhala words. For example "ග" in Sinhala Unicode character written as "ga" in Roman scripts. The most common 284 Unicode characters used in writing convert to Roman script. The Sinhala Unicode characters like "අ" and "අැ" written same as "a" in Roman scripts in lower case. Also the "ඳ්"and "ණ්" characters written same as "n" in Roman scripts. So among 284 Sinhala Unicode characters written in Roman script, repeating 51 characters with same Roman script were remove and 233 Sinhala Unicode characters written in Roman scripts were taken as features. The features are taken from Sinhala Unicode characters shown in Table 5.4.

Table 5.4 Sinhala Unicode Characters

Sinhala Alphabets														
a අ	aa ආ	A ඇ	ae ඈ	i ඉ	ii ඊ	u උ	uu ඌ	e එ	ea ඒ	l ළු	o ඔ	oe ඕ	au ඖ	
ka ක	kaa කා	kA කැ	kae කෑ	ki කි	kii කී	ku කු	kuu කූ	ke කෙ	kea කේ	kl කෙළ	ko කො	koe කෝ	kau කෞ	k ක්
ba බ	baa බා	bA බැ	bae බෑ	bi බි	bii බී	bu බු	buu බූ	be බෙ	bea බේ	bl බෙළ	bo බො	boe බෝ	bau බෞ	b බ්
ga ග	gaa ගා	gA ගැ	gae ගෑ	gi ගි	gii ගී	gu ගු	guu ගූ	ge ගෙ	gea ගේ	gl ගෙළ	go ගො	goe ගෝ	gau ගෞ	g ග්
ma ම	maa මා	mA මැ	mae මෑ	mi මි	mii මී	mu මු	muu මූ	me මෙ	mea මේ	ml මෙළ	mo මො	moe මෝ	mau මෞ	m ම්
cha ච	chaa චා	chA චැ	chae චෑ	chi චි	chii චී	chu චු	chuu චූ	che චෙ	chea චේ	chl චෙළ	cho චො	choe චෝ	chau චෞ	ch ච්
ya ය	yaa යා	yA යැ	yae යෑ	yi යි	yii යී	yu යු	yuu යූ	ye යෙ	yea යේ	yl යෙළ	yo යො	yoe යෝ	yau යෞ	y ය්
ja ජ	jaa ජා	jA ජැ	jae ජෑ	ji ජි	jii ජී	ju ජු	juu ජූ	je ජෙ	jea ජේ	jl ජෙළ	jo ජො	joe ජෝ	jau ජෞ	j ජ්
ra ර	raa රා	rA රැ	rae රෑ	ri රි	rii රී	ru රු	ruu රූ	re රෙ	rea රේ	rl රෙළ	ro රො	roe රෝ	rau රෞ	r ර්
ta ට	taa ටා	tA ටැ	tae ටෑ	ti ටි	tii ටී	tu ටු	tuu ටූ	te ටෙ	tea ටේ	tl ටෙළ	to ටො	toe ටෝ	tau ටෞ	t ට්
la ල	laa ලා	lA ලැ	lae ලෑ	li ලි	lii ලී	lu ලු	luu ලූ	le ලෙ	lea ලේ	ll ලෙළ	lo ලො	loe ලෝ	lau ලෞ	l ල්
da ඩ	daa ඩා	dA ඩැ	dae ඩෑ	di ඩි	dii ඩී	du ඩු	duu ඩූ	de ඩෙ	dea ඩේ	dl ඩෙළ	do ඩො	doe ඩෝ	dau ඩෞ	d ඩ්
wa ව	waa වා	wA වැ	wae වෑ	wi වි	wii වී	wu වු	wuu වූ	we වෙ	wea වේ	wl වෙළ	wo වො	woe වෝ	wau වෞ	w ව්
tha ත	thaa තා	thA තැ	thae තෑ	thi ති	thii තී	thu තු	thuu තූ	the තෙ	thea තේ	thl තෙළ	tho තො	thoe තෝ	thau තෞ	th ත්
sa ස	saa සා	sA සැ	sae සෑ	si සි	sii සී	su සු	suu සූ	se සෙ	sea සේ	sl සෙළ	so සො	soe සෝ	sau සෞ	s ස්
dha ද	dhaa දා	dhA දැ	dhae දෑ	dhi දි	dhii දී	dhu දු	dhuu දූ	dhe දෙ	dhea දේ	dhl දෙළ	dho දො	dhoe දෝ	dhau දෞ	dh ද්
ha හ	haa හා	hA හැ	hae හෑ	hi හි	hii හී	hu හු	huu හූ	he හෙ	hea හේ	hl හෙළ	ho හො	hoe හෝ	hau හෞ	h හ්
na න	naa නා	nA නැ	nae නෑ	ni නි	nii නී	nu නු	nuu නූ	ne නෙ	nea නේ	nl නෙළ	no නො	noe නෝ	nau නෞ	n න්
Na ණ	Naa ණා	NA ණැ	Nae ණෑ	Ni ණි	Nii ණී	Nu ණු	Nuu ණූ	Ne ණෙ	Nea ණේ	Nl ණෙළ	No ණො	Noe ණෝ	Nau ණෞ	N ණ්

Source: Real Time Unicode Converter - UCSC – University of Colombo

5.3.3.2 Language-Specific Dictionaries

In order to identify the English words, two corpora were used as dictionaries. The corpora are British National Corpus (BNC) and LEXNORM corpus. Existence of a word in these two corpora were taken as two features.

- BNC: A computer corpus of 100 million words of British English, written and spoken [44].
- LEXNORM: A lexical normalization dataset released by Han et al. (2012) [45]. This dataset used to identify the spelling variations are expected in social media data.

5.3.3.3 Term Frequency

The frequency of each unique word occurs in the Sinhala-English code-mixed dataset was taken as a feature.

Eventually, 236 features were taken for language identification system for Sinhala-English code-mixed text.

5.4 Summary

This chapter discussed the design of the system. Design for language identification system for Tamil-English code-mixed text and design for language identification system for Sinhala-English code-mixed text is discussed. Feature identification plays a significant part of this project. The feature identification from Tamil-English code-mixed text and Sinhala-English code-mixed text are discussed. Implementation of the proposed design is discussed in the next chapter.

Chapter 6

Implementation

6.1 Introduction

Chapter 5 discussed detailed design of language identification system for Tamil-English code-mixed text and language identification system for Sinhala-English code-mixed text. Implementation detail of each module of the system design is explained with the used algorithms, software and tools in this chapter.

6.2 Language Identification System for Tamil-English Code-Mixed Text

Language identification system for Tamil-English code-mixed text developed by using Jupiter notebook 5.5.0 with Python 3.6.4. Mainly Pandas, nltk and numpy libraries were used in preprocessing and feature identification process. For the training, testing and model creation process Weka 3.6 software was used. Java jdk 1.8, NetBeans IDE 8.0.2 and weka-stable-3.8.0 library were used to predict the language labels of unseen Tamil-English sentences at word level. The detailed implementation of language identification system for Tamil-English code-mixed text is explained in this section.

6.2.1 Preprocessing

In the preprocessing module raw Tamil-English code-mixed sentences was taken as input. A sample of sentences taken for input shown in Figure 6.1.

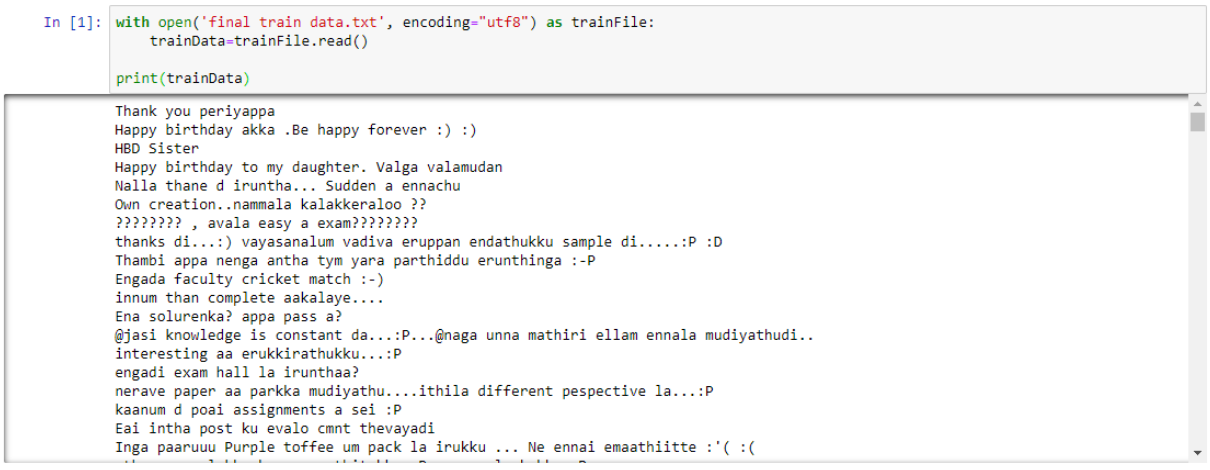


Figure 6.1 Sample of Tamil-English Code-Mixed Data

Tokenizer in NLTK used to tokenize input Tamil-English sentences as words. The lower() function used to convert the words in lowercase. A sample of Tamil-English code-mixed words shown in Figure 6.2.

```
In [2]: import re
        from nltk.tokenize import word_tokenize

        cleanData=re.sub('[.@/]+', ' ', trainData)
        cleanTrainData=cleanData.lower()

        print("\n Words in Training data")
        TrainWordList = word_tokenize(cleanTrainData)
        TrainWordList=[re.sub('\ufeff', '',x) for x in TrainWordList]

        print(TrainWordList)
```

```
'dress', 'namba', 'aalu', 'full', 'white', 'aravinth', 'always', 'kamal', 'sir', 'than', 'maththavanga', 'yaara', 'irundhalu
m', 'pinnalathan', 'annan', 'pulliya', 'anna', 'vazhlthukal', 'padinnarum', 'petru', 'seerum', 'sirappuma', 'peru', 'valvu',
'vazhanum', 'annana', 'kettada', 'solunga', 'fans', 'love', 'you', 'bye', 'coming', 'soooon', 'mano', 'yours', 'faithfully',
'antha', 'raja', ',', 'manishanuku', 'bathila', 'oru', 'koranga', 'thanaku', 'thonaiya', 'vecharam', 'epic', 'story', 'anu',
'chechi', 'ethupati', 'anu', 'radha', 'sriram', 'yen', 'ipadi', 'ayitanga', 'appadi', 'semma', 'pic', 'bhogan', 'sir', 'appap
paaaaa', 'enna', 'azhagu', 'aravindswamy', 'eppadi', 'irunthalum', 'engal', 'favourite', 'than', 'aravind', 'swamy', 'inka',
'kekaaaa', 'ariva', 'erukura', 'angal', 'yellam', 'alaga', 'eruka', 'mattanga', 'arvind', 'sir', ',', 'enaku', 'romba', 'naal
a', 'oru', 'aasai', 'arvind', 'swami', 'fans', 'entu', 'solitu', 'surya', 'da', 'family', 'avlo', 'seenu', 'illa', 'smara',
'than', 'iruku', 'beautiful', 'smile', 'sema', 'azhagu', 'lovely', 'awesome', 'looking', 'superb', 'bro', 'bogan', 'shoot',
'ella', 'use', 'paina', 'costum', 'thana', 'cha', 'evanalam', 'oru', 'manisan', 'nenga', 'ethukku', 'thanda', 'sari', 'chinn
a', 'chinna', 'mazhaithullikal', 'serthu', 'vacheneeaa', 'choppu', 'vilaiyatula', 'use', 'pannuvom', 'ha', 'ha', 'ha', 'hs',
'chubby', 'appadiye', 'killu', 'chubby', 'appadiye', 'kiu', 'cup', 'kulla', 'enna', 'sir', 'irrukum', 'tea', 'ah', 'coffee',
'ah', 'juice', 'ah', 'illa', 'matter', 'ah', '?', 'edhu', 'epdiyum', 'irukuttum', 'en', 'vuyir', ',', 'alagu', 'sellam', ',',
'en', 'manam', 'virumbum', 'anbu', 'aravind', 'samykku', 'munnaala', 'nee', 'vuñna', 'kaiya', 'vachu', 'maraichalum', 'sari',
'maraikkaatiyum', 'sari', ',', 'ellaar', 'kannum', 'aravind', 'maeladhaana', 'irukkum', '!', 'kan', 'cover', 'alagu', '``',
'elloraiyum', 'kollai', 'kollum', 'azhagu', 'smile', 'you', 'are', 'the', 'best', 'looking', 'person', 'enakku', 'indha', 'ma
thiri', 'yellam', 'varadhu', 'enga', 'anna', 'familykeva', 'orukodi', 'like', 'anna', 'kita', 'sollunga', 'thalabathy', '80',
'avarkuda', 'nan', 'serndu', 'nadikkaporennu', 'bye', 'mano', 'en', 'ippideeee', 'enku', 'rmb', 'pdcha', 'mv', 'and', 'ds',
```

Figure 6.2 Sample of Tamil-English Code-Mixed words

It was observe the some words contains wordplay characters. For example in the word “maaaaaaaaaaa” the character ‘a’ appears lot of times. So that word consider as wordplay word. The regular expression used to remove the wordplay characters. The characters appears more than two times were eliminated. So the regular expression used to replace the word “maaaaaaaaaaa” as “maa”. After that, the term frequency of each unique word was taken using count() function. A sample unique word with term frequency shown in Figure 6.3.

```
In [5]: Trainwordfreq = []
        for w in trainWords:
            Trainwordfreq.append(trainWords.count(w))

        def sortFreqDict(freqdict):
            aux = [(freqdict[key],key) for key in freqdict]
            aux.sort()
            aux.reverse()
            return aux

        for a,b in zip(trainWords,Trainwordfreq):
            trainWordFreqDict=dict(zip(trainWords,Trainwordfreq))

        trainWordFreq_sorteddict = sortFreqDict(trainWordFreqDict)
        for s in trainWordFreq_sorteddict: print(str(s))
```

```
(178, 'da')
(138, 'la')
(113, 'oru')
(110, 'enna')
(100, 'nee')
(99, 'than')
(85, 'sir')
(73, 'ah')
(71, 'illa')
(70, 'ku')
(59, 'neenga')
(55, 'enga')
(53, 'poi')
(51, 'tamil')
```

Figure 6.3 Sample of Unique Tamil-English Code-Mixed words with Term Frequency

After that the unique Tamil-English code-mixed word annotated with language tags. There are three language tags, “tam” for Tamil words, and “eng” for English words and “rest” for all other words. The “rest” tag includes Named Entities, Acronyms, Universal, mixed and other language tags. A sample of annotated Tamil-English code-mixed words with language tags shown in Figure 6.4.

	Words	Lang Class
2	da	tam
3	la	tam
4	oru	tam
5	enna	tam
6	nee	tam
7	sir	eng
8	illa	tam
9	ku	tam
10	neenga	tam
11	enga	tam
12	poi	tam
13	tamil	tam
14	dai	tam
15	unga	tam
16	ellam	tam
17	nu	tam
18	ne	tam
19	nalla	tam
20	super	eng
21	ithu	tam
22	intha	tam
23	'	rest

Figure 6.4 Sample of Annotated Tamil-English Code-Mixed Words with Language Tags

6.2.2 Feature identification of Tamil-English code-mixed text

The annotated Tamil-English code-mixed word with languages tags was taken as the input for feature identification module. In this process the features mentioned in section 5.2.3 was identified for each words. The features are Tamil Unicode characters in Roman scripts, language specific dictionaries, double consonant, and term frequency. A sample Tamil word with features shown in Figure 6.5

```
{'Words': 'enna', 'Lang Class': 'tam', 'Term Freq': 110.0, 'a': '', 'aa': '', 'ae': '', 'ai': '', 'bb': '', 'cc': '', 'dd': '',
'e': '', 'ee': '', 'ff': '', 'gg': '', 'gn': '', 'gna': '', 'gne': '', 'gni': '', 'gnu': '', 'h': '', 'ha': '', 'haa': '', 'ha
e': '', 'hai': '', 'he': '', 'hi': '', 'hii': '', 'ho': '', 'how': '', 'hu': '', 'huu': '', 'i': '', 'ii': '', 'j': '', 'ja':
'', 'jaa': '', 'jai': '', 'je': '', 'ji': '', 'jo': '', 'ju': '', 'k': '', 'ka': '', 'kaa': '', 'kai': '', 'ke': '', 'kea': '',
'ki': '', 'kk': '', 'ko': '', 'ksh': '', 'ksha': '', 'ku': '', 'kuu': '', 'l': '', 'la': '', 'laa': '', 'lae': '', 'lai': '',
'le': '', 'li': '', 'll': '', 'lo': '', 'loa': '', 'low': '', 'lu': '', 'm': '', 'ma': '', 'maa': '', 'mae': '', 'mai': '', 'm
e': '', 'mi': '', 'mm': '', 'mo': '', 'mu': '', 'n': '', 'na': '', 'naa': '', 'nae': '', 'nai': '', 'ne': '', 'ng': '', 'nga':
'', 'ngaa': '', 'ngai': '', 'nge': '', 'ngi': '', 'ngo': '', 'ngu': '', 'nh': '', 'nha': '', 'ni': '', 'nii': '', 'nn': '', 'n
o': '', 'now': '', 'nu': '', 'nuu': '', 'o': '', 'oa': '', 'oo': '', 'ow': '', 'p': '', 'pa': '', 'paa': '', 'pae': '', 'pai':
'', 'pe': '', 'pi': '', 'po': '', 'poa': '', 'pow': '', 'pp': '', 'pu': '', 'puu': '', 'q': '', 'r': '', 'ra': '', 'raa': '',
'rae': '', 'rai': '', 're': '', 'ri': '', 'rii': '', 'ro': '', 'roa': '', 'row': '', 'rr': '', 'ru': '', 'ruu': '', 's': '', 's
a': '', 'saa': '', 'sai': '', 'se': '', 'sh': '', 'sha': '', 'shaa': '', 'she': '', 'shi': '', 'sho': '', 'shu': '', 'si': '',
'so': '', 'sri': '', 'ss': '', 'su': '', 'suu': '', 't': '', 'ta': '', 'taa': '', 'tae': '', 'tai': '', 'te': '', 'th': '', 'th
a': '', 'thaa': '', 'thae': '', 'thai': '', 'the': '', 'thi': '', 'thii': '', 'tho': '', 'thu': '', 'thuu': '', 'ti': '', 'to':
'', 'tow': '', 'tt': '', 'tu': '', 'tuu': '', 'u': '', 'uu': '', 'v': '', 'va': '', 'vaa': '', 'vae': '', 'vai': '', 've': '',
'vi': '', 'vii': '', 'vo': '', 'voa': '', 'vu': '', 'vuu': '', 'y': '', 'ya': '', 'yaa': '', 'yae': '', 'yai': '', 'ye': '', 'y
i': '', 'yii': '', 'yo': '', 'yu': '', 'yy': '', 'z': '', 'za': '', 'zai': '', 'ze': '', 'zi': '', 'zu': '', 'BNC_Corpus': '',
'LexNorm_Corpus': ''}
```


Figure 6.5 Sample Tamil word with features

In the feature identification process of Tamil-English code-mixed text the below features extracted from the words.

- Tamil Unicode characters in Roman scripts: The presence of 261 features in the words were identified. This is a Boolean feature. Presence of each feature recorded as 1 otherwise 0.

- Language-specific dictionaries: Use to identify the presence of a word in dictionaries. This is a Boolean feature. Presence of word will be 1 otherwise 0. The presence of words in BNC and LEXNORM dictionaries was identified.

- Double consonant: The presence of 16 double consonant features in the words was identified. This is a Boolean feature. Presence of each feature recorded as 1 otherwise 0.

A sample of Tamil word after the feature identification process shown in Figure 6.6.


```
{'Words': 'nalla', 'Lang Class': 'tam', 'Term Freq': 43.0, 'a': 1, 'aa': 0, 'ae': 0, 'ai': 0, 'bb': 0, 'cc': 0, 'dd': 0, 'e':
0, 'ee': 0, 'ff': 0, 'gg': 0, 'gn': 0, 'gna': 0, 'gne': 0, 'gni': 0, 'gnu': 0, 'h': 0, 'ha': 0, 'haa': 0, 'hae': 0, 'hai': 0,
'he': 0, 'hi': 0, 'hii': 0, 'ho': 0, 'how': 0, 'hu': 0, 'huu': 0, 'i': 0, 'ii': 0, 'j': 0, 'ja': 0, 'jaa': 0, 'jai': 0, 'je':
0, 'ji': 0, 'jo': 0, 'ju': 0, 'k': 0, 'ka': 0, 'kaa': 0, 'kai': 0, 'ke': 0, 'kea': 0, 'ki': 0, 'kk': 0, 'ko': 0, 'ksh': 0, 'ksh
a': 0, 'ku': 0, 'kuu': 0, 'l': 1, 'la': 1, 'laa': 0, 'lae': 0, 'lai': 0, 'le': 0, 'li': 0, 'll': 1, 'lo': 0, 'loa': 0, 'low':
0, 'lu': 0, 'm': 0, 'ma': 0, 'maa': 0, 'mae': 0, 'mai': 0, 'me': 0, 'mi': 0, 'mm': 0, 'mo': 0, 'mu': 0, 'n': 1, 'na': 1, 'naa':
0, 'nae': 0, 'nai': 0, 'ne': 0, 'ng': 0, 'nga': 0, 'ngaa': 0, 'ngai': 0, 'nge': 0, 'ngi': 0, 'ngo': 0, 'ngu': 0, 'nh': 0, 'nh
a': 0, 'ni': 0, 'nii': 0, 'nn': 0, 'no': 0, 'now': 0, 'nu': 0, 'nuu': 0, 'o': 0, 'oa': 0, 'oo': 0, 'ow': 0, 'p': 0, 'pa': 0, 'p
aa': 0, 'pae': 0, 'pai': 0, 'pe': 0, 'pi': 0, 'po': 0, 'poa': 0, 'pow': 0, 'pp': 0, 'pu': 0, 'puu': 0, 'q': 0, 'r': 0, 'ra': 0,
'raa': 0, 'rae': 0, 'rai': 0, 're': 0, 'ri': 0, 'rii': 0, 'ro': 0, 'roa': 0, 'row': 0, 'rr': 0, 'ru': 0, 'ruu': 0, 's': 0, 's
a': 0, 'saa': 0, 'sai': 0, 'se': 0, 'sh': 0, 'sha': 0, 'shaa': 0, 'she': 0, 'shi': 0, 'sho': 0, 'shu': 0, 'si': 0, 'so': 0, 'sr
i': 0, 'ss': 0, 'su': 0, 'suu': 0, 't': 0, 'ta': 0, 'taa': 0, 'tae': 0, 'tai': 0, 'te': 0, 'th': 0, 'tha': 0, 'thaa': 0, 'tha
e': 0, 'thai': 0, 'the': 0, 'thi': 0, 'thii': 0, 'tho': 0, 'thu': 0, 'thuu': 0, 'ti': 0, 'to': 0, 'tow': 0, 'tt': 0, 'tu': 0,
'tuu': 0, 'u': 0, 'uu': 0, 'v': 0, 'va': 0, 'vaa': 0, 'vae': 0, 'vai': 0, 've': 0, 'vi': 0, 'vii': 0, 'vo': 0, 'voa': 0, 'vu':
0, 'vuu': 0, 'y': 0, 'ya': 0, 'yaa': 0, 'yae': 0, 'yai': 0, 'ye': 0, 'yi': 0, 'yii': 0, 'yo': 0, 'yu': 0, 'yy': 0, 'z': 0, 'z
a': 0, 'zai': 0, 'ze': 0, 'zi': 0, 'zu': 0, 'BNC_Corpus': 0, 'LexNorm_Corpus': 0}
```


Figure 6.6 Sample Tamil word with identified features with values

Among this embedded features there were some Tamil Unicode characters in Roman scripts features had the mean value as zero. The below 77 features mean value identified as zero. These features did not play a significant role in algorithm. So these features were removed from the training dataset of Tamil-English code-mixed text.

['gnaa','gnae','gnai','gnii','gno','gnoa','gnow','gnuu','hoa','jae','jii','joa','jow','juu','kii','koa','kow','kshaa','kshae','kshai','kshe','kshi','kshii','ksho','kshoa','kshow','kshu','kshuu','lii','luu','mii','moa','mow','muu','ngae','ngii','ngoa','ngow','nguu','nhaa','nhae','nhai','nhe','nhi','nhii','nho','nhoa','nhow','nhu','nhuu','noa','pii','sae','shae','shai','shii','shoa','show','shuu','sii','soa','sow','thoa','thow','tii','toa','vow','yoa','yow','yuu','zaa','zae','zii','zo','zoa','zow','zuu'].

So now the training dataset consists of 203 features. In addition of corresponding language tags totally 204 attributes included in training data of Tamil-English code-mixed text.

6.2.3 Model Development for Tamil-English code-mixed text

For the model development process, the Weka 3.9.2 tool was used. Different machine learning classifiers such as Support Vector Machine, Naïve Bayes, Logistic Regression, Random Forest and Decision Tree used to create the models in order to evaluate the model performance. The feature-based model created based on identified features and languages labels as input. The dataset subjected to 10 fold cross-validation in order to evaluate the predictive models. The overall accuracy and F-Measure of different classifiers were recorded to identify the most suitable model for perdition process. The parameter used in model development process for different classifiers shown in Table 6.1.

Table 6.1 The parameter used in model development by different classifiers for Tamil-English code-mixed text

Classifiers	Parameters Used
Support Vector Machine	Linear Kernel, C=1
Logistics Regression	Ridge = 1.0E-8
Decision Tree	Confident Factor=0.25, Number of objects=2
Naive Bayes	Use Kernel Estimator=False
Random Forest	Number of iteration = 10

6.3 Language Identification System for Sinhala-English Code-Mixed Text

Language identification system for Sinhala-English code-mixed text developed by using Jupiter notebook 5.5.0 with Python 3.6.4. Mainly Pandas, nltk and numpy libraries were used in preprocessing and feature identification process. For the training, testing and model creation process Weka 3.6 software was used. Java jdk 1.8, NetBeans IDE 8.0.2 and weka-stable-3.8.0 library were used to predict the language labels of unseen Sinhala-English sentences at word level. The detailed implementation of language identification system for Sinhala-English code-mixed text is explained in this section.

6.3.1 Preprocessing

In the preprocessing module raw Sinhala-English code-mixed sentences was taken as input. A sample of sentences taken for input shown in Figure 6.7.

```
In [1]: with open('Sinhala-English code mixed text.txt', encoding="utf-8") as trainFile:
            trainData=trainFile.read()

        print(trainData)
```

```
Meka wath ehenam damu e man kiyanna
Ehakota lakshan inne kohomada
Den meke man ko ?
Honda prashnayak akke..😊 blouse eka thama tag kare
e phone 1ke prashnayak.screen 1ke kawruth penniwa.nathnam kawruhari crop karala ..
Onna mama nm newei
screen eke pena okkoma photo ekata wadinawa
Metchara mahathata indalath math nane e 4to eke...
ayyage phone 1ka ..
balanna api dennata karana wenaskam. Api dennawama photo eken kapala
eka galen kurullo dennekma maranawa kiyanne mehema scene walata wenna athi
Nilupul aiyage ara gorilla glass effect ekenda mokadda ekenda danne na api dennawa kepune...
ahaka inna rilaw , gorillo allanna epa okata , gaanata cut karala tiyenne
Ow man hitanne e gorilla glass eken gorillo tika witarak filter karala aranda koheda.
Anna hari...eka thamai wela thiyenne
Wow....good job putha.
kolla goda yamin sitina bawa penawa,
Oya colombo gihin karana ewa anith aya karane na bro
Mana ube kathawa neh oke tiyenne
```

Figure 6.7 Sample of Sinhala-English Code-Mixed Data

Tokenizer in NLTK used to tokenize input Sinhala-English sentences as words. The lower() function used to convert the words in lowercase. A sample of Sinhala-English code-mixed words shown in Figure 6.8.

```
In [2]: import re
        from nltk.tokenize import word_tokenize

        cleanData=re.sub('[.@/]+', ' ', trainData)
        cleanTrainData=cleanData.lower()

        print("\nWords in Training data")
        TrainWordList = word_tokenize(cleanTrainData)
        TrainWordList=[re.sub('\ufeff', '',x) for x in TrainWordList]

        print(TrainWordList)
```

```
Words in Training data
['meka', 'wath', 'ehenam', 'damu', 'e', 'man', 'kiyanna', 'ehakota', 'lakshan', 'inne', 'kohomada', 'den', 'meke', 'man', 'k
o', '?', 'honda', 'prashnayak', 'akke', '😊', 'blouse', 'eka', 'thama', 'tag', 'kare', 'e', 'phone', '1ke', 'prashnayak', 'sc
reen', '1ke', 'kawruth', 'penniwa', 'nathnam', 'kawruhari', 'crop', 'karala', 'onna', 'mama', 'nm', 'newei', 'screen', 'eke',
'pena', 'okkoma', 'photo', 'ekata', 'wadinawa', 'metchara', 'mahathata', 'indalath', 'math', 'nane', 'e', '4to', 'eke', 'ayya
ge', 'phone', '1ka', 'balanna', 'api', 'dennata', 'karana', 'wenaskam', 'api', 'dennawama', 'photo', 'eken', 'kapala', 'eka',
'galen', 'kurullo', 'dennekma', 'maranawa', 'kiyanne', 'mehema', 'scene', 'walata', 'wenna', 'athi', 'nilupul', 'aiyage', 'ar
a', 'gorilla', 'glass', 'effect', 'ekenda', 'mokadda', 'ekenda', 'danne', 'na', 'api', 'dennawa', 'kepune', 'ahaka', 'inna',
'rilaw', ',', 'gorillo', 'allanna', 'epa', 'okata', ',', 'gaanata', 'cut', 'karala', 'tiyenne', 'ow', 'man', 'hitanne', 'e',
'gorilla', 'glass', 'eken', 'gorillo', 'tika', 'witarak', 'filter', 'karala', 'aranda', 'koheda', 'anna', 'hari', 'eka', 'tha
mai', 'wela', 'thiyenne', 'wow', 'good', 'job', 'putha', 'kolla', 'goda', 'yamin', 'sitina', 'bawa', 'penawa', ',', 'oya', 'c
olombo', 'gihin', 'karana', 'ewa', 'anith', 'aya', 'karane', 'na', 'bro', 'mana', 'ube', 'kathawa', 'neh', 'oke', 'tiyenne',
'matath', 'tama', 'matakai', 'uncle', 'issara', 'oyawa', 'iskoleta', 'ekkaragena', 'ena', 'widiya', 'vas', 'mama', 'oyage',
'wedding', 'eke', 'photos', 'balala', 'mata', 'ae', 'kale', 'matak', 'una', 'vas', 'ara', 'balance', 'karana', 'ekatanam', 'j
ehan', 'one', 'na', 'man', 'hitanne', 'meka', 'liyanna', 'hethuva', 'mokakda', 'danne', 'ne', 'neda', 'athi', 'yantham', 'nal
```

Figure 6.8 Sample of Sinhala-English Code-Mixed words

It was observe the some words contains wordplay characters. For example in the word “chooooooty” the character ‘o’ appears lot of times. So that word consider as wordplay word. The regular expression used to remove the wordplay characters. The characters appears more than two times were eliminated. So the regular expression used to replace the word “chooooooty” as “chooty”. After that, the term frequency of each unique word was taken using count() function. A sample unique Sinhala-English words with term frequency shown in Figure 6.9.

```
In [4]: Trainwordfreq = []
        for w in trainWords:
            Trainwordfreq.append(trainWords.count(w))

        def sortFreqDict(freqdict):
            aux = [(freqdict[key],key) for key in freqdict]
            aux.sort()
            aux.reverse()
            return aux

        for a,b in zip(trainWords,Trainwordfreq):
            trainWordFreqDict=dict(zip(trainWords,Trainwordfreq))

        trainWordFreq_sorteddict = sortFreqDict(trainWordFreqDict)
        for s in trainWordFreq_sorteddict: print(str(s))
```

```
(23, 'set')
(21, 'yako')
(21, 'photo')
(21, 'inne')
(21, 'hari')
(21, 'ai')
(20, 'wela')
(20, 'thama')
(20, 'neh')
(20, 'meka')
(20, 'mata')
(20, 'ekata')
```

Figure 6.9 Sample of Unique Sinhala-English Code-Mixed words with Term Frequency

After that the unique Sinhala-English code-mixed word annotated with language tags. There are three language tags, “sin” for Sinhala words, and “eng” for English words and “rest” for all other words. The “rest” tag includes Named Entities, Acronyms, Universal, mixed and other language tags. A sample of annotated Sinhala-English code-mixed words with language tags shown in Figure 6.10.

api	sin
ekak	sin
oya	sin
eke	sin
dan	sin
uba	sin
ow	sin
wage	sin
epa	sin
mama	sin
kiyala	sin
set	eng
yako	sin
photo	eng
inne	sin
hari	sin
ai	sin
wela	sin
thama	sin
meka	sin
mata	sin
ekata	sin

Figure 6.10 Sample of Annotated Sinhala-English Code-Mixed Words with Language Tags

6.3.2 Feature identification of Sinhala-English code-mixed text

The annotated Sinhala-English code-mixed word with languages tags was taken as the input for feature identification module. In this process the features mentioned in section 5.3.3 was identified for each words. The features are Sinhala Unicode characters in Roman scripts, language specific dictionaries, and term frequency. A sample of a Sinhala word with features shown in Figure 6.11.

{'Words': 'kiyala', 'Lang Class': 'sin', 'TermFreq': 24.0, 'a': '', 'aa': '', 'ae': '', 'au': '', 'b': '', 'ba': '', 'baa': '', 'bae': '', 'bau': '', 'be': '', 'bea': '', 'bi': '', 'bo': '', 'bu': '', 'buu': '', 'ch': '', 'cha': '', 'chaa': '', 'che': '', 'chi': '', 'cho': '', 'chu': '', 'd': '', 'da': '', 'daa': '', 'dau': '', 'de': '', 'dea': '', 'dh': '', 'dha': '', 'dhe': '', 'dhi': '', 'dhu': '', 'di': '', 'dii': '', 'do': '', 'du': '', 'e': '', 'ea': '', 'g': '', 'ga': '', 'gaa': '', 'gau': '', 'ge': '', 'gi': '', 'go': '', 'gu': '', 'h': '', 'ha': '', 'haa': '', 'he': '', 'hea': '', 'hi': '', 'hii': '', 'ho': '', 'hu': '', 'i': '', 'ii': '', 'j': '', 'ja': '', 'jaa': '', 'je': '', 'ji': '', 'jo': '', 'ju': '', 'k': '', 'ka': '', 'kaa': '', 'kau': '', 'ke': '', 'ki': '', 'kii': '', 'ko': '', 'ku': '', 'kuu': '', 'l': '', 'la': '', 'laa': '', 'le': '', 'lea': '', 'li': '', 'lo': '', 'lu': '', 'm': '', 'ma': '', 'maa': '', 'mae': '', 'me': '', 'mi': '', 'mo': '', 'mu': '', 'muu': '', 'n': '', 'na': '', 'naa': '', 'ne': '', 'nea': '', 'ni': '', 'no': '', 'nu': '', 'o': '', 'oe': '', 'r': '', 'ra': '', 'raa': '', 'rae': '', 'rau': '', 're': '', 'rea': '', 'ri': '', 'rii': '', 'ro': '', 'ru': '', 's': '', 'sa': '', 'saa': '', 'sae': '', 'sau': '', 'se': '', 'sea': '', 'si': '', 'sii': '', 'so': '', 'su': '', 't': '', 'ta': '', 'taa': '', 'te': '', 'tea': '', 'th': '', 'tha': '', 'thaa': '', 'the': '', 'thi': '', 'thii': '', 'tho': '', 'thu': '', 'ti': '', 'to': '', 'tu': '', 'u': '', 'uu': '', 'w': '', 'wa': '', 'waa': '', 'we': '', 'wi': '', 'wo': '', 'wu': '', 'y': '', 'ya': '', 'yaa': '', 'yau': '', 'ye': '', 'yi': '', 'yo': '', 'yu': '', 'BNC_Corpus': '', 'LexNorm_Corpus': ''}

Figure 6.11 Sample of Sinhala word with features

In the feature identification process of Sinhala-English code-mixed text the below features extracted from the words.

- Sinhala Unicode characters in Roman scripts: The presence of 233 features in the words were identified. This is a Boolean feature. Presence of each feature recorded as 1 otherwise 0.

- English language-specific dictionaries: Use to identify the presence of a word in dictionaries. This is a Boolean feature. Presence of word will be 1 otherwise 0. The presence of words in BNC and LEXNORM dictionaries was identified.

A sample of Sinhala word after the feature identification process shown in Figure 6.12.

{'Words': 'kiyala', 'Lang Class': 'sin', 'TermFreq': 24.0, 'a': 1, 'aa': 0, 'ae': 0, 'au': 0, 'b': 0, 'ba': 0, 'baa': 0, 'bae': 0, 'bau': 0, 'be': 0, 'bea': 0, 'bi': 0, 'bo': 0, 'bu': 0, 'buu': 0, 'ch': 0, 'cha': 0, 'chaa': 0, 'che': 0, 'chi': 0, 'cho': 0, 'chu': 0, 'd': 0, 'da': 0, 'daa': 0, 'dau': 0, 'de': 0, 'dea': 0, 'dh': 0, 'dha': 0, 'dhe': 0, 'dhi': 0, 'dhu': 0, 'di': 0, 'dii': 0, 'do': 0, 'du': 0, 'e': 0, 'ea': 0, 'g': 0, 'ga': 0, 'gaa': 0, 'gau': 0, 'ge': 0, 'gi': 0, 'go': 0, 'gu': 0, 'h': 0, 'ha': 0, 'haa': 0, 'he': 0, 'hea': 0, 'hi': 0, 'hii': 0, 'ho': 0, 'hu': 0, 'i': 1, 'ii': 0, 'j': 0, 'ja': 0, 'jaa': 0, 'je': 0, 'ji': 0, 'jo': 0, 'ju': 0, 'k': 1, 'ka': 0, 'kaa': 0, 'kau': 0, 'ke': 0, 'ki': 1, 'kii': 0, 'ko': 0, 'ku': 0, 'kuu': 0, 'l': 1, 'la': 1, 'laa': 0, 'le': 0, 'lea': 0, 'li': 0, 'lo': 0, 'lu': 0, 'm': 0, 'ma': 0, 'maa': 0, 'mae': 0, 'me': 0, 'mi': 0, 'mo': 0, 'mu': 0, 'muu': 0, 'n': 0, 'na': 0, 'naa': 0, 'ne': 0, 'nea': 0, 'ni': 0, 'no': 0, 'nu': 0, 'o': 0, 'oe': 0, 'r': 0, 'ra': 0, 'raa': 0, 'rae': 0, 'rau': 0, 're': 0, 'rea': 0, 'ri': 0, 'rii': 0, 'ro': 0, 'ru': 0, 's': 0, 'sa': 0, 'saa': 0, 'sae': 0, 'sau': 0, 'se': 0, 'sea': 0, 'si': 0, 'sii': 0, 'so': 0, 'su': 0, 't': 0, 'ta': 0, 'taa': 0, 'te': 0, 'tea': 0, 'th': 0, 'tha': 0, 'thaa': 0, 'the': 0, 'thi': 0, 'thii': 0, 'tho': 0, 'thu': 0, 'ti': 0, 'to': 0, 'tu': 0, 'u': 0, 'uu': 0, 'w': 0, 'wa': 0, 'waa': 0, 'we': 0, 'wi': 0, 'wo': 0, 'wu': 0, 'y': 1, 'ya': 1, 'yaa': 0, 'yau': 0, 'ye': 0, 'yi': 0, 'yo': 0, 'yu': 0, 'BNC_Corpus': 0, 'LexNorm_Corpus': 0}

Figure 6.12 Sample of Sinhala word with identified features with values

Among this embedded features there were some Sinhala Unicode characters in Roman scripts identified features had the mean value as zero. The below 76 feature's mean value identified as zero. These features did not play a significant role in algorithm. So these features were removed from the Sinhala-English training dataset.

[bii, boe, chae, chau, chea, chii, choe, chuu, dae, dhaa, dhae, dhau, dhea, dhii, dho, dhoe, dhuu, doe, duu, gae, gea, gii, goe, guu, hae, hau, hoe, huu, jae, jau, jea, jii, joe, juu, kae, kea, koe, lae,

lau, lii, loe, luu, mau, mea, mii, moe, nae, nau, nii, noe, nuu, roe, ruu, soe, suu, tae, tau, thae, thau, thea, thoe, thuu, tii, toe, tuu, wae, wau, wea, wii, woe, wuu, yae, yea, yii, yoe, yuu]

So now the training dataset consists of 160 features. In addition of corresponding language tags totally 161 attributes included in training data of Sinhala-English code-mixed text.

6.3.3 Model Development for Sinhala-English code-mixed text

For the model development process, the Weka 3.9.2 tool was used. Different machine learning classifiers such as Support Vector Machine, Naïve Bayes, Logistic Regression, Random Forest and Decision Tree used to create the models in order to evaluate the model performance. The feature-based model created based on identified features and languages labels as input. The dataset subjected to 10 fold cross-validation in order to evaluate the predictive models. The overall accuracy and F-Measure of different classifiers were recorded to identify the most suitable model for perdition process. The parameter used in model development process for different classifiers shown in Table 6.2.

Table 6.2 The parameter used in model development by different classifiers for Sinhala-English code-mixed text

Classifiers	Parameters Used
Support Vector Machine	Linear Kernel, C=1
Logistics Regression	Ridge = 1.0E-8
Decision Tree	Confident Factor=0.25, Number of objects=2
Naive Bayes	Use Kernel Estimator=True
Random Forest	Number of iteration = 100

6.4 Summary

In this chapter the implementation of the word level language identification system for Tamil-English and Sinhala-English code-mixed text is discussed in detail. In the next chapter, evaluation of the proposed solution is given with experimental results.

Chapter 7

Evaluation

7.1 Introduction

Evaluation strategy, experimental design and results are discussed in this chapter. Experiment results of model development for Tamil-English code-mixed text and Sinhala-English code-mixed text is discussed in this chapter. Also sample based evaluation results of predictive models for Tamil-English code-mixed text and Sinhala-English code-mixed text is discussed in this chapter.

7.2 Experimental design

Experiments designed to evaluate the performance of the models developed based on Tamil-English feature set and Sinhala-English feature set. Also another experiment designed to evaluate the model prediction for test data based on sample based analysis.

7.2.1 Experimental design for Model Evaluation

For the model evaluation process, the Weka 3.9.2 tool was used. Different machine learning classifiers such as Support Vector Machine, Naïve Bayes, Logistic Regression, Random Forest and Decision Tree used to evaluate the performance of models developed based on Tamil-English code-mixed text and Sinhala-English code-mixed text. The knowledge flow of experiment process for model evaluation shown in Figure 7.1.

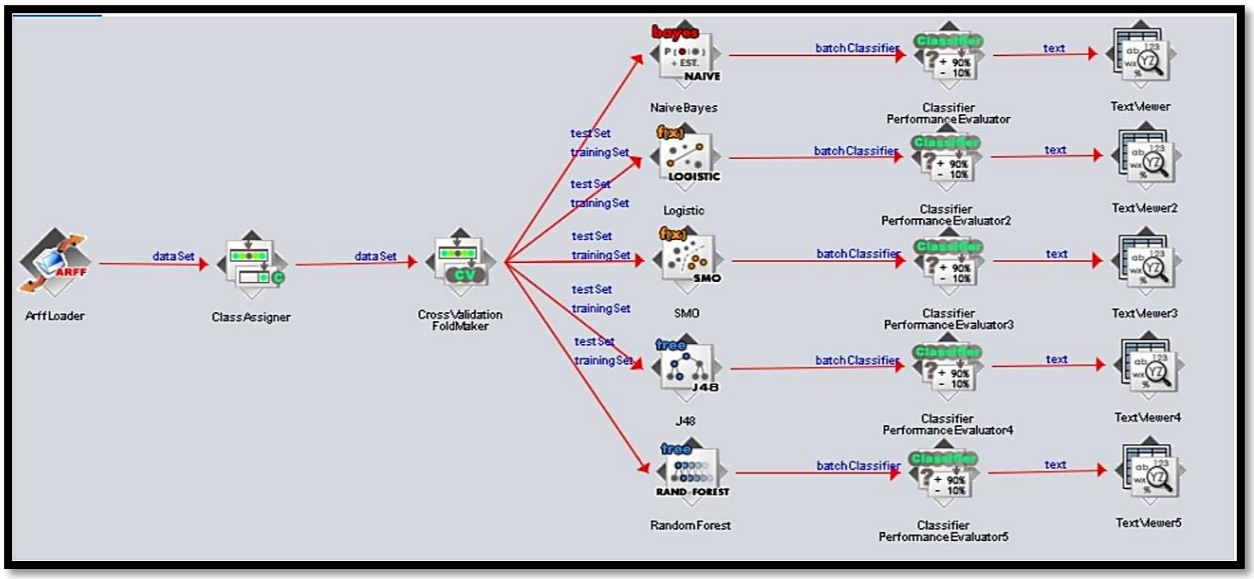


Figure 7.1 The knowledge flow of experiment process for model evaluation

7.2.1.1 Evaluation Strategy for predictive Models

Models were tested using 10 fold cross-validation. Cross-validation is a technique to evaluate predictive models by partitioning the original sample into a training set to train the model, and a test set to evaluate the model.

In k-fold cross-validation, the original sample is randomly partitioned into k number of subsamples. Of the k subsamples, one subsample is retained as the validation data for testing the model, and the remaining k-1 subsamples are used for training. The cross validation process is then repeated k times (the folds), with each of the k subsamples used exactly once as the validation data. The k results from the folds is then averaged (or otherwise combined) to produce a single answer. The advantage of this method is that all observations are used for both training and validation, and each observation is used for validation exactly once.

When comparing the models and evaluating performance of them with each other, different measures like Accuracy (%), and F-Measure were considered in the evaluation process.

Accuracy is how close a measured value is to the actual (true) value. Accuracy retrieves the percentage of correctly classified instances.

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$$

Precision is a value of the accuracy provided by a unique class that was predicted.

$$Precision = \frac{TP}{TP + FP}$$

Recall is a measure of the ability of a prediction model to select instances of a certain class from a data set. It is c also called sensitivity, and points to the true positive rate.

$$Recall = \frac{TP}{TP + FN}$$

Where, TP = True Positive
TN = True Negative
FP = False Positive
FN = False Negative

The F score, also called the F1 score or F measure, is a measure of a test's accuracy. The F score is defined as the weighted harmonic mean of the test's precision and recall. This score is calculated with the precision and recall of a test taken into account. Precision, also called the positive predictive value, is the proportion of positive results that truly are positive. Recall, also called sensitivity, is the ability of a test to correctly identify positive results to get the true positive rate. The F score reaches the best value, meaning perfect precision and recall, at a value of 1. The worst F score, which means lowest precision and lowest recall, would be a value of 0.

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

7.2.2 Experimental design for Testing of Models

A system is developed to test the finalized Tamil-English code-mixed model (Tanglish model with SVM) and Sinhala-English code-mixed model (Singlish model with Random Forest). The User Interface of the testing of finalized model shown in Figure 7.2.

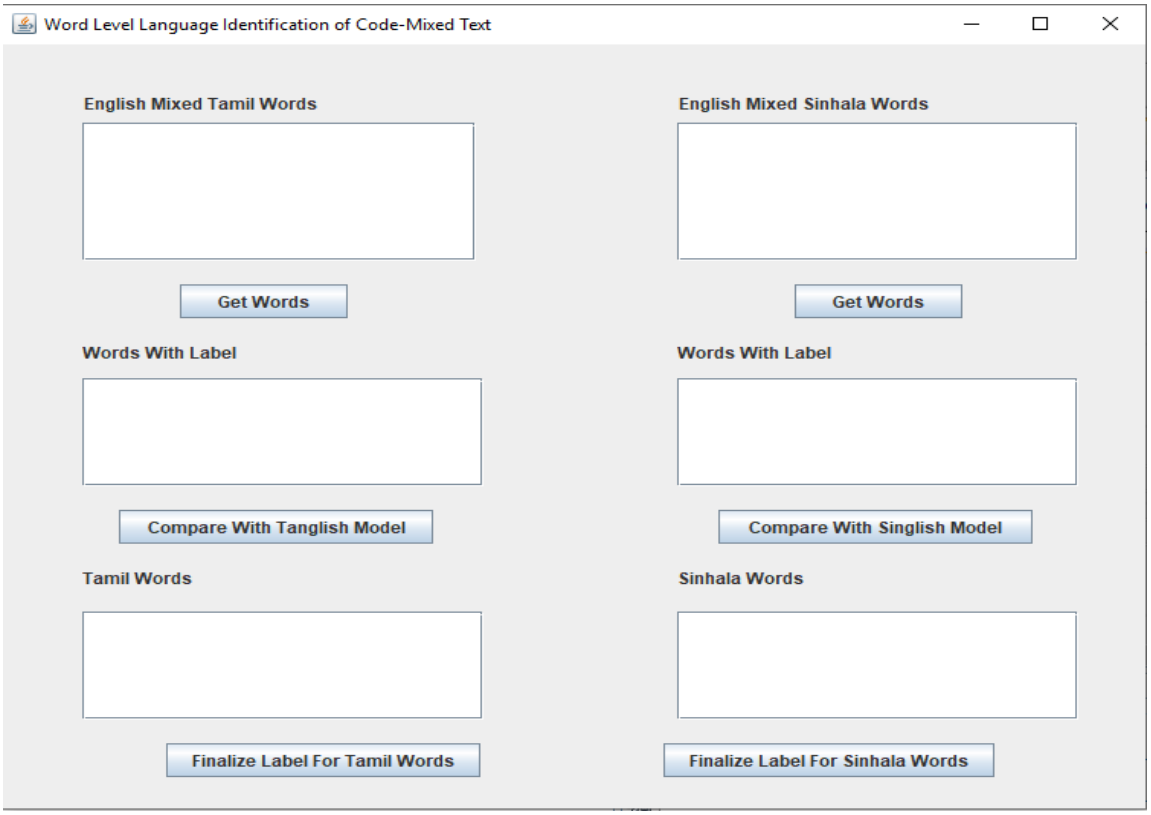


Figure 7.2 The User Interface for testing finalized model for language identification of code-mixed text

In this testing process identified word from a Tamil-English code-mixed sentence was taken as input for testing the Tanglish model (SVM), likewise Sinhala-English code-mixed sentence was taken as input for testing the Singlish model (Random Forest). After that for each word of Tamil-English code-mixed sentence the features such as Tamil Unicode characters in Roman scripts, language specific dictionaries, double consonant, and term frequency was identified and that features compared with Tanglish model and for each word the language labels was predicted using the Tanglish model. Likewise for each word of Sinhala-English code-mixed sentence the features such as Sinhala Unicode characters in Roman scripts, language specific dictionaries, and term frequency was identified and that features compared with Singlish model and for each word the language labels was predicted using the Singlish model. The statistics of the dataset used for testing of models shown in Table 7.1.

Table 7.1 The statistics of the dataset used for reevaluation of models

Test Data	Tamil-English code-mixed data	Sinhala-English code-mixed data
Sentences	50	50
Tokens	312	273
Words	261	216

7.2.2.1 Evaluation Strategy for Testing Models

In the testing process 50 sentences from Tamil-English code-mixed text and 50 sentences from Sinhala-English code-mixed sentences were taken. The sentences were annotated using language labels by human annotators at word level. After that Tanglish words feature matrix and Singlish words feature matrix compared with Tanglish model (SVM) and Singlish model (Random Forest) respectively. The accuracy and the F-measure for each tags were recorded.

7.3 Experimental Results

7.3.1 Experiment Results for Model Evaluation

The model evaluation results obtained from different machine learning classifiers such as Support Vector Machine, Naïve Bayes, Logistic Regression, Random Forest and Decision Tree by using 10 fold cross validation for Tamil-English code-mixed text and Sinhala-English code-mixed text is presented in this section.

The confusion matrix, overall accuracy and F-Measures of ‘tam’, ‘eng’, and ‘rest’ tags obtained from different classifiers for Tamil-English code-mixed text shown in Table 7.2.

Table 7.2 Overall results obtained from different classifiers for Tamil-English code-mixed text

Classifiers	Confusion Matrix				Accuracy	F-Measure (a=tam)	F-Measure (b=eng)	F-Measure (c=rest)
Support Vector Machine (Linear Kernel, C=1)		a	b	c	89.46%	0.947	0.806	0.240
	a	6705	82	32				
	b	193	895	51				
	c	443	106	100				
Logistics Regression (Ridge = 1.0E-8)	a	6650	93	76	89.09%	0.947	0.794	0.273
	b	175	892	72				
	c	400	123	126				
Decision Tree (Confident Factor=0.25, Num of objects=2)	a	6624	132	63	88.89%	0.945	0.752	0.396
	b	231	833	75				
	c	344	111	194				
Naive Bayes (Use Kernel Estimator=False)	a	5974	282	563	84.61%	0.910	0.721	0.304
	b	97	892	150				
	c	245	160	244				
Random Forest (Number of iteration = 10)	a	6781	34	4	86.06%	0.922	0.646	0.127
	b	548	581	10				
	c	559	45	45				

Among this results for Tamil-English code-mixed text SVM with a linear kernel gave 89.46% accuracy for language identification system for Tamil-English code-mixed text at the word level. This model is good for identifying Tamil and English language tags. Because the F-Measure for ‘tam’ and ‘eng’ tags were 0.947 and 0.806 respectively. So this SVM model was taken as finalized model as Tanglish data for testing process.

The confusion matrix, overall accuracy and F-Measures of ‘sin’, ‘eng’, and ‘rest’ tags obtained from different classifiers for Sinhala-English code-mixed text shown in Table 7.3.

Table 7.3 Overall results obtained from different classifiers for Sinhala-English code-mixed text

Classifiers	Confusion Matrix				Accuracy	F-Measure (a=sin)	F-Measure (b=eng)	F-Measure (c=rest)
Support Vector Machine (Linear Kernel, C=1)		a	b	c	88.92%	0.942	0.749	0.174
	a	2274	44	6				
	b	106	283	8				
	c	125	32	18				
Logistics Regression (Ridge = 1.0E-8)	a	2241	55	28	88.33%	0.943	0.727	0.273
	b	86	280	31				
	c	100	38	37				
Decision Tree (Confident Factor=0.25, Num of objects=2)	a	2250	50	24	89.30%	0.943	0.731	0.474
	b	109	268	20				
	c	89	18	68				
Naive Bayes (Use Kernel Estimator=True)	a	2185	105	34	86.81%	0.936	0.715	0.087
	b	66	319	12				
	c	94	71	10				
Random Forest (Number of iteration = 100)	a	2280	24	20	90.50%	0.949	0.758	0.513
	b	118	271	8				
	c	82	23	70				

Among this results for Sinhala-English code-mixed text, Random forest classifier gave 90.5% accuracy for language identification system for Sinhala-English code-mixed text at the word level. This model is good for identifying Sinhala and English language tags. Because the F-Measure for 'sin' and 'eng' tags were 0.949 and 0.758 respectively. So this Random forest model was taken as finalized model as Singlish data for testing process.

In order to identify the important features from feature set, feature evaluation was done with different classifiers by adding and removing the features. The accuracy obtained from different classifiers for different feature set was recorded. The features evaluation chart for Tamil-English code-mixed text and Sinhala-English code-mixed text illustrated in Figure 7.3 and Figure 7.4 respectively.

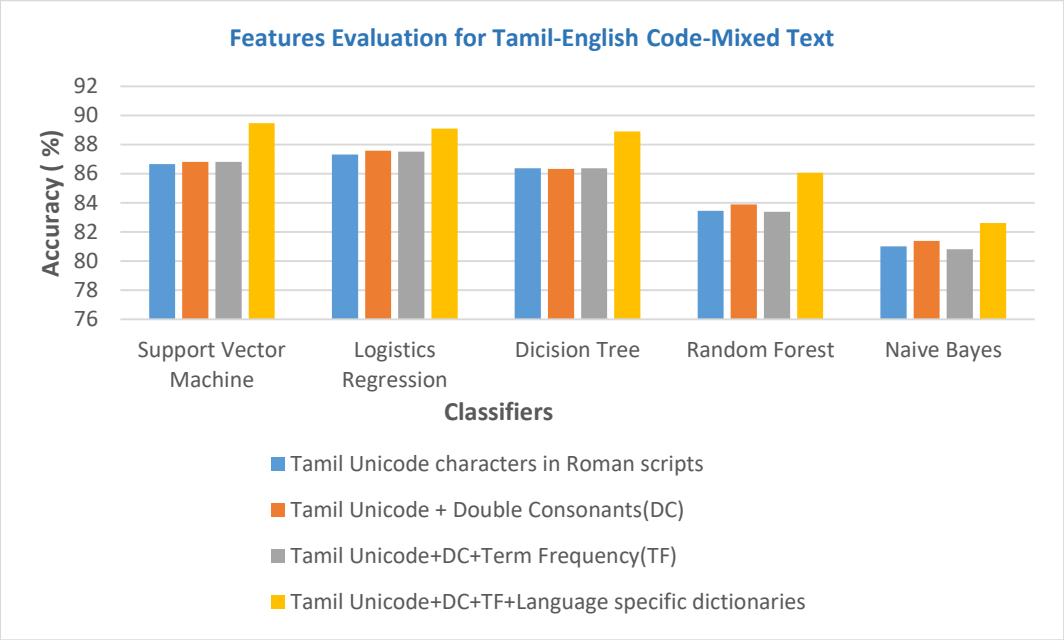


Figure 7.3 Evaluation of features impotency for Tamil-English code-mixed text

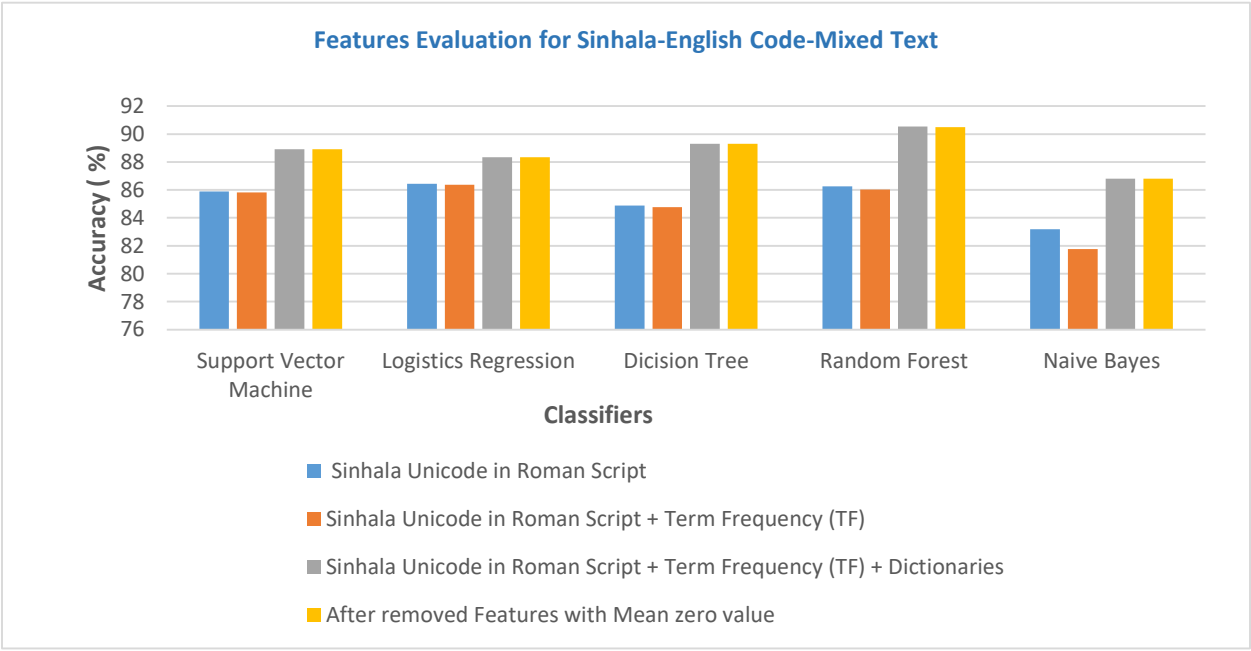


Figure 7.4 Evaluation of features impotency for Sinhala-English code-mixed text

7.3.2 Experiment Results for Model Testing

The testing results obtained from Tanglish model and Singlish model presented in this section. The confusion matrix, overall accuracy and F-Measures of 'tam', 'sin', and 'eng' tags obtained from testing process of Tanglish model and Singlish model shown in Table 7.4.

Table 7.4 Testing results obtained from Tanglish model and Singlish model

Models	Confusion Matrix				Accuracy	F-Measure (a=tam)	F-Measure (b=eng)
Tanglish model (SVM)		a	b	c	93.87%	0.965	0.894
	a	207	7	0			
	b	1	38	0			
	c	7	1	0			
	Confusion Matrix				Accuracy	F-Measure (a=sin)	F-Measure (b=eng)
Singlish Model (Random Forest)	a	177	1	1	95.83%	0.975	0.929
	b	3	26	0			
	c	4	0	4			

7.4 Summary

In this chapter, evaluation strategy, experimental study, datasets used and the results are discussed. Next chapter discusses the interpretation of the experimental results given in this chapter along with the conclusion and future work.

Chapter 8

Conclusion and Future Work

8.1 Introduction

In the previous chapter we discussed about the evaluation strategy along with the obtained results of the proposed solution. This chapter focuses on interpreting the results given in evaluation, discussing the limitations and future work of the solution.

8.2 Concluding remarks

This paper discusses problems with code mixed data and proposed a feature-based embedded methodology to automatic language identification of Tamil-English and Sinhala-English code mixed data. The methodology used for this system is a novel approach implemented as machine learning classifier based on features such as Tamil Unicode characters in Roman scripts, dictionaries, double consonant, and term frequency for Tamil-English code-mixed text and features such as Sinhala Unicode characters in Roman scripts, dictionaries, and term frequency for Sinhala-English code-mixed text. Different machine learning classifiers such as SVM, Random Forest, Naive Bayes, Logistic Regression, and Decision Tree used to evaluate the performance.

Among the predictive models for Tamil-English code-mixed text, SVM with a linear kernel gave 89.46% accuracy for language identification system for Tamil-English code-mixed text at the word level. This model is good for identifying Tamil and English language tags. Because the F-Measure for ‘tam’ and ‘eng’ tags were 0.947 and 0.806 respectively. But this model not much identified ‘rest’ tags properly. This was seen that most words belong to ‘rest’ tag incorrectly classified in ‘tam’ and ‘eng’ tags. This was happened because of some words were mixed with Tamil and English language. For example “ricela” word “rice” belongs to English language and “la” belongs to the Tamil language.

Among the predictive models for Sinhala-English code-mixed text, Random forest classifier gave 90.5% accuracy for language identification system for Sinhala-English code-mixed text at the word level. This model is good for identifying Sinhala and English language tags. Because the F-Measure for ‘sin’ and ‘eng’ tags were 0.949 and 0.758 respectively. But Random forest model not

much identified ‘rest’ tags properly. The F-measure of ‘rest’ for Random forest classification model were 0.513. This was seen that most words belong to ‘rest’ tag incorrectly classified in ‘sin’ and ‘eng’ tags. This was happened because of some words were mixed with numbers, Sinhala and English language. For example “container1ka” word “container” belongs to English language, “ka” belongs to the Sinhala language and ‘1’ belongs to number. Also name entities identified wrongly in ‘sin’ and ‘eng’ tags.

In the testing process of Tanglish model with SVM and Singlish model with Random Forest gave accuracy as 93.87% and 95.83% respectively. Tanglish model with SVM gave F-Measure for ‘tam’ and ‘eng’ tags were 0.965 and 0.894 respectively for the testing process. Singlish model with Random Forest gave F-Measure for ‘sin’ and ‘eng’ tags were 0.975 and 0.929 respectively for the testing process. So this the evidence that the Tanglish model with SVM and Singlish model with Random Forest most of the times predict the language labels correctly at word level.

8.3 Limitation and Future work

In this research mainly focused with language identification of noisy code-mixed social media data. Because of spelling mistakes and different forms of writing styles the code-mixed data becomes noisy. So it was challenged to identify the language tags of those word. Since some of the word were correctly identifies using some rules like eliminating wordplay characters appears in the word. Also LEXNORM corpus used to identify the spelling variation appears in English words. But there were some words mixed with two languages at word level. For example in the “studentskku” word “”student” belongs to English language and “kku” belongs to Tamil language. So these kind of word incorrectly classified to “tam” and “eng” tags. As a future work code-mixing within word can be detecting by segment the word as smaller units. The word composed of sequences of subunits associated with different languages, then the language tags can be detected for the subunits of word.

As further to improve the performance of language identification system more features can be added into the system. For example context information of the current word, like previous and after word language tags can be add as the features. Also can evaluate the features importance of the model by pruning some features by parameter tuning. By changing the parameters option in classifiers can able to identify the most dominating features for the dataset. Also, the performance

of the model can be evaluated with neural network techniques like multi-layer perception, recurrent neural networks and modern Deep Learning approaches.

8.4 Summary

In chapter 1, aim and objectives were defined for this research project. First, we need to get a comprehensive background knowledge on the selected research area namely, word level language identification of code-mixing text in social media. Once the importance of the research problem is identified in chapter 1, study is performed on the current approaches for language identification of code-mixed text to address the word level language identification of code-mixed text problem in chapter 2. Word level language identification system for code-mixed text is designed and developed using technologies such as natural language processing and machine learning as given in chapter 3, chapter 4 and chapter 5. Extensive evaluation that is conducted for the proposed solution is given along with the experimental setup and the obtained results in chapter 6. Accordingly, it is evident that all the objectives defined at the beginning of the project are successfully met in this research.

References

[1] A. Das and B. Gambäck, “Code-mixing in social media text: the last language identification frontier?,” 2015.

[2] P. McNamee, “Language Identification: A Solved Problem Suitable for Undergraduate Instruction,” J Comput Sci Coll, vol. 20, no. 3, pp. 94–101, Feb. 2005.

[3] U. Barman, A. Das, J. Wagner, and J. Foster, “Code mixing: A challenge for language identification in the language of social media,” in Proceedings of the first workshop on computational approaches to code switching, 2014, pp. 13–23.

[4] A. Das and B. Gambäck, “Identifying languages at the word level in code-mixed indian social media text,” 2014.

[5] D. Crystal, *Language and the Internet*. Cambridge [u.a.: Cambridge Univ. Press, 2008.

[6] T. Hidayat, *An analysis of code switching used by facebookers*. Sekolah Tinggi Keguruan dan Ilmu Pendidikan (STKIP) Siliwangi Bandung, 2008.

[7] P. Muysken, “code-switching and grammatical theory,” p. 23.

[8] P. Muysken, Bilingual speech: a typology of code-mixing, Nachdr. Cambridge: Cambridge Univ. Press, 2002.

[9] “Sociolinguistics by Bernard Spolsky,” p. 5.

[10] L. S. Kia, X. Cheng, T. K. Yee, and C. W. Ling, “Code-Mixing of English in the Entertainment News of Chinese Newspapers in Malaysia,” Int. J. Engl. Linguist., vol. 1, no. 1, Mar. 2011.

[11] S. Ajita and K. Amit, “POS Tagging of Hindi-English Code Mixed Text from Social Media.”

[12] A. Ehsan and S. A. Aziz, “CODE-MIXING IN URDU NEWS OF A PRIVATE PAKISTANI CHANNEL: A CASE STUDY,” vol. 5, no. 1, p. 10, 2014.

[13] T. Hidayat, An analysis of code switching used by facebookers. Sekolah Tinggi Keguruan dan Ilmu Pendidikan (STKIP) Siliwangi Bandung, 2008.

[14] S. Ghosh, S. Ghosh, and D. Das, “Complexity Metric for Code-Mixed Social Media Text,” ArXiv170701183 Cs, Jul. 2017.

[15] F. Fahmee and M. F. Yong, “Language Choice in Online Written Communication among Maldivian Professionals,” 3L Lang. Linguist. Lit., vol. 22, no. 2, 2016.

[16] A. K. Joshi, “processing of sentences with intra-sentential code-switching,” in Coling 1982: Proceedings of the Ninth International Conference on Computational Linguistics, 1982.

[17] E. M. GOLD, “Language Identification in the Limit,” p. 28.

[18] P. V. Veena, M. Anand Kumar, and K. P. Soman, “Character Embedding for Language Identification in Hindi-English Code-mixed Social Media Text,” Comput. Sist., vol. 22, no. 1, Mar. 2018.

[19] D. Nguyen and A. S. Doğruöz, "Word Level Language Identification in Online Multilingual Communication," in Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, Seattle, Washington, USA, 2013, pp. 857–862.

[20] C. Lignos and B. E. Al, Toward Web-scale Analysis of Codeswitching.

[21] R. Řehůřek and M. Kolkus, "Language identification on the web: Extending the dictionary method," in International Conference on Intelligent Text Processing and Computational Linguistics, 2009, pp. 357–368.

[22] W. B. Cavnar and J. M. Trenkle, "N-gram-based text categorization," *Ann Arbor Mi*, vol. 48113, no. 2, pp. 161–175, 1994.

[23] T. Joachims, "Svmlight: Support vector machine," SVM-Light Support Vector Mach. Httpsvmlight Joachims Org Univ. Dortm., vol. 19, no. 4, 1999.

[24] T. Baldwin and M. Lui, "Language identification: The long and the short of the matter," in Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, 2010, pp. 229–237.

[25] B. King and S. Abney, "Labeling the Languages of Words in Mixed-Language Documents using Weakly Supervised Methods," in Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Atlanta, Georgia, 2013, pp. 1110–1119.

[26] Y. Vyas, S. Gella, J. Sharma, K. Bali, and M. Choudhury, "POS Tagging of English-Hindi Code-Mixed Social Media Content," in Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 2014, pp. 974–979.

[27] M. Piergallini, R. Shirvani, G. S. Gautam, and M. Chouikha, "Word-Level Language Identification and Predicting Codeswitching Points in Swahili-English Language Data," in Proceedings of the Second Workshop on Computational Approaches to Code Switching, Austin, Texas, 2016, pp. 21–29

[28] S. Carter, W. Weerkamp, and M. Tsagkias, "Microblog language identification: overcoming the limitations of short, unedited and idiomatic text," Lang. Resour. Eval., vol. 47, no. 1, pp. 195–215, Mar. 2013.

[29] T. Solorio et al., "Overview for the First Shared Task on Language Identification in Code-Switched Data," in Proceedings of the First Workshop on Computational Approaches to Code Switching, Doha, Qatar, 2014, pp. 62–72.

[30] G. Chittaranjan, Y. Vyas, K. Bali, and M. Choudhury, "Word-level Language Identification using CRF: Code-switching Shared Task Report of MSR India System," in Proceedings of the First Workshop on Computational Approaches to Code Switching, Doha, Qatar, 2014, pp. 73–79.

[31] R. Kumar, A. Kumar, K. P. Soman, and K. P. Soman, "AmritaCEN_NLP@ FIRE 2015 Language Identification for Indian Languages in Social Media Text," 2015.

[32] S. Deepu and R. Pethuru, "A Framework for Text Analytics using the Bag of Words (BoW) Model for Prediction," Int. J. Adv. Netw. Appl. IJANA.

[33] E. Charniak, Introduction to Artificial Intelligence. Pearson Education India, 1985

[34] P. M. Nadkarni, L. Ohno-Machado, and W. W. Chapman, “Natural language processing: an introduction,” J. Am. Med. Inform. Assoc., vol. 18, no. 5, pp. 544–551, Sep. 2011.

[35] S. Bird, E. Klein, and E. Loper, Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit. O’Reilly Media, Inc., 2009.

[36] W.-L. Chao, “Machine Learning Tutorial,” p. 56.

[37] S. R. Kalmegh, “Comparative Analysis of WEKA Data Mining Algorithm RandomForest, RandomTree and LADTree for Classification of Indigenous News Data,” vol. 5, no. 1, p. 11, 2015.

[38] F. Pérez-Cruz, P. L. Alarcón-Diana, A. Navia-Vázquez, and A. Artés-Rodríguez, “Fast Training of Support Vector Classifiers,” in Advances in Neural Information Processing Systems 13, T. K. Leen, T. G. Dietterich, and V. Tresp, Eds. MIT Press, 2001, pp. 734–740.

[39] S. le Cessie and J. C. van Houwelingen, “Ridge Estimators in Logistic Regression,” J. R. Stat. Soc. Ser. C Appl. Stat., vol. 41, no. 1, pp. 191–201, Mar. 1992.

[40] George H. John, Pat Langley: Estimating Continuous Distributions in Bayesian Classifiers. In: Eleventh Conference on Uncertainty in Artificial Intelligence, San Mateo, 338-345, 1995.

[41] I. H. Witten, E. Frank, L. Trigg, M. Hall, G. Holmes, and S. J. Cunningham, “Weka: Practical Machine Learning Tools and Techniques with Java Implementations,” p. 5.

[42] S. L. Salzberg, “C4.5: Programs for Machine Learning by J. Ross Quinlan. Morgan Kaufmann Publishers, Inc., 1993,” Mach. Learn., vol. 16, no. 3, pp. 235–240, Sep. 1994.

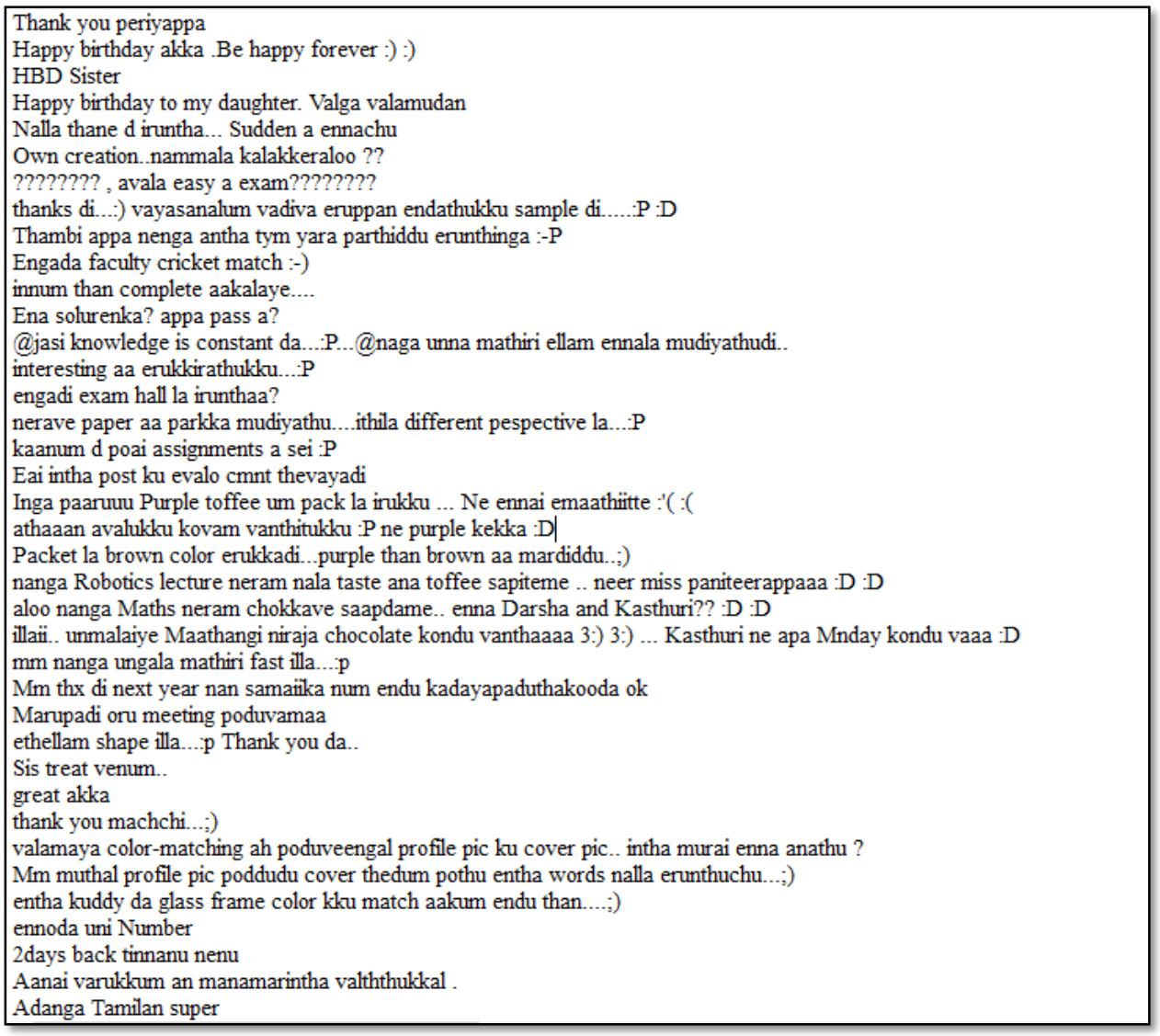
[43] L. Breiman, “Random Forests,” Mach. Learn., vol. 45, no. 1, pp. 5–32, Oct. 2001.

[44] G. Nelson, “Guy Aston and Lou Burnard, The BNC handbook: exploring the British National Corpus with SARA. Edinburgh Textbooks in Empirical Linguistics. Edinburgh: Edinburgh University Press, 1998. Pp. 256. Hardback £43.50, ISBN 0 7486 1054 5; paperback £16.50, ISBN 0 7486 1055 3,” Engl. Lang. Linguist., vol. 6, no. 1, pp. 197–221, May 2002.

[45] B. Han, P. Cook, and T. Baldwin, “Automatically constructing a normalisation dictionary for microblogs,” in Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning, 2012, pp. 421–432.

Appendix A

Sample of datasets used in evaluation process



Thank you periyappa
Happy birthday akka .Be happy forever :) :)
HBD Sister
Happy birthday to my daughter. Valga valamudan
Nalla thane d iruntha... Sudden a ennachu
Own creation..nammala kalakkeraloo ??
???????? , avala easy a exam????????
thanks di...:) vayasanalum vadiva eruppan endathukku sample di.....:P :D
Thambi appa nenga antha tym yara parthiddu erunthinga :-P
Engada faculty cricket match :-)
innum than complete aakalaye....
Ena solurenka? appa pass a?
@jasi knowledge is constant da...:P...@naga unna mathiri ellam ennala mudiyathudi..
interesting aa erukkirathukku...:P
engadi exam hall la irunthaa?
nerave paper aa parkka mudiyathu....ithila different pespective la...:P
kaanum d poai assignments a sei :P
Eai intha post ku evalo cmnt thevayadi
Inga paaruuu Purple toffee um pack la irukku ... Ne ennai emaathiitte :'(:(
athaaan avalukku kovam vanthitukku :P ne purple kekka :D
Packet la brown color erukkadi...purple than brown aa mardiddu..;)
nanga Robotics lecture neram nala taste ana toffee sapiteme .. neer miss paniteerappaaa :D :D
aloo nanga Maths neram chokkave saapdame.. enna Darsha and Kasthuri?? :D :D
illaii.. unmalaiye Maathangi niraja chocolate kondu vanthaaaa 3:) 3:) ... Kasthuri ne apa Mnday kondu vaaa :D
mm nanga ungala mathiri fast illa...:p
Mm thx di next year nan samaiika num endu kadayapaduthakooda ok
Marupadi oru meeting poduvamaa
ethellam shape illa...:p Thank you da..
Sis treat venum..
great akka
thank you machchi...;)
valamaya color-matching ah poduveengal profile pic ku cover pic.. intha murai enna anathu ?
Mm muthal profile pic poddudu cover thedum pothu entha words nalla erunthuchu...;)
entha kuddy da glass frame color kku match aakum endu than....;)
ennoda uni Number
2days back tinnanu nenu
Aanai varukkum an manamarintha valththukkal .
Adanga Tamilan super

Figure A.1 Sample of Tamil-English code-mixed dataset

Meka wath ehenam damu e man kiyanna
Ehakota lakshan inne kohomada
Den meke man ko ?
Honda prashnayak akke..☺ blouse eka thama tag kare
e phone 1ke prashnayak.screen 1ke kawruth penniwa.nathnam kawruhari crop karala ..
Onna mama nm newei
screen eke pena okkoma photo ekata wadinawa
Metchara mahathata indalath math nane e 4to eke...
ayyage phone 1ka ..
balanna api dennata karana wenaskam. Api dennawama photo eken kapala
eka galen kurullo dennekma maranawa kiyanne mehema scene walata wenna athi
Nilupul aiyage ara gorilla glass effect ekenda mokadda ekenda danne na api dennawa kepune...
ahaka inna rilaw , gorillo allanna epa okata , gaanata cut karala tiyenne
Ow man hitanne e gorilla glass eken gorillo tika witarak filter karala aranda koheda.
Anna hari...eka thamai wela thiyenne
Wow....good job putha.
kolla goda yamin sitina bawa penawa,
Oya colombo gihin karana ewa anith aya karane na bro
Mana ube kathawa neh oke tiyenne
Matath tama matakai uncle issara oyawa iskoleta ekkaragena ena widiya vas..
Mama oyage wedding eke photos balala mata ae kale matak una vas..
Ara balance karana ekatanam jehan one na man hitanne
Meka liyanna hethuva mokakda danne ne neda
Athi yantham Nalika innawa hodata
Ova maha katha vada Vasana akke, Sambolai bathui hari hadala denava
Ammo nalika sathiyak yanakan naanna epa..
Ape akka kohomath nanne nane😂..akke
Me taram lagata awa nam mawa balannath enna tibbane walasmulle nangoooo
Shah maru maru kattiya set wela
Aye amataka nowena plain t ekak dunn ooon
ayye ai ohoma.
ubath warenko ban!!
Adoooo nena muuu treat dela
Hik hik hee.... tawa podden අපිත ම wena ledak hedenwa! Paw
awa podden Nisal aiyata heart attack hedenwa!!
Thibila Api danne na neh...
Ekath narakama ne
ekanam hoda wedak malli

Figure A.2 Sample of Sinhala-English code-mixed dataset

Detailed experiment results of models obtained by different classifiers for Tamil-English code-mixed text

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        7700               89.4621 %
Incorrectly Classified Instances       907               10.5379 %
Kappa statistic                          0.6556
Mean absolute error                      0.2518
Root mean squared error                  0.3216
Relative absolute error                108.163  %
Root relative squared error             94.2785 %
Total Number of Instances             8607

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.983    0.356    0.913      0.983    0.947      0.719    0.821     0.914     tam
                 0.786    0.025    0.826      0.786    0.806      0.777    0.923     0.717     eng
                 0.154    0.010    0.546      0.154    0.240      0.263    0.567     0.149     rest
Weighted Avg.    0.895    0.286    0.874      0.895    0.875      0.692    0.815     0.831

=== Confusion Matrix ===

    a    b    c   <-- classified as
 6705   82   32 |    a = tam
  193  895   51 |    b = eng
  443  106  100 |    c = rest
```

Figure A.3 Results obtained by SVM classifier for Tamil-English code-mixed text

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        7668               89.0903 %
Incorrectly Classified Instances       939               10.9097 %
Kappa statistic                          0.6542
Mean absolute error                      0.1028
Root mean squared error                  0.2328
Relative absolute error                 44.1404 %
Root relative squared error             68.2502 %
Total Number of Instances             8607

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.975    0.322    0.920      0.975    0.947      0.722    0.934     0.973     tam
                 0.783    0.029    0.805      0.783    0.794      0.763    0.962     0.843     eng
                 0.194    0.019    0.460      0.194    0.273      0.264    0.815     0.341     rest
Weighted Avg.    0.891    0.260    0.870      0.891    0.876      0.693    0.929     0.908

=== Confusion Matrix ===

    a    b    c   <-- classified as
 6650   93   76 |    a = tam
  175  892   72 |    b = eng
  400  123  126 |    c = rest
```

Figure A.4 Results obtained by Logistic Regression classifier for Tamil-English code-mixed text

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        7110               82.6072 %
Incorrectly Classified Instances      1497               17.3928 %
Kappa statistic                          0.5537
Mean absolute error                      0.1343
Root mean squared error                  0.2975
Relative absolute error                 57.6668 %
Root relative squared error             87.2188 %
Total Number of Instances             8607

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.876    0.191    0.946      0.876    0.910      0.629    0.911     0.968     tam
                 0.783    0.059    0.669      0.783    0.721      0.678    0.952     0.799     eng
                 0.376    0.090    0.255      0.376    0.304      0.241    0.766     0.225     rest
Weighted Avg.    0.826    0.166    0.857      0.826    0.839      0.606    0.906     0.890

=== Confusion Matrix ===

    a    b    c   <-- classified as
 5974  282  563 |    a = tam
   97  892  150 |    b = eng
  245  160  244 |    c = rest
```

Figure A.5 Results obtained by Naïve Bayes classifier for Tamil-English code-mixed text

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        7651               88.8928 %
Incorrectly Classified Instances       956               11.1072 %
Kappa statistic                          0.6506
Mean absolute error                      0.1102
Root mean squared error                  0.2522
Relative absolute error                 47.3194 %
Root relative squared error             73.9234 %
Total Number of Instances             8607

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.971    0.322    0.920      0.971    0.945      0.713    0.864     0.931     tam
                 0.731    0.033    0.774      0.731    0.752      0.716    0.898     0.716     eng
                 0.299    0.017    0.584      0.299    0.396      0.386    0.691     0.323     rest
Weighted Avg.    0.889    0.260    0.875      0.889    0.878      0.688    0.856     0.856

=== Confusion Matrix ===

    a    b    c   <-- classified as
 6624  132   63 |    a = tam
  231  833   75 |    b = eng
  344  111  194 |    c = rest
```

Figure A.6 Results obtained by Decision Tree classifier for Tamil-English code-mixed text

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        7407               86.0579 %
Incorrectly Classified Instances      1200               13.9421 %
Kappa statistic                          0.4704
Mean absolute error                      0.1594
Root mean squared error                  0.2619
Relative absolute error                 68.4515 %
Root relative squared error             76.7781 %
Total Number of Instances             8607

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.994    0.619    0.860      0.994    0.922      0.550    0.918     0.969     tam
                 0.510    0.011    0.880      0.510    0.646      0.636    0.950     0.817     eng
                 0.069    0.002    0.763      0.069    0.127      0.216    0.784     0.352     rest
Weighted Avg.    0.861    0.492    0.855      0.861    0.826      0.536    0.912     0.902

=== Confusion Matrix ===

    a    b    c   <-- classified as
 6781   34    4 |    a = tam
  548  581   10 |    b = eng
  559   45   45 |    c = rest
```

Figure A.7 Results obtained by Random Forest classifier for Tamil-English code-mixed text

Detailed experiment results of models obtained by different classifiers for Sinhala-English code-mixed text

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        2575               88.9157 %
Incorrectly Classified Instances       321               11.0843 %
Kappa statistic                          0.6154
Mean absolute error                      0.255
Root mean squared error                  0.3264
Relative absolute error                114.5306 %
Root relative squared error             97.8825 %
Total Number of Instances             2896

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.978    0.404    0.908      0.978    0.942      0.669    0.797     0.909     sin
                 0.713    0.030    0.788      0.713    0.749      0.712    0.879     0.625     eng
                 0.103    0.005    0.563      0.103    0.174      0.223    0.578     0.117     rest
Weighted Avg.    0.889    0.329    0.871      0.889    0.869      0.648    0.795     0.823

=== Confusion Matrix ===

    a    b    c   <-- classified as
 2274   44    6 |    a = sin
  106  283    8 |    b = eng
  125   32   18 |    c = rest
```

Figure A.8 Results obtained by SVM classifier for Sinhala-English code-mixed text

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        2558               88.3287 %
Incorrectly Classified Instances       338               11.6713 %
Kappa statistic                          0.6208
Mean absolute error                      0.0974
Root mean squared error                  0.2364
Relative absolute error                 43.7361 %
Root relative squared error             70.9086 %
Total Number of Instances             2896

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0.964    0.325    0.923      0.964    0.943      0.691    0.922     0.968     sin
               0.705    0.037    0.751      0.705    0.727      0.686    0.928     0.782     eng
               0.211    0.022    0.385      0.211    0.273      0.253    0.799     0.311     rest
Weighted Avg.  0.883    0.267    0.867      0.883    0.873      0.664    0.916     0.903

=== Confusion Matrix ===

    a    b    c   <-- classified as
 2241   55   28 |    a = sin
   86  280   31 |    b = eng
  100   38   37 |    c = rest
```

Figure A.9 Results obtained by Logistic Regression classifier for Sinhala-English code-mixed text

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        2514               86.8094 %
Incorrectly Classified Instances       382               13.1906 %
Kappa statistic                          0.5949
Mean absolute error                      0.1013
Root mean squared error                  0.261
Relative absolute error                 45.5187 %
Root relative squared error             78.2607 %
Total Number of Instances             2896

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0.940    0.280    0.932      0.940    0.936      0.670    0.931     0.980     sin
               0.804    0.070    0.644      0.804    0.715      0.670    0.957     0.828     eng
               0.057    0.017    0.179      0.057    0.087      0.070    0.793     0.202     rest
Weighted Avg.  0.868    0.235    0.847      0.868    0.854      0.634    0.926     0.912

=== Confusion Matrix ===

    a    b    c   <-- classified as
 2185  105   34 |    a = sin
   66  319   12 |    b = eng
   94   71   10 |    c = rest
```

Figure A.10 Results obtained by Naïve Bayes classifier for Sinhala-English code-mixed text

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        2586               89.2956 %
Incorrectly Classified Instances       310               10.7044 %
Kappa statistic                          0.6472
Mean absolute error                      0.1024
Root mean squared error                  0.2487
Relative absolute error                 45.9931 %
Root relative squared error             74.5857 %
Total Number of Instances             2896

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0.968     0.346     0.919      0.968     0.943      0.685     0.852     0.923     sin
               0.675     0.027     0.798      0.675     0.731      0.696     0.854     0.710     eng
               0.389     0.016     0.607      0.389     0.474      0.460     0.704     0.334     rest
Weighted Avg.  0.893     0.282     0.884      0.893     0.886      0.673     0.843     0.858

=== Confusion Matrix ===

    a    b    c   <-- classified as
 2250   50   24 |    a = sin
  109  268   20 |    b = eng
   89   18   68 |    c = rest
```

Figure A.11 Results obtained by Decision Tree classifier for Sinhala-English code-mixed text

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        2621               90.5041 %
Incorrectly Classified Instances       275                9.4959 %
Kappa statistic                          0.6789
Mean absolute error                      0.1102
Root mean squared error                  0.2217
Relative absolute error                 49.4902 %
Root relative squared error             66.4986 %
Total Number of Instances             2896

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0.981     0.350     0.919      0.981     0.949      0.717     0.949     0.983     sin
               0.683     0.019     0.852      0.683     0.758      0.730     0.966     0.862     eng
               0.400     0.010     0.714      0.400     0.513      0.514     0.851     0.532     rest
Weighted Avg.  0.905     0.284     0.898      0.905     0.897      0.706     0.945     0.939

=== Confusion Matrix ===

    a    b    c   <-- classified as
 2280   24   20 |    a = sin
  118  271    8 |    b = eng
   82   23   70 |    c = rest
```

Figure A.12 Results obtained by Random Forest classifier for Sinhala-English code-mixed text

Appendix B

Code section for Identification of Features for a given Sentence

```
import re
from nltk.tokenize import word_tokenize
import json
import pandas as pd
import numpy as np

def preProcessing(sentence):
    cleanData=re.sub('[.@/]+', ' ', sentence)
    cleanDataLower=cleanData.lower()

    print("\nTotal words in the sentence")
    sentenceWordList = word_tokenize(cleanDataLower)
    print(len(sentenceWordList))

    words=[re.sub(r'(.)\1+', r'\1\1', x).strip() for x in sentenceWordList]
    print("\nWords in the sentence")
    print(words)
    return words
words=preProcessing("blouse eka thama tag kare")
```

```
Total words in the sentence
5

Words in the sentence
['blouse', 'eka', 'thama', 'tag', 'kare']
```

```
def BNC_Corpus():
    from nltk.corpus.reader.bnc import BNCCorpusReader
    from nltk.collocations import BigramAssocMeasures, BigramCollocationFinder

    # Instantiate the reader
    bnc_reader = BNCCorpusReader(root="Corpora/bnc/Texts", fileids=r'[A-K]/\w*/\w*\.xml')

    list_of_fileids = ['A/A0/A00.xml', 'A/A0/A01.xml']
    #bigram_measures = BigramAssocMeasures()
    #finder = BigramCollocationFinder.from_words(bnc_reader.words(fileids=list_of_fileids))
    #scored = finder.score_ngrams(bigram_measures.raw_freq)
    bnc_words=bnc_reader.words(fileids=list_of_fileids, strip_space=True)
    #print(bnc_words)

    BncList=[]
    for w in bnc_words:
        BncList.append(w.lower())
    return BncList
```

```
import json
def LexNorm_Corpus():
    with open('train_data_20150430.json') as f:
        data = json.load(f)
    #print(data[1]["input"])
    normalizeListInput=[]
    normalizeListOutput=[]
    normalizeList1=[]
    key=0
    for w in data:
        normalizeListInput.extend(data[key]["input"])
        normalizeListOutput.extend(data[key]["output"])
        key+=1
    normalizeList1=normalizeListInput+normalizeListOutput
    normalizeList=[]
    for w in normalizeList1:
        normalizeList.append(w.lower())
    return normalizeList
```

```
def Tanglish_Corpus_Create():
    with open('Taglish_corpus.txt', encoding="utf8") as t:
        Data=t.read()
        clData=re.sub('[.@/]+', ' ', Data)
        cleanData=clData.lower()
        WordList = word_tokenize(cleanData)
        Words=[re.sub(r'(.)\1+', r'\1\1', x).strip() for x in WordList]
        wordfreq = []

        for w in Words:
            wordfreq.append(Words.count(w))

        for a,b in zip(Words,wordfreq):
            WordFreqDict=dict(zip(Words,wordfreq))

    with open('Tanglish_Corpus.json', 'w') as fp:
        json.dump(WordFreqDict, fp)

Tanglish_Corpus_Create()
```

```
def Tanglish_Corpus():
    with open('Tanglish_Corpus.json', 'r') as fp:
        data = json.load(fp)
    return data
```

```
def Singlish_Corpus_Create():
    with open('Singlish_corpus.txt', encoding="utf8") as t:
        Data=t.read()
        clData=re.sub('[.@/]+', ' ', Data)
        cleanData=clData.lower()
        WordList = word_tokenize(cleanData)
        Words=[re.sub(r'(.)\1+', r'\1\1', x).strip() for x in WordList]
        wordfreq = []

        for w in Words:
            wordfreq.append(Words.count(w))

        for a,b in zip(Words,wordfreq):
            WordFreqDict=dict(zip(Words,wordfreq))

    with open('Singlish_Corpus.json', 'w') as fp:
        json.dump(WordFreqDict, fp)

Singlish_Corpus_Create()
```

```
def Singlish_Corpus():
    with open('Singlish_Corpus.json', 'r') as fp:
        data = json.load(fp)
    return data
```

```
def featureExtraction_Tanglish(words):
    features={'Words': '', 'TermFreq': '', 'a': '', 'aa': '', 'ae': '', 'ai': '', 'bb': '', 'cc': '', 'dd': '', 'e': '', 'ee':
'', 'ff': '', 'gg': '', 'gn': '', 'gna': '', 'gne': '', 'gni': '', 'gnu': '', 'h': '', 'ha': '', 'haa': '', 'hae': '', 'hai': ''
, 'he': '', 'hi': '', 'hii': '', 'ho': '', 'how': '', 'hu': '', 'huu': '', 'i': '', 'ii': '', 'j': '', 'ja': '', 'jaa': '', 'ja
i': '', 'je': '', 'ji': '', 'jo': '', 'ju': '', 'k': '', 'ka': '', 'kaa': '', 'kai': '', 'ke': '', 'kea': '', 'ki': '', 'kk': ''
, 'ko': '', 'ksh': '', 'ksha': '', 'ku': '', 'kuu': '', 'l': '', 'la': '', 'laa': '', 'lae': '', 'lai': '', 'le': '', 'li': '',
'll': '', 'lo': '', 'loa': '', 'low': '', 'lu': '', 'm': '', 'ma': '', 'maa': '', 'mae': '', 'mai': '', 'me': '', 'mi': '', 'mm'
: '', 'mo': '', 'mu': '', 'n': '', 'na': '', 'naa': '', 'nae': '', 'nai': '', 'ne': '', 'ng': '', 'nga': '', 'ngaa': '', 'ngai':
'', 'nge': '', 'ngi': '', 'ngo': '', 'ngu': '', 'nh': '', 'nha': '', 'ni': '', 'nii': '', 'nn': '', 'no': '', 'now': '', 'nu':
'', 'nuu': '', 'o': '', 'oa': '', 'oo': '', 'ow': '', 'p': '', 'pa': '', 'paa': '', 'pae': '', 'pai': '', 'pe': '', 'pi': '', 'p
o': '', 'poa': '', 'pow': '', 'pp': '', 'pu': '', 'puu': '', 'q': '', 'r': '', 'ra': '', 'raa': '', 'rae': '', 'rai': '', 're':
'', 'ri': '', 'rii': '', 'ro': '', 'roa': '', 'row': '', 'rr': '', 'ru': '', 'ruu': '', 's': '', 'sa': '', 'saa': '', 'sai': '',
'se': '', 'sh': '', 'sha': '', 'shaa': '', 'she': '', 'shi': '', 'sho': '', 'shu': '', 'si': '', 'so': '', 'sri': '', 'ss': '',
'su': '', 'suu': '', 't': '', 'ta': '', 'taa': '', 'tae': '', 'tai': '', 'te': '', 'th': '', 'tha': '', 'thaa': '', 'thae': '',
'thai': '', 'the': '', 'thi': '', 'thii': '', 'tho': '', 'thu': '', 'thuu': '', 'ti': '', 'to': '', 'tow': '', 'tt': '', 'tu':
'', 'tuu': '', 'u': '', 'uu': '', 'v': '', 'va': '', 'vaa': '', 'vae': '', 'vai': '', 've': '', 'vi': '', 'vii': '', 'vo': '',
'voa': '', 'vu': '', 'vuu': '', 'y': '', 'ya': '', 'yaa': '', 'yae': '', 'yai': '', 'ye': '', 'yi': '', 'yii': '', 'yo': '', 'y
u': '', 'yy': '', 'z': '', 'za': '', 'zai': '', 'ze': '', 'zi': '', 'zu': '', 'BNC_Corpus': '', 'LexNorm_Corpus': ''}

    first_row=['Words', 'TermFreq', 'a', 'aa', 'ae', 'ai', 'bb', 'cc', 'dd', 'e', 'ee', 'ff', 'gg', 'gn', 'gna', 'gne', 'gni',
'gnu', 'h', 'ha', 'haa', 'hae', 'hai', 'he', 'hi', 'hii', 'ho', 'how', 'hu', 'huu', 'i', 'ii', 'j', 'ja', 'jaa', 'jai', 'je', 'j
i', 'jo', 'ju', 'k', 'ka', 'kaa', 'kai', 'ke', 'kea', 'ki', 'kk', 'ko', 'ksh', 'ksha', 'ku', 'kuu', 'l', 'la', 'laa', 'lae', 'la
i', 'le', 'li', 'll', 'lo', 'loa', 'low', 'lu', 'm', 'ma', 'maa', 'mae', 'mai', 'me', 'mi', 'mm', 'mo', 'mu', 'n', 'na', 'naa',
'nae', 'nai', 'ne', 'ng', 'nga', 'ngaa', 'ngai', 'nge', 'ngi', 'ngo', 'ngu', 'nh', 'nha', 'ni', 'nii', 'nn', 'no', 'now', 'nu',
'nuu', 'o', 'oa', 'oo', 'ow', 'p', 'pa', 'paa', 'pae', 'pai', 'pe', 'pi', 'po', 'poa', 'pow', 'pp', 'pu', 'puu', 'q', 'r', 'ra',
'raa', 'rae', 'rai', 're', 'ri', 'rii', 'ro', 'roa', 'row', 'rr', 'ru', 'ruu', 's', 'sa', 'saa', 'sai', 'se', 'sh', 'sha', 'sha
a', 'she', 'shi', 'sho', 'shu', 'si', 'so', 'sri', 'ss', 'su', 'suu', 't', 'ta', 'taa', 'tae', 'tai', 'te', 'th', 'tha', 'thaa',
'thae', 'thai', 'the', 'thi', 'thii', 'tho', 'thu', 'thuu', 'ti', 'to', 'tow', 'tt', 'tu', 'tuu', 'u', 'uu', 'v', 'va', 'vaa',
'vae', 'vai', 've', 'vi', 'vii', 'vo', 'voa', 'vu', 'vuu', 'y', 'ya', 'yaa', 'yae', 'yai', 'ye', 'yi', 'yii', 'yo', 'yu', 'yy',
'z', 'za', 'zai', 'ze', 'zi', 'zu', 'BNC_Corpus', 'LexNorm_Corpus']

    BncList=BNC_Corpus()
    LexNorm=LexNorm_Corpus()
    Tanglish=Tanglish_Corpus()
    data=[]

    testIndex=0
    for w in words:
        features1 = dict(features)
        features1["Words"]=w
        #data.append(features)
        data.insert(testIndex, features1)
        testIndex+=1


    newDict={}
    index=0
    for a in data:
        index+=1
        newDict[index] = a


    for l in newDict:
        word1=newDict[l]["Words"]
        #print(word1)


    for l in newDict:
        #print(l)
        str=newDict[l]["Words"]
        #str=words[l-1]
        #print(str)
        index1 = 2
        while index1 < len(first_row)-2:
            searchString= first_row[index1] in str
            #print(first_row[index1])

            #print(searchString)
            if(searchString==True):
                newDict[l][first_row[index1]]= 1
            else:
                newDict[l][first_row[index1]]= 0
            index1 += 1

        searchString1= str.lower() in BncList
        #print(searchString1)
        if(searchString1==True):
            newDict[l]["BNC_Corpus"]= 1
        else:
            newDict[l]["BNC_Corpus"]= 0

        searchString2= str.lower() in LexNorm
        #print(searchString)
        if(searchString2==True):
            newDict[l]["LexNorm_Corpus"]= 1
        else:
            newDict[l]["LexNorm_Corpus"]= 0

        if(str.lower() in Tanglish):
            newDict[l]["TermFreq"]= Tanglish[str]
        else:
            newDict[l]["TermFreq"]= 1
    #print(newDict)
    return newDict

Dictionary=featureExtraction_Tanglish(words)
```

```
def featureExtraction_Singlish(words):
    features={'Words': '', 'TermFreq': '', 'a': '', 'aa': '', 'ae': '', 'au': '', 'b': '', 'ba': '', 'baa': '', 'bae': '', 'bau'
: '', 'be': '', 'bea': '', 'bi': '', 'bo': '', 'bu': '', 'buu': '', 'ch': '', 'cha': '', 'chaa': '', 'che': '', 'chi': '', 'cho'
: '', 'chu': '', 'd': '', 'da': '', 'daa': '', 'dau': '', 'de': '', 'dea': '', 'dh': '', 'dha': '', 'dhe': '', 'dhi': '', 'dhu':
'', 'di': '', 'dii': '', 'do': '', 'du': '', 'e': '', 'ea': '', 'g': '', 'ga': '', 'gaa': '', 'gau': '', 'ge': '', 'gi': '', 'g
o': '', 'gu': '', 'h': '', 'ha': '', 'haa': '', 'he': '', 'hea': '', 'hi': '', 'hii': '', 'ho': '', 'hu': '', 'i': '', 'ii': '',
'j': '', 'ja': '', 'jaa': '', 'je': '', 'ji': '', 'jo': '', 'ju': '', 'k': '', 'ka': '', 'kaa': '', 'kau': '', 'ke': '', 'ki':
'', 'kii': '', 'ko': '', 'ku': '', 'kuu': '', 'l': '', 'la': '', 'laa': '', 'le': '', 'lea': '', 'li': '', 'lo': '', 'lu': '',
'm': '', 'ma': '', 'maa': '', 'mae': '', 'me': '', 'mi': '', 'mo': '', 'mu': '', 'muu': '', 'n': '', 'na': '', 'naa': '', 'ne':
'', 'nea': '', 'ni': '', 'no': '', 'nu': '', 'o': '', 'oe': '', 'r': '', 'ra': '', 'raa': '', 'rae': '', 'rau': '', 're': '', 'r
ea': '', 'ri': '', 'rii': '', 'ro': '', 'ru': '', 's': '', 'sa': '', 'saa': '', 'sae': '', 'sau': '', 'se': '', 'sea': '', 'si':
'', 'sii': '', 'so': '', 'su': '', 't': '', 'ta': '', 'taa': '', 'te': '', 'tea': '', 'th': '', 'tha': '', 'thaa': '', 'the': ''
, 'thi': '', 'thii': '', 'tho': '', 'thu': '', 'ti': '', 'to': '', 'tu': '', 'u': '', 'uu': '', 'w': '', 'wa': '', 'waa': '', 'w
e': '', 'wi': '', 'wo': '', 'wu': '', 'y': '', 'ya': '', 'yaa': '', 'yau': '', 'ye': '', 'yi': '', 'yo': '', 'yu': '', 'BNC_Corp
us': '', 'LexNorm_Corpus': ''}
    first_row=['Words', 'TermFreq', 'a', 'aa', 'ae', 'au', 'b', 'ba', 'baa', 'bae', 'bau', 'be', 'bea', 'bi', 'bo', 'bu', 'buu',
'ch', 'cha', 'chaa', 'che', 'chi', 'cho', 'chu', 'd', 'da', 'daa', 'dau', 'de', 'dea', 'dh', 'dha', 'dhe', 'dhi', 'dhu', 'di',
'dii', 'do', 'du', 'e', 'ea', 'g', 'ga', 'gaa', 'gau', 'ge', 'gi', 'go', 'gu', 'h', 'ha', 'haa', 'he', 'hea', 'hi', 'hii', 'ho',
'hu', 'i', 'ii', 'j', 'ja', 'jaa', 'je', 'ji', 'jo', 'ju', 'k', 'ka', 'kaa', 'kau', 'ke', 'ki', 'kii', 'ko', 'ku', 'kuu', 'l',
'la', 'laa', 'le', 'lea', 'li', 'lo', 'lu', 'm', 'ma', 'maa', 'mae', 'me', 'mi', 'mo', 'mu', 'muu', 'n', 'na', 'naa', 'ne', 'ne
a', 'ni', 'no', 'nu', 'o', 'oe', 'r', 'ra', 'raa', 'rae', 'rau', 're', 'rea', 'ri', 'rii', 'ro', 'ru', 's', 'sa', 'saa', 'sae',
'sau', 'se', 'sea', 'si', 'sii', 'so', 'su', 't', 'ta', 'taa', 'te', 'tea', 'th', 'tha', 'thaa', 'the', 'thi', 'thii', 'tho', 't
hu', 'ti', 'to', 'tu', 'u', 'uu', 'w', 'wa', 'waa', 'we', 'wi', 'wo', 'wu', 'y', 'ya', 'yaa', 'yau', 'ye', 'yi', 'yo', 'yu', 'BN
C_Corpus', 'LexNorm_Corpus']


    BncList=BNC_Corpus()
    LexNorm=LexNorm_Corpus()
    Singlish=Singlish_Corpus()
    data=[]

    testIndex=0
    for w in words:
        features1 = dict(features)
        features1["Words"]=w
        #data.append(features)
        data.insert(testIndex, features1)
        testIndex+=1

    newDict={}
    index=0
    for a in data:
        index+=1
        newDict[index] = a


    for l in newDict:
        word1=newDict[l]["Words"]
        #print(word1)


    for l in newDict:
        #print(l)
        #str=newDict[l]["Words"]
        str=words[l-1]
       # print(str)
        index1 = 2
        while index1 < len(first_row)-2:
            searchString= first_row[index1] in str
            #print(first_row[index1])

            if(searchString==True):
                newDict[l][first_row[index1]]= 1
            else:
                newDict[l][first_row[index1]]= 0
            index1 += 1

        searchString1= str.lower() in BncList
        #print(searchString1)
        if(searchString1==True):
            newDict[l]["BNC_Corpus"]= 1
        else:
            newDict[l]["BNC_Corpus"]= 0

        searchString2= str.lower() in LexNorm
        #print(searchString)
        if(searchString2==True):
            newDict[l]["LexNorm_Corpus"]= 1
        else:
            newDict[l]["LexNorm_Corpus"]= 0

        if(str.lower() in Singlish):
            newDict[l]["TermFreq"]= Singlish[str]
        else:
            newDict[l]["TermFreq"]= 1
    print(newDict)
    return newDict


Dictionary1=featureExtraction_Singlish(words)
```

{1: {'Words': 'blouse', 'TermFreq': 1, 'a': 0, 'aa': 0, 'ae': 0, 'au': 0, 'b': 1, 'ba': 0, 'baa': 0, 'bae': 0, 'bau': 0, 'be': 0, 'bea': 0, 'bi': 0, 'bo': 0, 'bu': 0, 'buu': 0, 'ch': 0, 'cha': 0, 'chaa': 0, 'che': 0, 'chi': 0, 'cho': 0, 'chu': 0, 'd': 0, 'da': 0, 'daa': 0, 'dau': 0, 'de': 0, 'dea': 0, 'dh': 0, 'dha': 0, 'dhe': 0, 'dhi': 0, 'dhu': 0, 'di': 0, 'dii': 0, 'do': 0, 'du': 0, 'e': 1, 'ea': 0, 'g': 0, 'ga': 0, 'gaa': 0, 'gau': 0, 'ge': 0, 'gi': 0, 'go': 0, 'gu': 0, 'h': 0, 'ha': 0, 'haa': 0, 'he': 0, 'hea': 0, 'hi': 0, 'hii': 0, 'ho': 0, 'hu': 0, 'i': 0, 'ii': 0, 'j': 0, 'ja': 0, 'jaa': 0, 'je': 0, 'ji': 0, 'jo': 0, 'ju': 0, 'k': 0, 'ka': 0, 'kaa': 0, 'kau': 0, 'ke': 0, 'ki': 0, 'kii': 0, 'ko': 0, 'ku': 0, 'kuu': 0, 'l': 1, 'la': 0, 'laa': 0, 'le': 0, 'lea': 0, 'li': 0, 'lo': 1, 'lu': 0, 'm': 0, 'ma': 0, 'maa': 0, 'mae': 0, 'me': 0, 'mi': 0, 'mo': 0, 'mu': 0, 'muu': 0, 'n': 0, 'na': 0, 'naa': 0, 'ne': 0, 'nea': 0, 'ni': 0, 'no': 0, 'nu': 0, 'o': 1, 'oe': 0, 'r': 0, 'ra': 0, 'raa': 0, 'rae': 0, 'rau': 0, 're': 0, 'rea': 0, 'ri': 0, 'rii': 0, 'ro': 0, 'ru': 0, 's': 1, 'sa': 0, 'saa': 0, 'sae': 0, 'sau': 0, 'se': 1, 'sea': 0, 'si': 0, 'sii': 0, 'so': 0, 'su': 0, 't': 0, 'ta': 0, 'taa': 0, 'te': 0, 'tea': 0, 'th': 0, 'tha': 0, 'thaa': 0, 'the': 0, 'thi': 0, 'thii': 0, 'tho': 0, 'thu': 0, 'ti': 0, 'to': 0, 'tu': 0, 'u': 1, 'uu': 0, 'w': 0, 'wa': 0, 'waa': 0, 'we': 0, 'wi': 0, 'wo': 0, 'wu': 0, 'y': 0, 'ya': 0, 'yaa': 0, 'yau': 0, 'ye': 0, 'yi': 0, 'yo': 0, 'yu': 0, 'BNC_Corpus': 0, 'LexNorm_Corpus': 0}, 2: {'Words': 'eka', 'TermFreq': 110, 'a': 1, 'aa': 0, 'ae': 0, 'au': 0, 'b': 0, 'ba': 0, 'baa': 0, 'bae': 0, 'bau': 0, 'be': 0, 'bea': 0, 'bi': 0, 'bo': 0, 'bu': 0, 'buu': 0, 'ch': 0, 'cha': 0, 'chaa': 0, 'che': 0, 'chi': 0, 'cho': 0, 'chu': 0, 'd': 0, 'da': 0, 'daa': 0, 'dau': 0, 'de': 0, 'dea': 0, 'dh': 0, 'dha': 0, 'dhe': 0, 'dhi': 0, 'dhu': 0, 'di': 0, 'dii': 0, 'do': 0, 'du': 0, 'e': 1, 'ea': 0, 'g': 0, 'ga': 0, 'gaa': 0, 'gau': 0, 'ge': 0, 'gi': 0, 'go': 0, 'gu': 0, 'h': 0, 'ha': 0, 'haa': 0, 'he': 0, 'hea': 0, 'hi': 0, 'hii': 0, 'ho': 0, 'hu': 0, 'i': 0, 'ii': 0, 'j': 0, 'ja': 0, 'jaa': 0, 'je': 0, 'ji': 0, 'jo': 0, 'ju': 0, 'k': 1, 'ka': 1, 'kaa': 0, 'kau': 0, 'ke': 0, 'ki': 0, 'kii': 0, 'ko': 0, 'ku': 0, 'kuu': 0, 'l': 0, 'la': 0, 'laa': 0, 'le': 0, 'lea': 0, 'li': 0, 'lo': 0, 'lu': 0, 'm': 0, 'ma': 0, 'maa': 0, 'mae': 0, 'me': 0, 'mi': 0, 'mo': 0, 'mu': 0, 'muu': 0, 'n': 0, 'na': 0, 'naa': 0, 'ne': 0, 'nea': 0, 'ni': 0, 'no': 0, 'nu': 0, 'o': 0, 'oe': 0, 'r': 0, 'ra': 0, 'raa': 0, 'rae': 0, 'rau': 0, 're': 0, 'rea': 0, 'ri': 0, 'rii': 0, 'ro': 0, 'ru': 0, 's': 0, 'sa': 0, 'saa': 0, 'sae': 0, 'sau': 0, 'se': 0, 'sea': 0, 'si': 0, 'sii': 0, 'so': 0, 'su': 0, 't': 0, 'ta': 0, 'taa': 0, 'te': 0, 'tea': 0, 'th': 0, 'tha': 0, 'thaa': 0, 'the': 0, 'thi': 0, 'thii': 0, 'tho': 0, 'thu': 0, 'ti': 0, 'to': 0, 'tu': 0, 'u': 0, 'uu': 0, 'w': 0, 'wa': 0, 'waa': 0, 'we': 0, 'wi': 0, 'wo': 0, 'wu': 0, 'y': 0, 'ya': 0, 'yaa': 0, 'yau': 0, 'ye': 0, 'yi': 0, 'yo': 0, 'yu': 0, 'BNC_Corpus': 0, 'LexNorm_Corpus': 1}, 3: {'Words': 'thama', 'TermFreq': 20, 'a': 1, 'aa': 0, 'ae': 0, 'au': 0, 'b': 0, 'ba': 0, 'baa': 0, 'bae': 0, 'bau': 0, 'be': 0, 'bea': 0, 'bi': 0, 'bo': 0, 'bu': 0, 'buu': 0, 'ch': 0, 'cha': 0, 'chaa': 0, 'che': 0, 'chi': 0, 'cho': 0, 'chu': 0, 'd': 0, 'da': 0, 'daa': 0, 'dau': 0, 'de': 0, 'dea': 0, 'dh': 0, 'dha': 0, 'dhe': 0, 'dhi': 0, 'dhu': 0, 'di': 0, 'dii': 0, 'do': 0, 'du': 0, 'e': 0, 'ea': 0, 'g': 0, 'ga': 0, 'gaa': 0, 'gau': 0, 'ge': 0, 'gi': 0, 'go': 0, 'gu': 0, 'h': 1, 'ha': 1, 'haa': 0, 'he': 0, 'hea': 0, 'hi': 0, 'hii': 0, 'ho': 0, 'hu': 0, 'i': 0, 'ii': 0, 'j': 0, 'ja': 0, 'jaa': 0, 'je': 0, 'ji': 0, 'jo': 0, 'ju': 0, 'k': 0, 'ka': 0, 'kaa': 0, 'kau': 0, 'ke': 0, 'ki': 0, 'kii': 0, 'ko': 0, 'ku': 0, 'kuu': 0, 'l': 0, 'la': 0, 'laa': 0, 'le': 0, 'lea': 0, 'li': 0, 'lo': 0, 'lu': 0, 'm': 1, 'ma': 1, 'maa': 0, 'mae': 0, 'me': 0, 'mi': 0, 'mo': 0, 'mu': 0, 'muu': 0, 'n': 0, 'na': 0, 'naa': 0, 'ne': 0, 'nea': 0, 'ni': 0, 'no': 0, 'nu': 0, 'o': 0, 'oe': 0, 'r': 0, 'ra': 0, 'raa': 0, 'rae': 0, 'rau': 0, 're': 0, 'rea': 0, 'ri': 0, 'rii': 0, 'ro': 0, 'ru': 0, 's': 0, 'sa': 0, 'saa': 0, 'sae': 0, 'sau': 0, 'se': 0, 'sea': 0, 'si': 0, 'sii': 0, 'so': 0, 'su': 0, 't': 1, 'ta': 0, 'taa': 0, 'te': 0, 'tea': 0, 'th': 1, 'tha': 1, 'thaa': 0, 'the': 0, 'thi': 0, 'thii': 0, 'tho': 0, 'thu': 0, 'ti': 0, 'to': 0, 'tu': 0, 'u': 0, 'uu': 0, 'w': 0, 'wa': 0, 'waa': 0, 'we': 0, 'wi': 0, 'wo': 0, 'wu': 0, 'y': 0, 'ya': 0, 'yaa': 0, 'yau': 0, 'ye': 0, 'yi': 0, 'yo': 0, 'yu': 0, 'BNC_Corpus': 0, 'LexNorm_Corpus': 0}, 4: {'Words': 'tag', 'TermFreq': 3, 'a': 1, 'aa': 0, 'ae': 0, 'au': 0, 'b': 0, 'ba': 0, 'baa': 0, 'bae': 0, 'bau': 0, 'be': 0, 'bea': 0, 'bi': 0, 'bo': 0, 'bu': 0, 'buu': 0, 'ch': 0, 'cha': 0, 'chaa': 0, 'che': 0, 'chi': 0, 'cho': 0, 'chu': 0, 'd': 0, 'da': 0, 'daa': 0, 'dau': 0, 'de': 0, 'dea': 0, 'dh': 0, 'dha': 0, 'dhe': 0, 'dhi': 0, 'dhu': 0, 'di': 0, 'dii': 0, 'do': 0, 'du': 0, 'e': 0, 'ea': 0, 'g': 1, 'ga': 0, 'gaa': 0, 'gau': 0, 'ge': 0, 'gi': 0, 'go': 0, 'gu': 0, 'h': 0, 'ha': 0, 'haa': 0, 'he': 0, 'hea': 0, 'hi': 0, 'hii': 0, 'ho': 0, 'hu': 0, 'i': 0, 'ii': 0, 'j': 0, 'ja': 0, 'jaa': 0, 'je': 0, 'ji': 0, 'jo': 0, 'ju': 0, 'k': 0, 'ka': 0, 'kaa': 0, 'kau': 0, 'ke': 0, 'ki': 0, 'kii': 0, 'ko': 0, 'ku': 0, 'kuu': 0, 'l': 0, 'la': 0, 'laa': 0, 'le': 0, 'lea': 0, 'li': 0, 'lo': 0, 'lu': 0, 'm': 0, 'ma': 0, 'maa': 0, 'mae': 0, 'me': 0, 'mi': 0, 'mo': 0, 'mu': 0, 'muu': 0, 'n': 0, 'na': 0, 'naa': 0, 'ne': 0, 'nea': 0, 'ni': 0, 'no': 0, 'nu': 0, 'o': 0, 'oe': 0, 'r': 0, 'ra': 0, 'raa': 0, 'rae': 0, 'rau': 0, 're': 0, 'rea': 0, 'ri': 0, 'rii': 0, 'ro': 0, 'ru': 0, 's': 0, 'sa': 0, 'saa': 0, 'sae': 0, 'sau': 0, 'se': 0, 'sea': 0, 'si': 0, 'sii': 0, 'so': 0, 'su': 0, 't': 1, 'ta': 1, 'taa': 0, 'te': 0, 'tea': 0, 'th': 0, 'tha': 0, 'thaa': 0, 'the': 0, 'thi': 0, 'thii': 0, 'tho': 0, 'thu': 0, 'ti': 0, 'to': 0, 'tu': 0, 'u': 0, 'uu': 0, 'w': 0, 'wa': 0, 'waa': 0, 'we': 0, 'wi': 0, 'wo': 0, 'wu': 0, 'y': 0, 'ya': 0, 'yaa': 0, 'yau': 0, 'ye': 0, 'yi': 0, 'yo': 0, 'yu': 0, 'BNC_Corpus': 0, 'LexNorm_Corpus': 1}, 5: {'Words': 'kare', 'TermFreq': 2, 'a': 1, 'aa': 0, 'ae': 0, 'au': 0, 'b': 0, 'ba': 0, 'baa': 0, 'bae': 0, 'bau': 0, 'be': 0, 'bea': 0, 'bi': 0, 'bo': 0, 'bu': 0, 'buu': 0, 'ch': 0, 'cha': 0, 'chaa': 0, 'che': 0, 'chi': 0, 'cho': 0, 'chu': 0, 'd': 0, 'da': 0, 'daa': 0, 'dau': 0, 'de': 0, 'dea': 0, 'dh': 0, 'dha': 0, 'dhe': 0, 'dhi': 0, 'dhu': 0, 'di': 0, 'dii': 0, 'do': 0, 'du': 0, 'e': 1, 'ea': 0, 'g': 0, 'ga': 0, 'gaa': 0, 'gau': 0, 'ge': 0, 'gi': 0, 'go': 0, 'gu': 0, 'h': 0, 'ha': 0, 'haa': 0, 'he': 0, 'hea': 0, 'hi': 0, 'hii': 0, 'ho': 0, 'hu': 0, 'i': 0, 'ii': 0, 'j': 0, 'ja': 0, 'jaa': 0, 'je': 0, 'ji': 0, 'jo': 0, 'ju': 0, 'k': 1, 'ka': 1, 'kaa': 0, 'kau': 0, 'ke': 0, 'ki': 0, 'kii': 0, 'ko': 0, 'ku': 0, 'kuu': 0, 'l': 0, 'la': 0, 'laa': 0, 'le': 0, 'lea': 0, 'li': 0, 'lo': 0, 'lu': 0, 'm': 0, 'ma': 0, 'maa': 0, 'mae': 0, 'me': 0, 'mi': 0, 'mo': 0, 'mu': 0, 'muu': 0, 'n': 0, 'na': 0, 'naa': 0, 'ne': 0, 'nea': 0, 'ni': 0, 'no': 0, 'nu': 0, 'o': 0, 'oe': 0, 'r': 1, 'ra': 0, 'raa': 0, 'rae': 0, 'rau': 0, 're': 1, 'rea': 0, 'ri': 0, 'rii': 0, 'ro': 0, 'ru': 0, 's': 0, 'sa': 0, 'saa': 0, 'sae': 0, 'sau': 0, 'se': 0, 'sea': 0, 'si': 0, 'sii': 0, 'so': 0, 'su': 0, 't': 0, 'ta': 0, 'taa': 0, 'te': 0, 'tea': 0, 'th': 0, 'tha': 0, 'thaa': 0, 'the': 0, 'thi': 0, 'thii': 0, 'tho': 0, 'thu': 0, 'ti': 0, 'to': 0, 'tu': 0, 'u': 0, 'uu': 0, 'w': 0, 'wa': 0, 'waa': 0, 'we': 0, 'wi': 0, 'wo': 0, 'wu': 0, 'y': 0, 'ya': 0, 'yaa': 0, 'yau': 0, 'ye': 0, 'yi': 0, 'yo': 0, 'yu': 0, 'BNC_Corpus': 0, 'LexNorm_Corpus': 0}}