

UNIVERSITY OF MORATUWA

Faculty of Engineering



Team Helabasa

Sinhala Spell Checker and Corrector

FINAL YEAR PROJECT REPORT

Declaration

This Final Year Project Work entitled “Sinhala Spelling Checker and Corrector” is presented in fulfillment of the mandatory requirement for Bachelor of Science in Engineering specialized in Computer Science Engineering.

We hereby declare that this work is entirely done by us under the guidance of Prof. Gihan Dias, Senior Lecturer of Computer Science and Engineering Department, University of Moratuwa and the results embodied with this report has not been submitted to any other university or higher academic institute for the award of any degree or diploma.

Submitted by

120690G – Weerasinghe P.A.K.S

150185P – Geesara K.P.K

150211U – Hasoga G.H.C.S

Date

19/11/2019

Name of the Supervisor

Prof. Gihan Dias

Abstract

Sinhala is the native language of the Sinhalese people who make up the largest ethnic group of Sri Lanka. The language belongs to the globe-spanning language tree, Indo-European. However, due to poverty in both linguistic and economic capital, Sinhala, in the perspective of Natural Language Processing tools and research, remains a resource-poor language which has neither the economic drive its cousin English has nor the sheer push of the law of numbers a language such as Chinese has. Due to the morphological richness of the language, very limited number of research has been done on the subject of spell checking of Sinhala language.

As a solution to that we have introduced a spell checker which incorporates morphological analyzers which has the ability to generate various inflected words for a particular root word. Apart from the words generated from the morphological analyzer, words from various other resources have been added to the spell checker to increase the accuracy of the speller. A machine learning based model called morfessor has been integrated to check the potential correct words due to the addition of new inflections to the root words. For the suggestion generation module a fasttext based methodology has been followed in order to reduce the time consumption in generating suggestions. An error model which incorporates minimum edit distance, frequency of the words and type of spelling error has been used to get the best suggestion for the user. Spell checker could achieve 96.9% accuracy while the suggestion generation module could achieve 80.53% accuracy for the first suggestion.

Dedication

This project is dedicated to the people of Sri Lanka who are giving contributions to our education from grade 1 to undergraduate studies by paying taxes throughout the hard times. Next, we dedicate this project to the government who provided funds for free education. Next, we dedicate this project to the Department of Computer Science and Engineering, University of Moratuwa which is conducting this priceless course and guiding us throughout our university career wonderfully with great passion to create people to make the world a better place. Finally, we are dedicating this project to our parents who supported us throughout our whole life with much love.

Acknowledgment

The work we present here is the result of one year long tireless effort. However, this would not get success without many other individuals and organizations who supported us throughout this work.

First of all, we would like to extend heartfelt gratitude to our project supervisor Prof. Gihan Dias. We are like to highly appreciate the consistent support and guidance given by Prof. Gihan Dias throughout the entire project. The support given to this project by sharing ideas and knowledge for us is priceless. The guidance provided us into the academic research field by sharing his experience with us would always be remembered and cherished throughout our academic career.

Next, we would like to show our gratitude to the Department of Computer Science and Engineering, University of Moratuwa, for creating the platform and providing the opportunity for such a cause. We were able to maintain focus on this cause without any disturbance, because of all the facilities including the final year project lab provided by the department. It created an encouraging atmosphere within the department. So, our work became easier.

Access to necessary but exclusive research material was readily satisfied by the Center for Information Technology Services (CITeS) of the University of Moratuwa. Special thanks go to Assistant network manager of CITeS, Mr. Pasan Ravinatha, who helped us whenever we got a problem in the remote server.

Last but not least, we are grateful to Dr. Surangika Ranathunga who provided us with constructive feedback on our research during the project proposal and mid evaluation sessions.

Table of Content

UNIVERSITY OF MORATUWA	0
Faculty of Engineering	0
Team Helabasa	0
Declaration	1
Abstract	2
Dedication	3
Acknowledgment	4
Table of Content	5
1. Introduction	7
1.2. Natural Language Processing (NLP)	7
1.3. Morphological Analysis	8
1.3.1. Morfessor	8
1.4. Sinhala Spelling Errors	9
1.5. Word embeddings	9
2. Research Goals	10
2.1. Problem	10
2.2. Motivation	10
2.3. Research Objectives	11
3. Literature Review & Theoretical Background	11
3.1 Spell Checker	11
3.1.1 Today's trends	12
3.1.1.1 Three Major Approaches	12
3.1.1.1.1 Data-Driven Approaches	12
3.1.1.1.2 Machine learning-based approach	14
3.1.1.1.3 Morphology/Dictionary-based approaches	16
3.1.2 Morphological Analysis with FST	17
3.1.2.1 FOMA	18
3.1.2.2 Defining the Golden Standard Definitions for the Morphology of Sinhala Words	19
3.1.2.3 Morphological analysis with Machine learning	20
3.2 Suggestion Generator	21
3.2.1 Word Embeddings	21
3.2.1.1 Word2Vec	22
3.2.1.1.1 CBOW	22
3.2.1.1.2 Skipgram	24

3.2.1.2 Fasttext	25
3.2.1.2.1 Subword Model	26
3.2.1.2.2 Word Similarity for OOV Words	27
3.2.1.2.3 Effect of the size of the N-grams	27
3.2.1.3 PyFasttext	28
3.2.2 Similarity Measures	29
3.2.2.1 Euclidean Similarity	29
3.2.2.2 Cosine Similarity	29
4. Methodology & Results Evaluation	30
4.1. Methodology	30
4.1.1. Word pre-processing model	30
3.1.1.1 Word tokenizer	32
3.1.1.2 Unicode cleaner	33
3.1.1.2 Word formatter	36
4.1.2. Noun Morphological Analyser	38
4.1.3. Root Extractor	39
4.1.4. Spelling Error Detection	40
4.1.5.1 Using Morfessor to identify words not listed in the dictionary.	42
4.1.6. Suggestion Generation Module	43
4.1.6.1 Fasttext Model	44
4.1.6.1.1 Data Preprocessing	45
4.1.6.1.2 Model Selection	45
4.1.6.1.3 Model Training	45
4.1.6.2 Permutation Generation module	47
4.1.6.3 Split word generator module	48
4.1.6.4 Error model	49
4.1.6.4.1 Minimum Edit Distance calculation module	51
4.1.6.4.2 Scoring Mechanism	52
4.1.7 Results and Evaluation	53
4.1.7.1 Spell Checking Evaluation	53
4.1.7.2 Suggestion Generator's Evaluation	54
4.1.8 Web Application	56
5. Conclusion	57
6. References	60

1. Introduction

1.1. Background

Spell checking applications are important parts of several fundamental applications that are used in day-to-day activities. Spell checkers are a well-known application of word processing applications. Apart from that, spell checkers are widely used in applications like Machine Translation Systems, Automatic Speech Recognition Systems, Optical Character Recognitions Systems and Text to Speech Systems. Due to the importance of Spell Checking Applications in the real world, many types of research have been conducted in this area.

The language we are focusing on is the Sinhala Language which is one of the official languages of Sri Lanka and is spoken by roughly around seventeen million people. Sinhala is also spoken as a second language by other ethnic groups in Sri Lanka, totaling about four million. Sinhala is an Indo-Aryan language which is a branch of Indo-European languages. As most of the Indo-Aryan languages, Sinhala is also a morphologically rich language. It is a syllabic script with its own writing system which is an offspring of the Brahmi script.

1.2. Natural Language Processing (NLP)

Natural Language processing is one of the most talked about areas in current technology industry. It is mainly concerned with human languages and computers. This field mainly focuses on how to program computers to analyse and interpret human language data. Natural language processing has become a necessity in the current trends of AI. Computers are always good at analysing and interpreting statistical data. But current researches are conducted to program computers to understand human languages. Spell checking is also a main subfield in Natural language processing. Although there has been a significant research on spell checking in languages like english, there are very limited research done on languages like Sinhala because of the limited resources available.

1.3. Morphological Analysis

In linguistics, morphology is the study of words in a language and finding relationships among different words in the same language. Different languages have different levels of morphological richness. Morphological analysis means analysing a word in language using its morphological information. Morphology analyses the structure of the words such as prefixes, suffixes, Case markers, Stems and root words.

Morphological analysis is a very important area in Natural language processing. Computers can understand human languages and words using the morphological information of the words and then it can build different relationships among words.

1.3.1. Morfessor

Morfessor is a probabilistic machine learning method to find the morphological segmentation from raw text data. It is developed by developed by the organizers of the Morpho Challenge competition. Current stable version of this algorithm is morfessor 2.0. The first version of Morfessor, called Morfessor Baseline, was developed by Creutz and Lagus (2002) its software implementation, Morfessor 1.0, released by Creutz and Lagus (2005b). Authors of the morfessor has claimed that the morfessor algorithm is language independent. We can use it to segment the stem and inflections of words in any language. The initial version has been developed using two approaches for unsupervised learning. The cost function of the first model has been derived from the Minimum Description Length principle from classical information theory while the cost function of the second method has been defined as the maximum likelihood of the data given the model (Creutz and Lagus, 2002).

Morfessor 2.0 is a new implementation of the Morfessor Baseline algorithm. It has been written in a modular manner and released as an open source project. Morfessor 2.0 also contains a Python library interface in order to be integrated in other python applications.

1.4. Sinhala Spelling Errors

In Sinhala language, basically, there are two main types of spelling errors named non-word where the word itself is incorrect and real-word errors where the word is correct but invalid in the context. Non-word errors can be categorized as substitution, insertion, deletion shuffling, etc. Substitution error type contains errors such as na-Na la-La dissension. Insertion error type means inserting more characters to the word while deletion means deleting characters from the word. Most of the already implemented Sinhala spell checkers have only focused on detecting only one or two categories of the above-mentioned errors.

1.5. Word embeddings

As the name suggests natural language processing means computationally processing natural languages to perform various tasks like machine translation, text to speech, spell checking etc. but computer are unable to understand the languages used by humans as it is because they can only interpret binary values. As a solution to that word embedding were introduced which is a mechanism of converting words into vectors which are more interpretable by computers.

Word embeddings are in fact a class of techniques where individual words are represented as real-valued vectors in a predefined vector space. Each word is mapped to one vector and the vector values are learned in a way that resembles a neural network, and hence the technique is often lumped into the field of deep learning. Key to the approach is the idea of using a dense distributed representation for each word.

Each word is represented by a real-valued vector, often tens or hundreds of dimensions. This is contrasted to the thousands or millions of dimensions required for sparse word representations, such as one-hot encoding. The distributed representation is learned based on the usage of words. This allows words that are used in similar ways to result in having similar representations, naturally capturing their meaning. This can be contrasted with the crisp but fragile representation in a bag of words model where, unless explicitly managed, different words have different representations, regardless of how they are used.

2. Research Goals

2.1. Problem

As Sinhala is the official and national language of Sri Lanka, most of the people are using Sinhala language in their day-to-day activities. Often the government ministries, departments and authorities mainly rely on Sinhala language to perform their daily tasks such as letter writing, preparing reports, communicating between employees. Although most of the current documents are hand written when converting those documents into soft copies well developed Sinhala Spell checker is much needed.

But when developing NLP applications like Spell checkers, resources are much more important. Research on languages like Sinhala have limited due the scarcity of resources. Resources like morphological analyzers and tagged corpora are very hard to find for Sinhala language. Over the course of time, languages tend to change in their phonetic, semantic, morphological or syntactic features. So due to the language changes continuous research should be done in order to keep the spell checkers upto date. As well as another difficulty the researchers are facing when dealing with Sinhala language is the problem of extracting data due to legacy fonts.

Currently developed spelling checkers are low on accuracy and they do not provide accurate suggestions. Our goal is to develop a spell checker which identifies mistakes accurately and provide better suggestions than existing spelling checkers.

2.2. Motivation

Although there are many spell-checker applications available for well-resourced languages like English, very little attention has been given to languages like Sinhala. One of the main obstacles of developing a spell checking engine for a language like Sinhala is the lack of resources. It is hard to find a tagged corpus for the Sinhala language. Developing a spell checking engine for a morphologically rich language like Sinhala is difficult. Due to the morphological richness, it is really difficult to develop a dictionary that covers all the possible inflections, derivations, and compounds obtainable from all roots. Most of the

currently developed Sinhala spell checking engines use a rule-based approach which has various limitations to reduce the size of the dictionary.

2.3. Research Objectives

Goal of this research is to develop a complete web application to

1. check the spelling of a given Sinhala document and
2. provide suggestions of correct words to the words that are identified as misspelled by the system.

The main objectives are achieved using the following sub objectives.

- Develop a complete noun morphology analyser for sinhala nouns using FOMA tool.
- Develop a Sinhala root words extractor.
- Generate a Sinhala dictionary of about 1 million spell corrected sinhala unique words
- Develop a Unicode cleaner for sinhala words.
- Develop a Sinhala word tokenizer based on regular expressions.
- Develop a Sinhala word formatter to convert all the Sinhala words into the same rendering format.
- Train a morfessor model for sinhala words for morpheme stemming and evaluate the model using a test set.
- Train a fasttext model to output suggestions for a given misspelled word.
- Develop an error model to filter out the suggestions and list them according to the closeness.
- Develop an interactive web application for spell checking using above resources.

3. Literature Review & Theoretical Background

3.1 Spell Checker

The computer-based spelling checker has quite a long history. In 1961, Les Earnest who was a former computer scientist did the first research about the spelling checker concept [22]. Ten years later, in 1971 Ralph Gorin, an undergraduate student, developed the first true spelling checker program for the English language. Since then, there have been numerous research projects in the field. Because of the researches and experiments carried out in this field,

several commercial and non-commercial spell checking engines have developed for past decades. But each spelling checker had different capabilities to detect and correct misspelled words accurately.

3.1.1 Today's trends

Today, spell checkers for English and European languages are well developed. Most of those spell checkers have exceeded accuracy rates of 90%. But, unfortunately, the accuracy of spelling checkers for most of the Indic languages such as Malayalam, Sinhala, and Tamil is still in the range of 75% ~ 85%. Because of that reason, plenty of researchers tended to do lots of research and experiments about how to create better spelling checkers for languages such as Sinhalese. Mostly, many researchers have identified major problems with Sinhala spellings and have directed their efforts to solve them effectively.

3.1.1.1 Three Major Approaches

When developing spelling checking engines, the approaches can be mainly categorized into three parts as Data-Driven approach, the Morphology/Dictionary based approach, and the Machine learning-based approach.

3.1.1.1.1 Data-Driven Approaches

A data-driven approach is one of the best approaches used in developing spelling checking mechanisms for various types of languages. Many researchers had come up with different techniques based on the data-driven approach which can be used in spelling checking effectively. Especially for a language like Sinhala or Tamil, this is a good way to develop a spelling checker.

There are some research works done by various researchers related to the data-driven approach. But, some of the researches have shown very good results for spelling checking in languages like Sinhala. In 2012, the researcher Asanka et. al. [1] developed a spelling checker named as “Subasa” for the Sinhala language. The main data-driven technique used in "Subasa" is the n-gram analysis method. Using this approach, "Subasa" identifies and corrects three main types of errors found in Sinhala sentences. Those three major error types can be identified as "retroflex and dental letter confusion", "the pronunciation and orthography of

the retroflex and palatal sibilants", and "pronunciation and orthography of aspirated and unaspirated consonants". The corpus received from the University of Colombo School of Computing (UCSC) had used to compute n-grams required to check spelling errors. When the spelling checker encounters an input text, first of all, the spelling checking algorithm tokenizes the input text into a unique word list. Then each unique word is compared with the corpus for further analysis. If the words are not found in the corpus then the words passed for further processing. If the words are found in the corpus, then the words removed from the unique word list. After that, a set of permutations are generated based on similar-sounding letters. Then, the set of permutations are used with the suggestion generator based on corpus oriented pre-defined unigram, bigram, and trigram frequencies. After that, for identifying best match spelling word for a misspelling word, the highest n-gram probability has used.

In the 2nd version of the “Subasa” spelling checker [2], the authors present a data-driven approach with slight modifications. In that paper, the authors have improved the above approach with the minimum edit distance technique. It gives the minimum number of transformations to convert a word to the target word. That transformation includes *insertion*, *deletion*, and *substitution*. Using this method, finding the acceptable string with a minimum number of transformation words from Sinhala corpus was very easy. In this approach, calculated possible permutation words select higher rank as the best solution and using minimum edit distance rule provided correction word list for the input word.

The above two data-driven methods are limited to check and correct only common types of spelling errors. But for checking and correcting spelling errors like context-based/contextual errors, the above-proposed methodologies are not good enough. In 2018, Buddhinee et. al. proposed a methodology [3] to develop a Sinhala Spelling checker that has abilities to detect both normal Sinhala spelling errors and contextual spelling errors. According to the proposed methodology, similar structure, similar-sounding letters, and minimum edit distance have used to check and correct spelling errors. When detecting the best suggestion for a given incorrectly spelled word, they have used context-based analysis instead of previously proposed methods. Due to context-based analysis, it gives a more accurate suggestions for incorrect words. As an example, consider a sentence "පාසල් විවෘත කඩඉම් ළමුන් හට නිකුත් කෙරේ.". In this sentence, "විවෘත" word is incorrect. Using the traditional

trigram analysis, their algorithm generates the words "විභාගය" and "විවිභාගය" as suggestions. After getting the suggestions, they introduce all the suggestions and the word: "භාසල්" before the incorrect word: "විබාගය" into the contextual analyzing algorithm to find the most suitable suggestion: "විභාගය" according to the Sinhala corpus that they have used. Using this set of advanced methods, their algorithm got an average precision of 84.73%, a recall of 83.69 % and an f-measure of 86.01% for the overall methodologies.

3.1.1.1.2 Machine learning-based approach

Machine learning is a highly advanced technology used in various areas of computer science and data science to solve some data-related problems. Nowadays, there is a trend to use machine learning models to solve problems cognitively (behaving like a man). Therefore, to solve spelling related problems faced in various languages, machine learning approaches are used. Because of the new trend, lots of researchers move their thinking along the path of machine learning to identify better solutions for both spelling checking and suggestion generating. In some cases, to develop the major parts of spell checking engines such as morphological analyzers, machine learning approaches are used.

The Chinese language is a complex language compared to many other languages all around the world. Therefore most of the spelling checkers developed for the Chinese language have lower accuracy than any regular spell checker has. To overcome this problem, a Chinese researcher has proposed a solution based on Machine Learning. It is possible to use an approach based on Machine Learning to develop a highly accurate spelling checker for languages like Sinhala. In this research paper [6] the author has proposed a well-trained model to detect spelling errors in the Chinese language. Also, he has managed to train his training model using a large raw corpus. Both Lexical based and statistical-based approaches have used in these Chinese spelling checkers to ensure not only more accuracy and efficiency but also language complexity reduction. Also, the author has especially stated the possibility of identifying and correcting spelling errors which are related to the context of the world using the pure-statistical based approach called Maximum Entropy.

Most of the Indo-Aryan languages like Sinhala have rich morphological rules. Therefore it is not suitable to follow traditional methodologies for constructing a morphological analyzer especially for languages like Sinhala. Because of that reason, some researchers have followed completely different methods for constructing morphological analyzers. In 2018, Premjith et. al. proposed a unique methodology [7] to develop a morphological analyzer for the Malayalam language. This methodology is completely based on the deep learning approach. According to the proposed methodology, first of all, morphemes have to be completely separated from the original word. Generating and separating morphemes from an original word is also known as sandhi splitting. This step is vital for language with rich morphologies, sandhi, and inflections. Due to sandhi found in original words, the majority of morphological changes occur at the conjoining position of morphemes. Because of these reasons, finding all the morpheme boundaries becomes a relatively tough task. To overcome these issues, the authors have provided a deep learning-based approach for learning the possible rule for identifying the morphemes automatically and segmenting them from the original word. To do that they developed three different systems based on Long Short-Term Memory (LSTM), Recurrent Neural Network (RNN), and Gated Recurrent Unit (GRU). Using the three systems, they have obtained accuracies 97.88%, 98.08%, and 98.16% respectively for the Malayalam language.

The degree of morphological richness and inflections related to original words are very similar to languages like Malayalam, Hindi, and Sinhala. Therefore, any effective approaches used to develop a spelling checker for any Indo-Aryan languages can be used in similar languages. In 2018, Sarooj et. al. proposed a neural network-based methodology [8] to develop a spell checker for Malayalam language. The spell checker has two main processes to handle both error detection and correction. The error detection section of the proposed model is based on the Long Short-Term Memory neural network that is well trained to identify misspelled words and the position of the misspelled word in a given sentence. The error detection accuracy is measured using a calculation mechanism called F1-score. For the error correction process, they used candidate word suggestions to select the most probable word.

3.1.1.1.3 Morphology/Dictionary-based approaches

The dictionary-based approach is the oldest way to develop spelling checkers for any language. The logic behind the dictionary-based spelling checkers is very simple. A set of root words and a set of related inflections are the main part of any dictionary-based spelling checker. Whenever the algorithm gets input text for checking spelling errors, it will look up the dictionary to find out whether there is a valid word or not for a given word.

Morphology based spelling checkers follow a similar type of methodology the same as dictionary-based spelling checker algorithms to check whether an input word is correct or not. Morphology-based spelling checkers use a set of morphological rules to check the correctness of any given input word. If the morphological analyzer can't find any possible morphological rules from the set of morphological rules define for a given language to check the correctness of the word then the spelling checker identifies the word as an incorrect word based on the set of morphological rules.

Most research works done in these two areas had shown that if both dictionaries and set of morphological rules are complete enough to deal with the majority (95%-100%) of words of a language, then these two approaches are considered as very good ways to develop spelling checkers. Also, some researchers have found some specific ways to improve overall accuracy and the performance of these spelling checking algorithms.

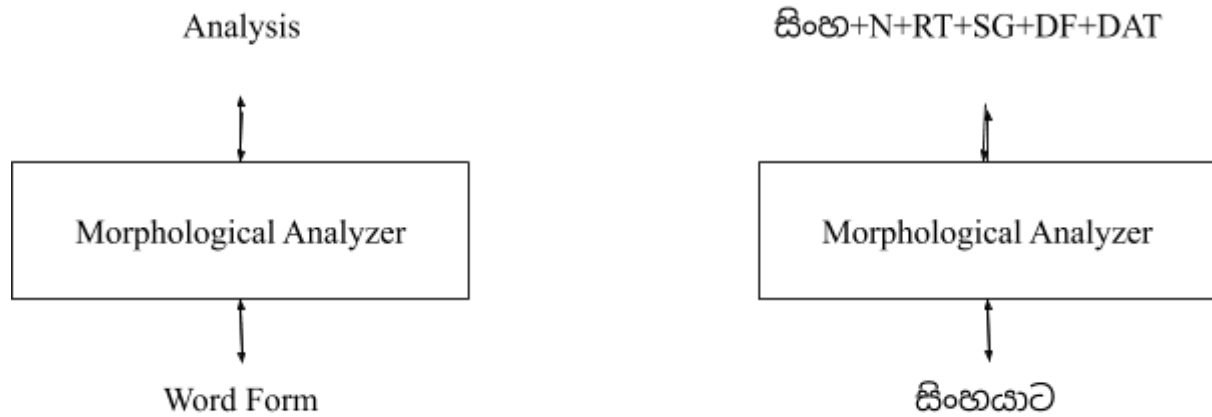
These days, spell checking engine designers don't use a single dictionary to check spelling errors due to performance reduction occurred when accessing a large single dictionary. Kaur et. al. have proposed a technique based on dictionary clustering for the Punjabi language to provide better speed and better performance in spelling error detection and correction. Most of the traditional dictionaries were based on morphological characteristics in the respective language. But In this approach [4], a dictionary is organized as clusters that have divided according to Punjabi language nouns, verbs, adjectives, and pronouns. These categories of each word of the input document identified at the stage of run time. After identifying the

category of the word, each word is searched in the predicted cluster. Because of the smaller size of clusters, spell checker processing time can be reduced.

Having rich morphological rules in a language may cause some difficulties while developing a morphological analyzer for the language. Since Sinhala is an Indo-Aryan language, there are well defined morphological rules. Therefore it is difficult to develop a morphological analyzer for languages like Sinhala. But, there are some research works which have provided some advanced methodologies to develop morphological analyzers for the languages similar to Sinhala. In this research paper [5], the authors have proposed a methodology to develop a spelling checker for the Telugu language. This methodology is based on morphological analysis and Sandhi splitting rules. It consists of two parts: a well-defined set of routines for scanning the submitted text document(*Morphological Analyzer and sandhi splitting rules*) and identifying valid words in the text, and an algorithm for comparing the unrecognized words and word parts in the text against a known and carefully-defined list of variant spelled words and word parts. The authors have tested the Morphological analyzer with a 3 million word length corpus. Also, they have covered 95%~97% of the whole corpus correctly. Therefore the proposed methodology is highly beneficial for the project like developing a spelling checker for languages like Sinhala.

3.1.2 Morphological Analysis with FST

Morphology [18] is the part of linguistics that deals with the study of words, their internal structure and partially their meanings. It refers to the identification of a word stem from a full word form. A morpheme in morphology is the smallest units that carry meaning and fulfill some grammatical function. Morphological Analysis is the process of providing grammatical information of a word given its suffix. A morphological analyzer implemented as a finite-state transducer can be considered as a BlackBox that translates word forms (such as runs) into a string that represents its morphological makeup, such as run+V+3p+Sg: a verb in the third person singular present tense.



Since finite-state transducers are inherently bidirectional devices, i.e. we can run a transducer in the inverse direction as well as the forward direction, the same FST, once we've built it, can serve both as a generator and an analyzer. The standard practice is to build morphological transducers so that the *input* (or domain) side is the analysis side, and the *output* (or range) side contains the word forms.

3.1.2.1 FOMA

Foma [19] is a compiler, programming language, and the C library for constructing finite-state automata and transducers. The library contains efficient implementations of all classical automata/transducer algorithms: determinization, minimization, epsilon-removal, composition, boolean operations.

It has specific support for many natural language processing applications such as producing morphological analyzers.

According to the documentation [21] FOMA supports features such as

- Xerox-compatible regular expression and scripting syntax (xfst/lexc)
- Python bindings
- Separate C API for constructing and handling automata
- Import/export from Xerox/AT&T/OpenFST tools
- Separate utility (flookup) for applying automata with various strategies

- Supports flag diacritics
- Contains functions for constraining reduplication (`_eq()`)

3.1.2.2 Defining the Golden Standard Definitions for the Morphology of Sinhala Words

Research conducted by the University of Colombo school of computing on sinhala morphology [14] shows how sinhala nouns and sinhala verbs can be classified on how these words are inflected and their morphological information.

According to this research sinhala nouns can be divided mainly into 26 groups as below.

Class	Root word Example
Masculine Front Vowel. Mid Vowel	ගව
Masculine Germinated Consonant	බල
Masculine Back Vowel	එළු
Masculine Retroflex- 1.1	කපුටු
Masculine Retroflex- 1.2	උතුමා
Masculine Retroflex- 2.1	කුමරා
Masculine Retroflex- 2.2	සහකරු
Masculine Consonant-1	මිනිස්
Masculine Consonant-2	හරක්
Masculine Consonant-3	ගිරා
Feminine Front Vowel.MidVowel	කුමරි
Feminine Back Vowel	ආයී

Feminine Consonant	මව්
Neuter Front Vowel.MidVowel	මේස
Neuter Germinated Consonant	කපු
Neuter Back Vowel	පුව
Neuter Retroflex- 1	සිරුරු
Neuter Retroflex- 2	ඉර
Neuter Consonant	ගස්
Neuter MidVowel	කඩ
Neuter Kinship 1	අක්කා
Neuter Kinship 2	ගුරුතුමා
Neuter Kinship 3	මල්ලි
Uncountable Consonant Ending	කාබන්
Uncountable Vowel Ending	සීනි
Irregular Animate	නෝනා

We used above information when developing our Noun morphology analyser by defining a FOMA class for each of the above classes.

3.1.2.3 Morphological analysis with Machine learning

A team of researchers at the University of Colombo school of computing has conducted research [15] and evaluation on how to use popular machine learning algorithm used in morpheme segmentation, Morfessor.

They have used the Morfessor Categories-MAP, version 0.9.2, (Creutz and Lagus, 2012) as the script to be evaluated. They have used two sets of different datasets to train morfessor model and evaluate the results.

1. Full distinct word list extracted from a 10 M corpus
2. Distinct list of words with frequency of more than 1

With some initial experiments they have identified that when the sinhala script is romanized it performs better. Therefore both the above datasets were romanized (turn to english script using some rules) before they are been used to train the morfessor modal.

They have previously done some research on morpheme segmentation of sinhala and have declared and created a dataset of sinhala roots and how the segmentation happens. (Defining the Golden Standard Definitions for the Morphology of Sinhala Words). They have used this dataset to evaluate the results generated from the morfessor algorithm. According to their research this algorithm performed better predicting noun stems than verbs. Also accuracy of the model which was trained using second data set has gained significant improvement on accuracy on stemming. Based on these facts, it can be concluded that the Morfessor algorithm performs well with lower numbers of more accurate data than large numbers of erroneous data, as would be expected of it.

Overall this morfessor algorithm has predicted stemming for more than 50% accurately according to this research.

3.2 Suggestion Generator

3.2.1 Word Embeddings

In very simplistic terms, Word Embeddings [20] are the texts converted into numbers and there may be different numerical representations of the same text. Machine Learning algorithms and almost all Deep Learning Architectures are incapable of processing strings or plain text in their raw form. They require numbers as inputs to perform any sort of job, be it classification, regression. A Word Embedding format generally tries to map a word using a dictionary to a vector. The different types of word embeddings can be broadly classified into two categories-

- Frequency-based Embedding
- Prediction based Embedding

Frequency-based embeddings are deterministic methods to determine word vectors. But these methods proved to be limited in their word representations. To overcome the limitations word2vec was introduced to the NLP community. These methods were prediction based in the sense that they provided probabilities to the words and proved to be state of the art for tasks like word analogies and word similarities.

3.2.1.1 Word2Vec

Word2Vec is an approach that helps us to achieve similar vectors for similar words. Words that are related to each other are mapped to points that are closer to each other in a high dimensional space. Word2vec model predicts a word by using its neighbors, by learning dense vectors called embedding. Word2Vec builds on the fact that words that share similar contexts also share semantic meanings. Word2vec is available in two flavors: the CBOW model and the skip-gram model. Skip-gram: When surrounding words(also known as context words) are predicted using the input(target) word. CBOW(Continuous Bag Of Words): When the target word is predicted using the surrounding words(context words).

3.2.1.1.1 CBOW

The way CBOW [20] work is that it tends to predict the probability of a word given a context. A context may be a single word or a group of words. Consider the example: *Have a great day.*

Let the input to the Neural Network be the word, *great*. The CBOW based model tries to predict a target word (*day*) using a single context input word *great*. More specifically, the one-hot encoding of the input word and measure the output error compared to one-hot encoding of the target word (*day*).

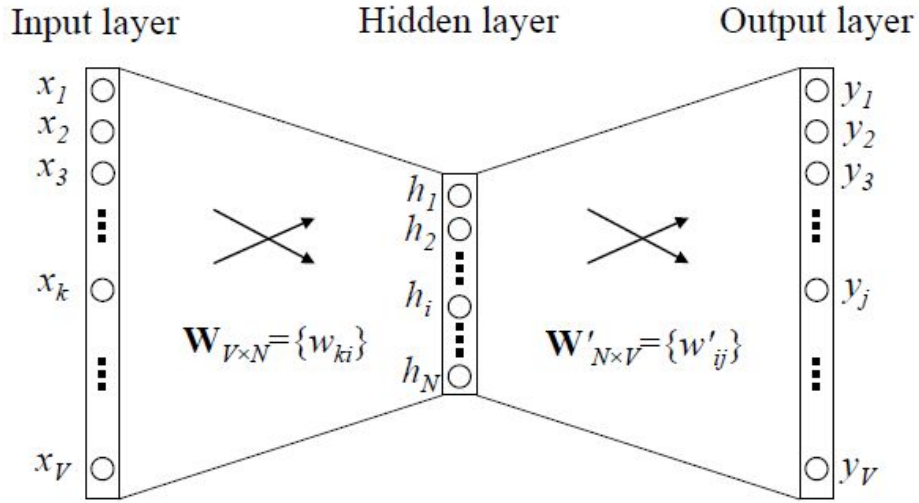
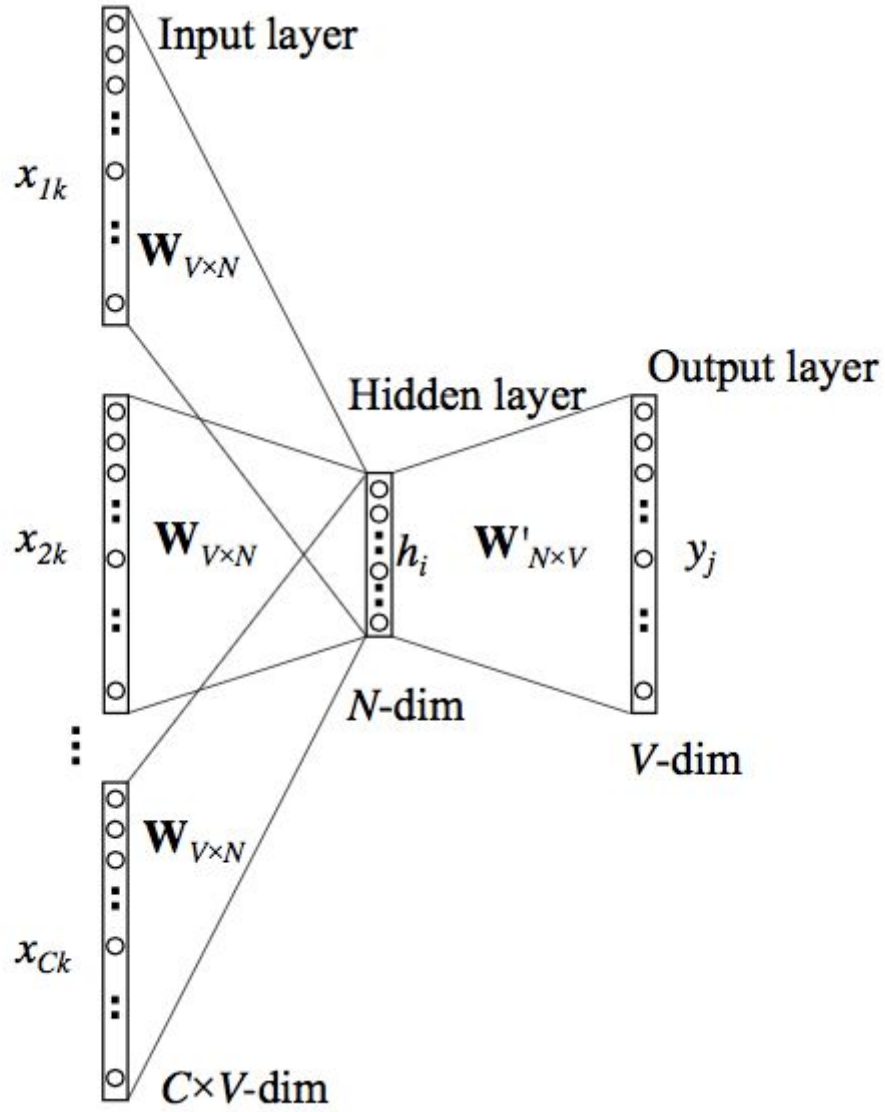


Figure 1: A simple CBOW model with only one word in the context

The input or the context word is a one-hot encoded vector of size V . The hidden layer contains N neurons and the output is again a V length vector with the elements being the softmax values.

The hidden layer neurons just copy the weighted sum of inputs to the next layer. There is no activation like sigmoid, tanh or ReLU. The only non-linearity is the softmax calculations in the output layer.

But, the above model used a single context word to predict the target. Multiple context words to do the same.



The above model takes C context words. When $W_{V \times N}$ is used to calculate hidden layer inputs, we take an average over all these C context word inputs.

3.2.1.1.2 Skipgram

Skip – gram [20] follows the same topology as of CBOW. It just flips CBOW's architecture on its head. The aim of skip-gram is to predict the context given a word. The input vector for skip-gram is going to be similar to a 1-context CBOW model. Also, the calculations up to hidden layer activations are going to be the same. The difference will be in the target variable. Two separate errors are calculated with respect to the two target variables and the two error vectors obtained are added element-wise to obtain a final error vector which is propagated back to update the weights. The weights between the input and the hidden layer

are taken as the word vector representation after training. The loss function or the objective is of the same type as the CBOW model.

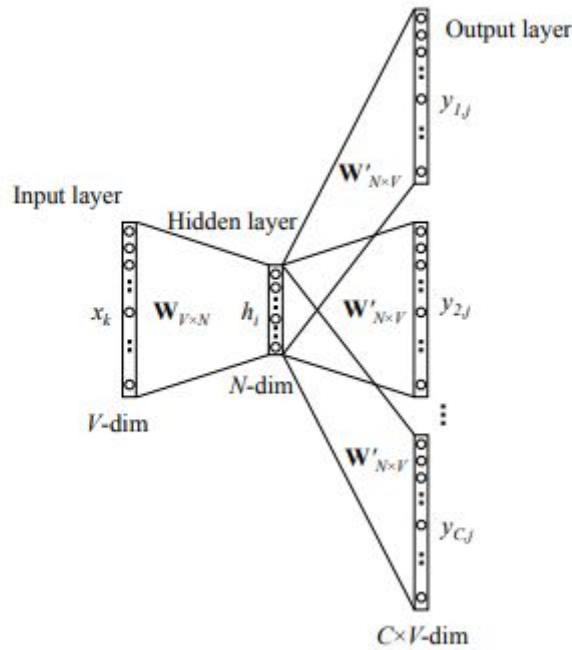


Figure 3: The skip-gram model.

As an example: given the sentence *'Poets have been mysteriously silent on the subject of cheese'* and the target word *'silent'*, a skip-gram model tries to predict the target using a random close-by word, like *'subject'* or *'mysteriously'*. The cbow model takes all the words in a surrounding window, like $\{been, mysteriously, on, the\}$, and uses the sum of their vectors to predict the target.

skip-gram models work better with subword information than cbow which is more useful in generating syntactically similar words.

3.2.1.2 Fasttext

FastText [17] is an open-source, free, lightweight library that allows users to learn text representations and text classifiers which works on standard, generic hardware. Fasttext is developed and maintained by Facebook which currently contains about 157 pre-trained language models including Sinhala. Fasttext models mainly used for text classifications and word representations which incorporates sophisticated Natural Language Processing concepts such as using Subword information for enriching word vectors. One of the unique features of

Fasttext is the ability to generate word vectors for the out of vocabulary words. Fasttext models have this ability as they incorporate n-grams information when generating a word vector. As an example, when we consider the word '*water*' with $n=3$, fasttext representation of character n-gram is $\langle wa, wat, at, ate, te, ter, er \rangle$. So when generating the final word vector for the word all the character n-grams are used with the given word '*water*' itself. So when an OOV word is given to the model, word vector can be generated using the character n-gram information of the word. In contrast, Word2Vec models consider each word in the trained corpus as atomic element and only generate vectors for words in the corpus. Fasttext models can be trained using both supervised and unsupervised algorithms to generate word vectors according to the use-case. As well as it provides both Skip-gram and CBOW algorithms in generating word embeddings.

3.2.1.2.1 Subword Model

Most of the word embedding models propose techniques to represent each word of the trained corpus as distinct vectors without sharing parameters among the vectors. In particular, they ignore the internal structure of words, which is an important limitation for morphologically rich languages, such as Sinhala. These languages contain many word forms that occur rarely (or not at all) in the training corpus, making it difficult to learn good word representations. Because many word formations follow rules, it is possible to improve vector representations for morphologically rich languages by using character level information.

In the paper [9], researchers propose a method to learn representations for character n-grams, and to represent words as the sum of the n-gram vectors. They have introduced an extension of continuous skip-gram model which is incorporated with the fasttext model which takes into account subword information.

Apart from the skip-gram model which was described earlier which uses distinct word vectors to predict the context, the proposed model proposes a different scoring mechanism which takes the subword information into account. Each word w is represented as a bag of character n-gram. Special boundary symbols \langle and \rangle are added at the beginning and end of words, allowing to distinguish prefixes and suffixes from other character sequences. As well as the word w itself is included in the set of its n-grams, to learn a representation for each

word (in addition to character n-grams). Taking the word *where* and $n = 3$ as an example, it will be represented by the character n-grams: $\langle wh, whe, her, ere, re \rangle$ and the special sequence $\langle where \rangle$. Consider a dictionary of ngrams of size G . Given a word w , denoted by $G_w \subset \{1, \dots, G\}$ the set of n-grams appearing in w . A vector representation \mathbf{z}_g is associated to each n-gram g . A word is represented by the sum of the vector representations of its n-grams.

$$s(w, c) = \sum_{g \in G_w} \mathbf{z}_g^T \mathbf{v}_c.$$

This simple model allows sharing the representations across words, thus allowing to learn reliable representation for rare words. In order to bound the memory requirements of the model, hashing function that maps n-grams to integers in 1 to K is used. Character sequences is hashed using the Fowler-Noll-Vo hashing function

3.2.1.2.2 Word Similarity for OOV Words

Fasttext model is capable of building word vectors for words that do not appear in the training set. For such words, the model simply average the vector representation of its n-grams. In order to assess the quality of these representations, the model analyze which of the n-grams match best for OOV words. For each pair of words, the model display the cosine similarity between each pair of n-grams that appear in the words.

3.2.1.2.3 Effect of the size of the N-grams

According to the research paper [9] proposes model in fasttext relies on the use of character n-grams to represent words as vectors. According to the research paper the choice of the range of the character n-grams is arbitrary and depends on the language. As mentioned in the research paper for both English and German, our arbitrary choice of 3-6 was a reasonable decision, as it provides satisfactory performance across languages.

	2	3	4	5	6
2	59	55	56	59	60
3		60	58	60	62
4			62	62	63
5				64	64
6					65

(b) DE Semantic

	2	3	4	5	6
2	45	50	53	54	55
3		51	55	55	56
4			54	56	56
5				56	56
6					54

(c) DE Syntactic

	2	3	4	5	6
2	78	76	75	76	76
3		78	77	78	77
4			79	79	79
5				80	79
6					80

(e) EN Semantic

	2	3	4	5	6
2	70	71	73	74	73
3		72	74	75	74
4			74	75	75
5				74	74
6					72

(f) EN Syntactic

According to the research paper it is important to include longer character n-gram sequences to get the best results. The longer the character sequences better it learns the similarity measures between the words either syntactically or semantically.

3.2.1.3 PyFasttext

A python binding for fasttext which can be used to access fasttext models.

PyFasttext provides following functionalities.

- Word representation learning using skipgram and cbow
- Word vectors
 - Access word vectors
 - Calculate the word similarity
 - Get the most similar words to a given word

```
>>> model.nearest_neighbors('dog', k=2)
[('dogs', 0.7843924736976624), ('cat', 75596606254577637)]
```

- Word analogies

```
>>> model.most_similar(positive=['woman', 'king'], negative=['man'], k=1)
[('queen', 0.77121970653533936)]
```

- Text classification with supervised learning
 - Prediction
 - Labels and probabilities

3.2.2 Similarity Measures

When the fasttext model is trained using a corpus, it learns various features of words using the subword information. The words which have closer syntactic or semantic similarity are clustered in the vector space. There are various measurements to get the nearest vectors for a given vector.

3.2.2.1 Euclidean Similarity

The euclidean distance between vectors X and Y is defined as follows.

$$d(x, y) = \sqrt{\sum_i^n (x_i - y_i)^2}$$

In other words, euclidean distance is the square root of the sum of squared differences between corresponding elements of the two vectors. Euclidean distance is only appropriate for data measured on the same scale. One of the main disadvantages of euclidean similarity is curse of dimensionality.

3.2.2.2 Cosine Similarity

Cosine similarity calculates similarity by measuring the cosine of angle between two vectors. The equation is as follows.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

Mathematically, cosine similarity is a measure of the similarity between two non-zero vectors of an interior product space that measures the cosine of the angle between them. The cosine of 0 ° is 1 and is smaller than 1 for every angle in the interval (0, π] radians. So it is an

orientation judgment and not a size judgment: Two vectors with the same orientation have a cosine similarity of 1 two by 90 ° aligned vectors have a similarity of 0 and two diametrically opposite vectors have a similarity of -1, regardless of their size. Unlike euclidean similarity dimensionality of the vators is not a problem with the cosine similarity.

4. Methodology & Results Evaluation

4.1. Methodology

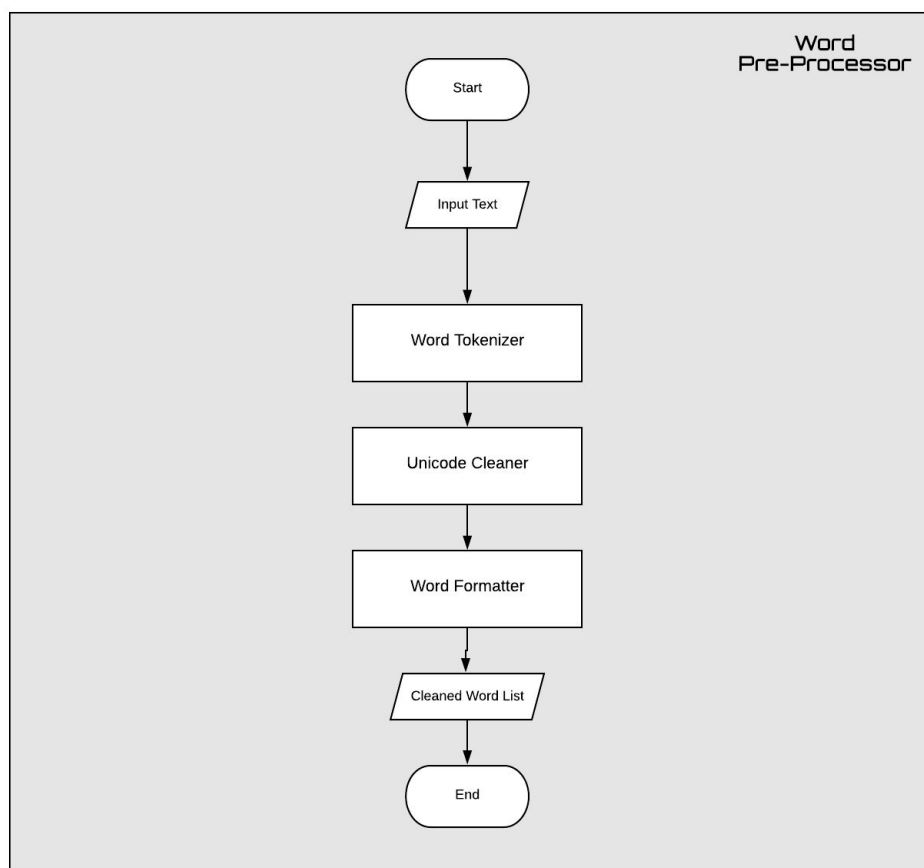
The main intention of this project is to develop an application to dynamically identify the spelling errors in a document written in Sinhala and correct those spelling mistakes. All the spelling error types identified and mentioned above in the report are identified by the system and suggestions for those misspelled words are shown to the user. Users can select the correct match from the suggested correct words. Suggestions are shown in the order as the closest correct word is the first word in the suggestion list. Each misspelled word is underlined in red color. Users can click on that word and get the relevant suggestions. After the correct suggestion replaces the misspelled word, that word will be underlined by a green color.

4.1.1. Word pre-processing model

When someone writes Sinhala text using a word processing application, he or she might accidentally create Unicode errors and formatting errors. For example, if someone wants to erase a part of a Sinhala word then he or she can press the backspace key on the keyboard. This backspace key pressing most of the time let zero-width joiner (invisible word joining character) remain in unwanted places in the text causing Unicode error. So, it is vital to have a word pre-processor for our spell checker to correct these types of errors. Otherwise, both spelling error detection and correction processes will be completely wrong.

Therefore, the input text is pre-processed before feeding it to the system. In the pre-processing module, the system tokenizes the text for building a list containing unique Sinhala words. This process removes spaces, punctuation marks and other unnecessary symbols in the input text. Also, Unicode errors are identified and fixed in this step using a

Unicode cleaner. According to the Sinhala Unicode character alphabet, it is possible to write some Sinhala words in two different ways. When the words (written in two ways) are rendered, both words are visually identical. But having multiple Unicode character sequences for a single Sinhala word will cause problems in both the spell checking process and the suggestion generating process. Therefore, a word formatter is implemented inside the pre-processing model. Also, the Unicode cleaning process and word formatting process are done not only for the input text but also for the data used to create the suggestions generator and the dictionary. So, it helps to prevent mismatching between input text and the data used in the suggestions generator and the dictionary.



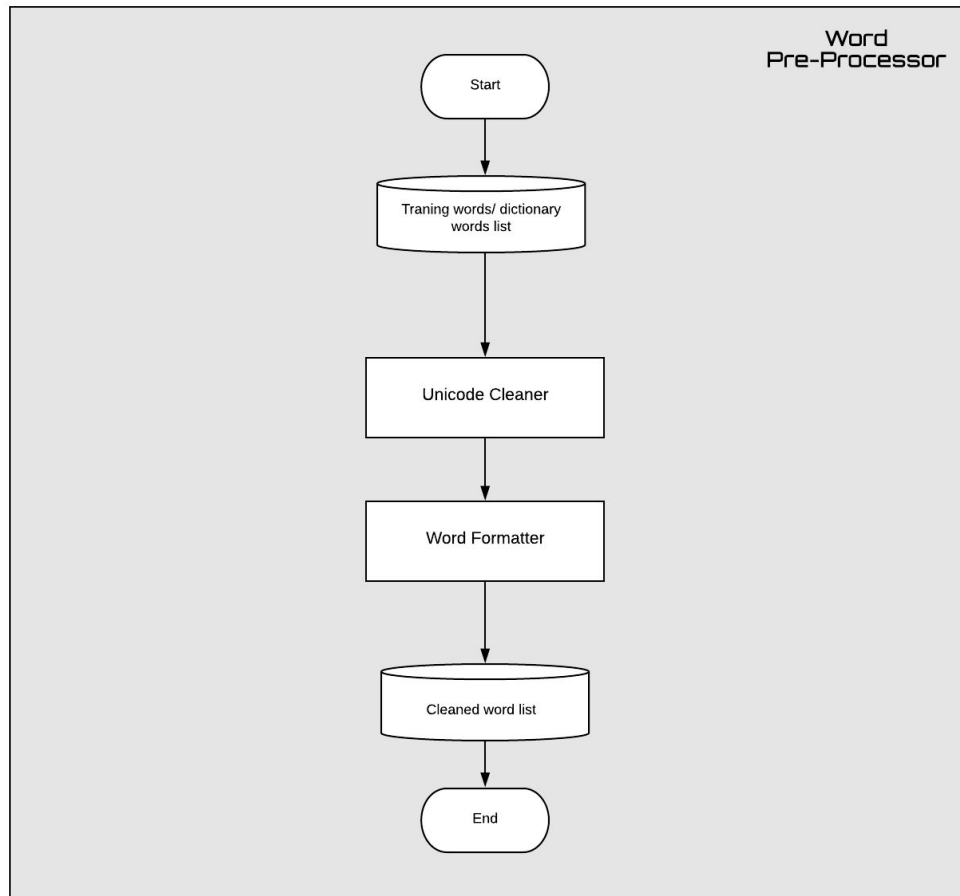


Fig: Pre-process data used in dictionary and training data

3.1.1.1 Word tokenizer

Tokenization is the process of breaking a stream of text up into words, paragraphs, symbols or other meaningful elements called tokens. The list of tokens becomes input for further processing such as parsing or text mining. Before feeding to the spell checker, an input text should be split into separate Sinhala Words. Therefore, a Sinhala word tokenizer is included in the word pre-processing unit. So, specially designed regular expression rules which are based on Sinhala Unicode range were used to develop the tokenizer for our spell checker. The main regular expression used in the tokenizer and how do the Sinhala words (tokens) split using tokenizer are shown below.

```
"/\s*(?<start_part>(-|~|`|!|@|#|$|%|^|&|\*|\(|\)|{|}\|\\[|\\]|;|:|\"|'|<br><|,|\\.|>|\\?|\\|/|\\\\|\\||-|_|\\+|=)*)(?<sin_word>[\\u0d80-\\u0dfe\\u200d\\u200c\\u200b]+)(?<end_part>(~|`|!|@|#|$|%|^|&|\*|\(|\)|{|}\|\\[|\\]|;|:|\"|'|<br><|,|\\.|>|\\?|\\|/|\\\\|\\||-|_|\\+|=)*)\s*/g"
```

ශ්‍රී ලංකාවේ භක්වන විධායක abc ජනාධිපතිවරයා ලෙසින් කේරී පත් වී සිටින ගෝඨාභය
රාජපක්ෂ 123 මහතා අනුරාධපුරයේ දී නව පදවියේ දිවුරුම් දුන්නේ රටට නව බලාපොරොත්තුවක්
දැනී කරමිනි.

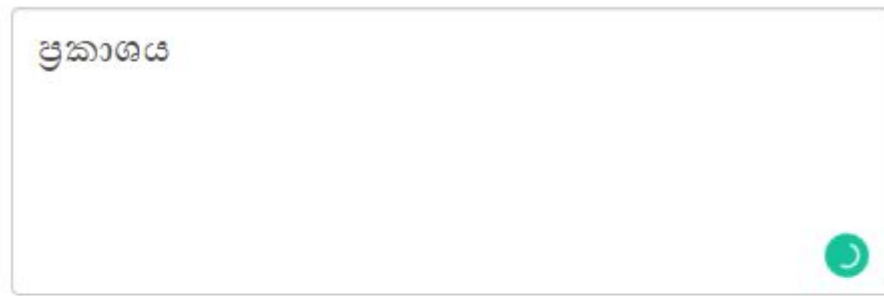
Match 24		
Full match	159-166	කරමිනි.
Group `start_part`	n/a	
Group `sin_word`	n/a	කරමිනි
Group `end_part`	n/a	.

“Sin_word” group always catch valid Sinhala words when tokenization process is happening. Therefore, all the valid Sinhala words are captured before feeding to spell checker.

3.1.1.2 Unicode cleaner

Unicode errors are one of the common types of errors found in any word text. These errors are not problematic when words are graphically rendered. But, for Natural Language Processing (NLP) purposes, having unicode errors in a word text causes huge problems. In most cases, Unicode errors are occurred due to having invisible character joiners such as zero width joiner (\u200d). These types of unicode characters helps to change the appearance of two or more visible characters into a combined version. For example, ප්‍රකාශය word can be shown like below.

Message



ପ୍ରକାଶଣ

Results



ପ ୀ ଣ କ ଶ ଣ ଣ

Fig: Presence of Zero width joiner(bule space) correctly in ପ୍ରକାଶଣ word

But, if a writer accidentally let these invisible word characters to be placed in the wrong places then the entire word will become a wrong word in character level. For example, ଫିଲ୍ମ word can be shown below. Characters of the ଫିଲ୍ମ word are correctly displayed in the first figure. But, in the second figure there is an unwanted zero width joiner(\u200d) present among other visible characters. Since having unwanted zero width joiners in a word doesn't cause any rendering issues, there is no any visible changes in both ଫିଲ୍ମ words. But both words are two different words according to character sequences. Therefore, it is necessary to clean these Unicode errors before feeding to spell checker or suggestion generator. Also, all the data used in the spell checker and the suggestion generator should be go through the Unicode cleaning process.

Message

අංශය




Results

අ ා ශ ය

Fig: Correct character sequence of අංශය word

අංශය



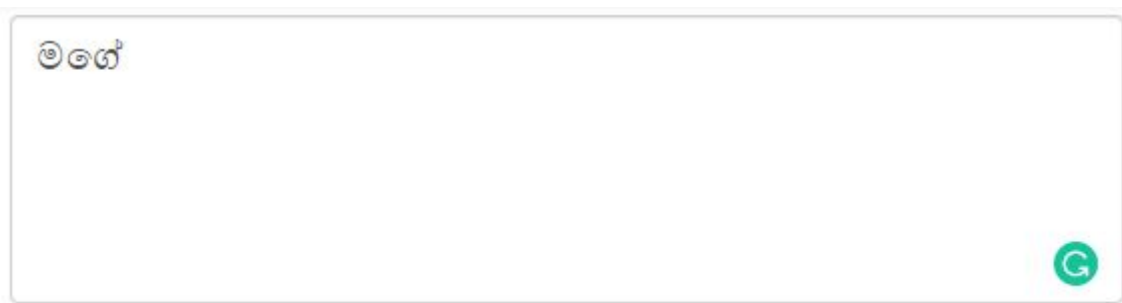
Results

අ ා ශ  ය

Fig: Incorrect character sequence of අංශය word

3.1.1.2 Word formatter

Word rendering and word writing are two different tasks done by the word rendering engine and the writer respectively. Sometimes two different character sequences might be rendered into a single word by the word rendering engine. This behavior is not an error at all because some people prefer different writing styles. But, Sinhala Unicode characters also have the same types of behaviors. Therefore it is necessary to convert all the words into a specific Unicode format before using data for creating Sinhala dictionary or checking spelling error of an input text. Otherwise, misformatted words will reduce the overall accuracy of the spell checker. The following figures show two different ways to write the word “මගේ”.



Results



Fig: Writing style of the word මගේ

මගේ

Results

මගේ

Fig: Another Writing style of the word මගේ

The word format used in the preprocessor changes character sequence of a Sinhala word according to the following rules. Also some word writing styles such as bonding words are converted into non-bonding word formats.

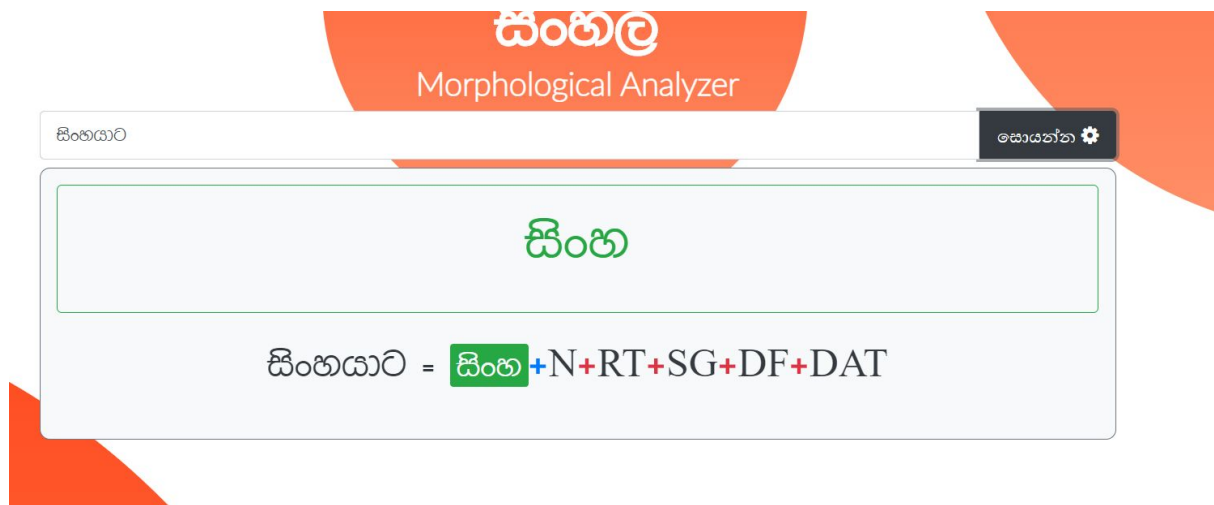
```
rules = [
    ['REPLACE', 'ේ', ['ෙ', 'ි']],
    ['REPLACE', 'ෙ', ['ෙ', 'ෙ']],
    ['REPLACE', 'ො', ['ෙ', 'ා']],
    ['REPLACE', 'ෝ', ['ෙ', 'ා', 'ි']],
    ['REPLACE', 'ෝ', ['ො', 'ි']],
    ['REPLACE', 'ෙ', ['ෙ', 'ෙ']],
    ['REPLACE', '෧෧', ['෧෧', '෧෧']]
]
```

Fig: Rules for converting word format

4.1.2. Noun Morphological Analyser

Then we developed a noun morphological analyzer component as a part of our project. Usage of this component is to morphologically analyze a given noun. When we input a noun into this component it will identify the root word of the given word and the inflected parts of the given word.

Ex- If we input the word **සිංහයාට**, This component will break the word into **සිංහ + යාට** and it will show the morphological details of this word like Singular-Plural, Definite-Indefinite, Case marker, etc.



We developed this morphological analyzer component using a C language library called FOMA which is mainly developed in helping develop language tools like morphological analyzers. In this, we use LEXC files and FOMA files to develop the application. In LEXC files we have to list root words that inflect in the same way in the same class and the rules (+N+RT+SG+DF+DAT: යාට#) that are used when the inflected part is joined with the root word (Ex- කොල්ලා, බල්ලා). In Sinhala Nouns we were able to find twenty-six noun classes as mentioned above based on the research Defining the Golden Standard Definitions for the Morphology of Sinhala Words. Below are those noun classes with the examples.

We found required data related to the prior research done and wrote FOMA language classes for each of the above classes. From that we were able to build a basic morphological analyzer for Sinhala nouns.

In the first step we were able to categorize 11,970 Sinhala noun root words into above 26 categories which generated over 500 000 Sinhala noun words with the inflected words.

Example Lexicon file is shown below.

```
!!!nouns.lexc!!!

Multichar_Symbols N +RT +SG +DF +NOM +CJ +NOM +ACC +ID +FN +VOC +PL +DAT +GEN +ABL
LEXICON Root
NounKinship3Stem;

LEXICON NounKinship3Stem

අක්කණ්ඩි NounKinship3;
අගමැති NounKinship3;
අගසවු NounKinship3;
අග්‍රාමාත්‍ය NounKinship3;
අග්‍රාමාත්‍ය NounKinship3;
අට්මස්ථානාධිපති NounKinship3;
අණකරු NounKinship3;
අදිකාරී NounKinship3;
අධ්‍යක්ෂක NounKinship3;
අධිපති NounKinship3;

LEXICON NounKinship3

+N+RT: #;
+N+RT+SG+DF+NOM: #;
+N+RT+SG+DF+ACC: #;
+N+RT+SG+VOC: #;
+N+RT+SG+DF+NOM+CJ:ක් #;
+N+RT+SG+DF+ACC+CJ:ක් #;
+N+RT+SG+DF+NOM+FN:ඹ #;
+N+RT+SG+DF+ACC+FN:ඹ #;
```

Then we wrote a python program to extract root words from sinhala corpuses. We thought if we could extract more root words that are not in basic morphology analyser we could improve the accuracy of the morphology analyser as well as the spell checker.

4.1.3. Root Extractor

Eleven thousand Sinhala noun word roots are not enough to cover the most widely used noun words in the Sinhala language. Therefore it is necessary to find out more Sinhala noun word roots for the noun morphological analyzer. To do that, 10 million words Sinhala tagged

corpus is used. First of all, all the unique words and their word frequency are extracted from the list of all noun-words in the tagged corpus. After that, all the unique words are sorted according to the Sinhala alphabet. Then inflection frequency for a given word is calculated using all the inflections found in the 29 FOMA noun classes and the sorted word list. Also, alongside the inflection frequency calculation process, new noun roots are extracted and classified according to the 29 FOMA classes. Then we manually checked the validity of the extracted roots using Sinhala word writing rules. The following figure shows the main code of the root extractor.

```

for word_key, word_value in word_array.items():
    if(count <=limit ):
        count+=1
        affixarr = []

        if(check_custom_affixes==1):
            unique_type = {'Noun-Fem-backVowel.lexc': {'frq': 0, 'prob': 0.0}, 'Noun-Fem-constant1.lexc':
                class_frequency=0

            print(str(count/limit*100.00)+"%")

            sword = word_value[0]
            sword_count = len(sword)
            affixarr.append(sword)
            first_char = sword[0]
            dor_count =0

            if (checked_word_list[word_key] !=0):
                next_word_range = { key:value for key, value in word_array.items() if key > word_key }
                root_count +=1
                afixer ={}
                for word_k,word_v in next_word_range.items():
                    dor_count+=1

                    if(dor_count<=dor_limit and sword_count>1):
                        if(word_k !=word_key ):
                            vword= word_v[0]

                            for affs in unique_aff[aff_class_name]:
                                affihghg = aff_generator(affs);

                                affs_char_size = len(affihghg)

```

Fig: Main code of the root extractor

4.1.4. Spelling Error Detection

At first, our idea was to feed each word into the morphological analyzer component after the preprocessing stage and the morphological analyzer validates each word one by one, which are fed into it. Then if any word cannot be morphologically analyzed correctly, this morphological analyzer component will output that word as an incorrect/misspelled word. But that method did not succeed as we thought because when each word is checked

dynamically in the web application, FOMA took considerable time to output whether that word was a valid one or not. For a spell-checking program time taken to check the spelling is very important. If a file with a large number of words Because of the large number of files used in foma (There were 26 class files only for Sinhala nouns), the speed of the spell-checking program reduced drastically. The reason for that is FOMA had to perform a lot of I/O operations to check whether a single word is valid. That IO operations (Opening and Closing a file) took some time which caused the whole program's performance to drop.

As a solution to this we generate all the inflected forms of the root words from the noun morphology analyzer and verb morphology analyzer, added other Sinhala words such as adjectives, adverbs and stopwords into that list and stored that wordlist in a python dictionary data structure Which contained close to 1M unique Sinhala words. Then we used a simple dictionary lookup method to check whether a given word is in that unique wordlist. For that, we used a python inbuilt method which checks if a given element is in the given dictionary and outputs a boolean value true or false in $O(1)$ time complexity. Using this method we were able to get the required speedup for spell checking. After identifying a given word as a misspelled word using the above method that word is passed on to the suggestion generation module to return the closest possible suggestions of correct words.



4.1.5.1 Using Morfessor to identify words not listed in the dictionary.

When evaluating the application after completing the spelling error detection part we identified that lots of inflected correct words were not identified as correct words because they were not in the dictionary. But when we analyse the dictionary we saw that root words of those inflected words which were not identified, were in the generated dictionary. When we researched and brainstormed about how to resolve this issue we came up with the idea of using morfessor algorithms in our project.

We created a dataset of 400k sinhala words and romanized that dataset (converting into english script) to train a morfessor model and tested with a test set that we created using golden standards for sinhala language. We got close to 70% accuracy from the morfessor model for stemming the root words. We included that morfessor model in our application because it will improve the accuracy and performance of the whole spell checking application.

Morfessor algorithm is applied as below, First we check if a given word to the application, is in the default dictionary. If that word is not in the dictionary we send that word into the morfessor model to break that word in to the root word and inflection (morpheme segmentation). Then we get the root word and check again to see whether the root word is in the dictionary. Also we developed another dictionary which contains all the inflections that can be applied to root words which contains close to 5000 inflections. After checking the root part of the segmented word we also check the inflected part of that segmented word in the inflections dictionary. If both the root part and the inflected part is in the relevant dictionaries we highlight that word in **yellow color** meaning that word might have some possibility of being a correct word (70% probability of being correct word according to our test set). Suggested words are also generated for these types of words.



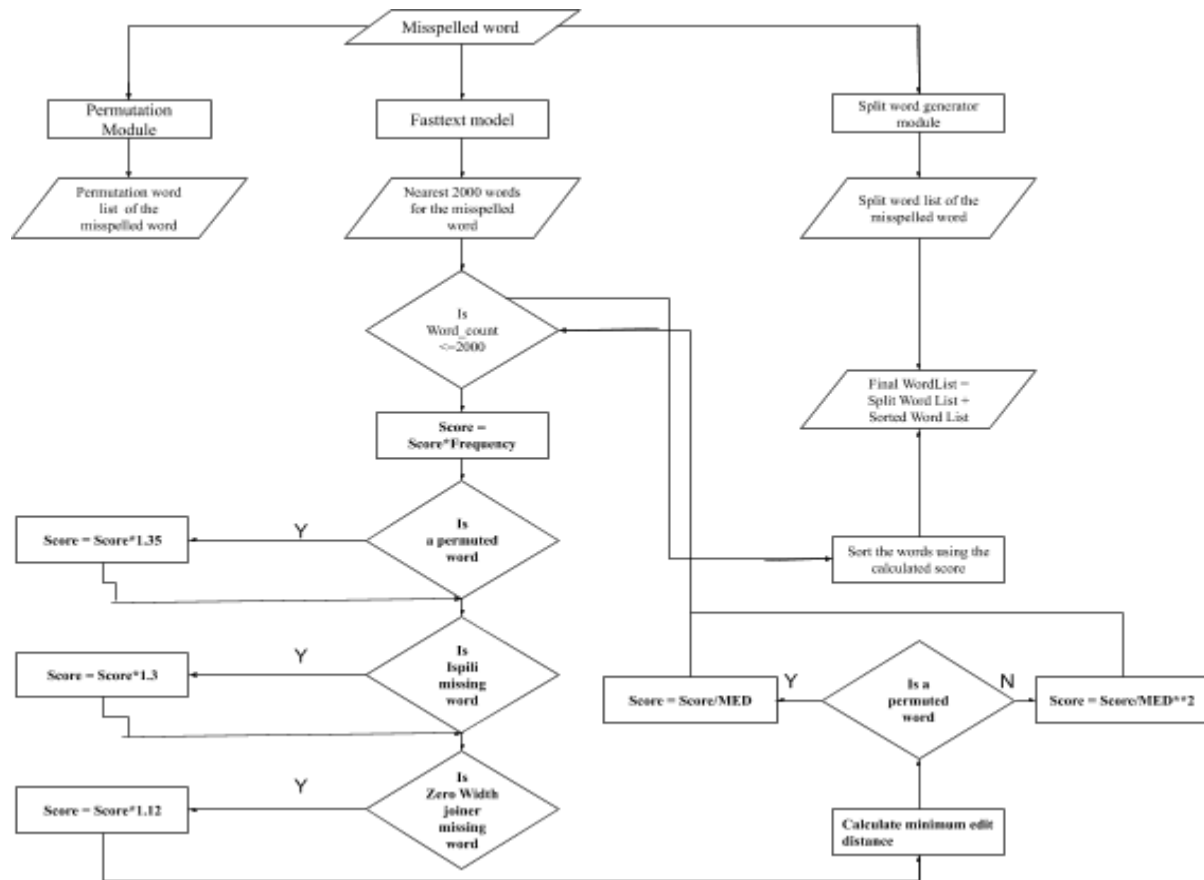
Word Type	Underlined Color	Suggestions Available
Incorrect	RED	YES
Corrected	GREEN	NO
Probably Correct	YELLOW	YES
Unicode Error	PURPLE	YES (Error fixed)

4.1.6. Suggestion Generation Module

So after a misspelled word is identified by the spell checker, we provide suggestions for the misspelled word. As mentioned in the related works, lots of previous applications have used dictionary lookup methods for this function. They have used minimum edit distance calculation methods to get the closest correct words for the misspelled word. But this method consumes a lot of time because, in this method, each word in the dictionary should be compared with their edit distances to the misspelled word.

Instead of this dictionary lookup method, we have trained a machine learning model that when we input the misspelled word, this model will output the closest suggestions for that

misspelled word. For this, we used a FastText model. The basic architecture of the suggestion generation module is as follows.



The suggestion generation module consists of 4 modules.

- Fasttext module which generates nearest 2000 words for a given misspelled word
- Permutation generation module which generates permutations of a given misspelled word
- Split word generator module which splits the given misspelled word if it can be split
- Error model which ranks the best suggestions for the user for a given misspelled word

4.1.6.1 Fasttext Model

Reasons for selecting fasttext model

- Ability to generate word vectors for out of vocabulary words(misspelled words can be out of vocabulary)
- Ability to calculate nearest neighbors quickly for a given word
- Use of subword information which can improve the syntactic similarity of the words

4.1.6.1.1 Data Preprocessing

First, we used a text corpus which we created by collecting data from Sinhala monolingual corpus available on WMT19 (World Machine Translation 2019). It contained approximately 110M words. Apart from the above corpus Tagged corpus available in the DMS also used to increase the word count. It also contained around 10M words. A dataset combining both the corpora was fed into the fasttext model. As the dataset contained unwanted characters such as numbers and special characters that are not useful in training the model we had to remove those characters from the dataset. For that, we used a preprocessing script to clean the dataset. After that, we used a word formatter to format the Sinhala words which are in different formats into one format. After that cleaned dataset was used to train the fasttext model.

4.1.6.1.2 Model Selection

According to fasttext there are 2 unsupervised algorithms that can be used to generate word representation given a corpus. Those 2 algorithms are namely CBOW and Skip-gram. As mentioned in the literature review each algorithm possesses a unique set of features that can be useful according to the use case as well as according to the used language. So, we wanted to select the algorithm for the Sinhala language to get the best word representation highlighting the syntactic similarity of the words.

According to the fasttext documentation out of the 2 algorithms Skip-gram algorithm performs better with subword information. Trained 2 models with default parameters using the 2 algorithms. After that evaluated the nearest neighbor output from the 2 models.

4.1.6.1.3 Model Training

Fasttext allows us to train our dataset in both supervised and unsupervised manners. With our dataset, we used both CBOW and Skip-gram algorithms which are unsupervised training

algorithms. We follow the documentation given by fasttext to fine-tune the parameters get the best word embeddings that can generate syntactically similar words.

Parameter	Related Property	Default Value
input	Path to the input file for training	-
output	Path to save the model	-
lr	Learning rate	0.05
dim	Size of the embedding vector	100
ws	Size of the context window	5
epoch	Number of epochs in model training	5
minCount	Minimal number of word occurrence	5
minn	Min length of char ngram	3
maxn	Max length of char ngram	6
neg	Number of negatives sampled	5
wordNgrams	Max length of word ngram	1
loss	Loss function type (ns, hs, softmax, ova)	ns
bucket	Number of buckets	2000000
thread	Number of threads	12

After that model was trained with some parameter tuning according to our dataset. We used different sizes of ngram information as we wanted to get the best pair which captures the syntactic information of the language.

Parameter	Changed Value
dim	300
minn	2
maxn	7

4.1.6.2 Permutation Generation module

Spelling errors caused due to phonetic similarity of the letters can be considered as one of the most common types of spelling errors in the Sinhala language. Similar sounding groups of letters are as follows.

{න,ණ} {ල,ළ} {ස,ශ,ෂ} {ත,ථ} {බ,භ} {ද,ධ} {ඳ,ද} {ට,ඨ} {ච,ඡ} {ජ} {ක,ඛ}
 {ග,ඝ} {ජ,ඤ} {ප,ඵ} {ච,ඪ}

When the misspelled word is fed into the module, it generates the permutations by searching the word for above similar-sounding letters and substituting them with their corresponding letters that belong to the same group. Among the generated words for a given misspelled word, there can be words with correct spellings and words with incorrect spellings. For example given the word බල්ලා, first, the module will generate the following words. බල්ලා, බළ්ලා, හල්ලා, බළ්ලා, හළ්ලා, හළ්ලා. These generated permutations will be filtered by the spell corrected dictionary and only the correctly spelled word will be returned as the final output of the model.

4.1.6.3 Split word generator module

Most of the time when users type something they tend to forget to insert a space between two words. Because of that, although the two words are correct separately the spell checker detects the word as an incorrect word due to missing space. So this module tries to split a given misspelled word into 2 correctly spelled words. As an example given the misspelled word අංගසම්පූර්ණඅකාර්යක්ෂම, the module should split it into 2 correctly spelled words as අංගසම්පූර්ණ and අකාර්යක්ෂම.

For that, the module should traverse through every word in the spell corrected dictionary which is a time-consuming task. So in order to speed up the process, the module uses nearest-neighbor output words from the fasttext model. As the fasttext model outputs syntactically similar words for a particular misspelled word the time consumption is very low compared to the previous approach.

The code snippet for the split checking module is as follows.

```
def split_check(words, word):
    first_word = []
    last_word = []
    space_suggestion = {}
    for i in words:
        if i[0] != '' and word != '' and i[0][0] != '' and i[0][0] == word[0] and i[0] in word:
            if get_hash(i[0]) in hashed_dic and get_hash_md5(i[0]) in spell_checking_dic:
                first_word.append(i[0])
        if i[0] != '' and word != '' and i[0][-1] != '' and i[0][-1] == word[-1] and i[0] in word:
            if get_hash(i[0]) in hashed_dic and get_hash_md5(i[0]) in spell_checking_dic:
                last_word.append(i[0])

    for i in first_word:
        for j in last_word:
            if i+j == word:
                s = i + ' ' + j
                space_suggestion[s]=1

    for i in range(1,len(word)):
        if word[-i:] in other_file and get_hash_md5(word[:-i]) in spell_checking_dic:
            space_suggestion[word[:-i] + ' ' + word[-i:]] = 2

    return space_suggestion
```

4.1.6.4 Error model

Although fasttext model outputs nearest neighbor words for a particular misspelled word, the most suitable suggestion is not highlighted always. That is because some of the spelling mistakes are more common than the other spelling mistakes in the Sinhala language. As an example, if we take a substitution error for the word බලල there is a higher probability that the error is caused due to the substituting ල for ඌ rather than substituting ම for ඌ. So higher probability should be given to such mistakes.

So the error model is responsible for scoring the nearest neighbor words suggested by the fasttext model to choose the best suggestions for the user. In order to identify the most common spelling mistakes, a test set of 457 words were chosen randomly and tried to identify the most common spelling mistakes in Sinhala language using those set of words. According to the dataset, the following were the key analyzing points that could be extracted.

Edit Distance	count	percentage		
1	312	68%	Insertion	80
			Deletion	7
			Substitution	225
2	100	21%	Insertion	25
			Deletion	20
			Substitution	55

3	22	0.04%	Insertion	7
			Deletion	2
			Substitution	13
4	22	0.04%	Insertion	10
			Deletion	2
			Substitution	10

Error	Count	Percentage
Nana Lala Error	148	32.2%
Ispili Papili Missing Error	135	29.5%
Joiner missing Error	58	12.5%

Nana Lala Error - Spelling mistakes happen due to phonetic similarity of the letters, Examples are බල්ලා for බල්ලා due to the phonetic similarity between ල and ඳ letters.

Ispili Papili Missing Error - Spelling mistakes happen due to missing a ispili, papili or alapili. Example of this type of error is මණ්ඩලය for මණ්ඩලය due to missing ඌ.

Joiner Missing Error - In Sinhala language, joiners are used to combine 2 characters to generate a single character. When the joiner is missing, the combined character is not

properly rendered. As an example, the character ଓ is rendered as ଓ̂ due to missing Zero Width Joiner(\u200d).

Used the above information to develop the scoring mechanism. The following are the steps followed to score the words given by the fasttext model.

4.1.6.4.1 Minimum Edit Distance calculation module

To calculate the minimum edit distance between 2 words Levenshtein minimum edit distance algorithm was used. The Levenshtein distance or edit distance between two strings is given by the minimum number of operations needed to transform one string into the other, where an operation is an insertion, deletion, or substitution of a single character.

Code snippet of the algorithm of the minimum edit distance calculating module.

```
def memoize(func):
    mem = {}

    def memoizer(*args, **kwargs):
        key = str(args) + str(kwargs)
        if key not in mem:
            mem[key] = func(*args, **kwargs)
        return mem[key]

    return memoizer

|
@call_counter
@memoize
def levenshtein(s, t): # Minimum edit distance algorithm
    if s == "":
        return len(t)
    if t == "":
        return len(s)
    if s[-1] == t[-1]:
        cost = 0
    else:
        cost = 1

    res = min([levenshtein(s[:-1], t) + 1,
               levenshtein(s, t[:-1]) + 1,
               levenshtein(s[:-1], t[:-1]) + cost])
    return res
```

In the algorithm for all the operations such as insertion, deletion and substitution, an equal cost value is assigned.

4.1.6.4.2 Scoring Mechanism

Step 01: Every word has a similarity score given by the fasttext model according to the cosine similarity between the misspelled word and the output word.

Score = score given by the fasttext model

Step 02: Multiply the Score by the frequency of the word. Frequencies of the words are extracted from the tagged corpus which has around 10 million words.

Score = Score*log(Frequency)

Step 03: If the word is in the permutation word list given by the permutation generator module multiply the current score by 1.35 (as 35% errors are permutation-based errors due to phonetic similarity of the letters).

Score = Score*1.35

Step 04: If the difference between the misspelled word and the fasttext output word is an Ispili or Papili, multiply the current score by 1.30 (as 30% of the errors are ispili-papili missing errors)

Score = Score*1.30

Step 05: If the zero-width joiner is missing in the misspelled word, multiply the current score by 1.12. (as 12% of the errors are missing zero-width joiner errors)

Score = Score*1.12

Step 06: Calculate the minimum edit distance between the fasttext output word and the misspelled word.

a) If the word is in the permuted word list

Score = Score/MED

b) If not

Score = Score/MED**2

Step 07: Loop through all the fasttext output words and sort out according to the scores of each word.

Step 08: If there are words returned from the split check module add them in front of the sorted list.

Step 09: Return the top 5 words to the user

4.1.7 Results and Evaluation

4.1.7.1 Spell Checking Evaluation

Out of 1000 Correct words

Spell Checker	No of Correct Words Identified as correct words	No of Correct words identified as incorrect words	True positive percentage (Accuracy)
Helabasa	969	31	96.9%
Microsoft Word	834	166	83.4%
Hunspell	794	206	79.4%
Subasa	920	80	92%

Out of 1000 Incorrect words

Spell Checker	No of Incorrect Words Identified as Incorrect words	No of Incorrect words identified as correct words	True Negative percentage (Accuracy)
Helabasa	910	90	91%
Microsoft Word	924	76	92.4%
Hunspell	855	145	85.5%
Subasa	472	528	47.2%

4.1.7.2 Suggestion Generator's Evaluation

There is no any standard lexicon or dataset to evaluate the accuracy of a suggestion generator for the Sinhala language. Therefore we used 200 words obtained from various documents containing difficult and commonly misspelled words as the baseline.

The first test was straight forward. Each entry was passed into the system and the output of the system was compared with the original entry.

To evaluate we used the Mean Reciprocal Rank(MRR) which is a statistical measure for evaluating any process that produces a list of possible responses to a sample of queries, ordered by the probability of correctness. The reciprocal rank of a query response is the multiplicative inverse of the rank of the first correct answer: 1 for first place, 1/2 for second place, 1/3 for third place and so on. The mean reciprocal rank is the average of the reciprocal ranks of results for a sample of queries.

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}.$$

Where rank_i refers to the rank position of the first relevant document for the i -th query.

For the second test set, used 100 misspelled words collected from Youtube and Lankadeepa comments and fed into the speller. For the misspelled words identified by the spell checker, the suggestions were generated and the Mean Reciprocal Rank was calculated.

	Number of Misspelled Words	Correctly classified	First suggestion accuracy	MRR
1	200	171	80.53%	0.8889
2	100	100	68%	0.7803

Model could produce high accuracy for the spelling mistakes occur due to the phonetic similarity of the letters. One of the main reasons for the reduction of the accuracy is because of the lack of correctly spelled words in the spell corrected dictionary. The accuracy of the

suggestion generator increases with the number of spell corrected words in the spell checking dictionary.

Most of the current baseline spell checkers for Sinhala language fail to produce suggestions for misspelled words which are combination of 2 correctly spelled words. But our suggestion generator has the ability to split up complex words such as follows.

In Helabasa Suggestion Generator: 74%

In Microsoft Word Spell Checker: 64%

After computing the accuracy of our model, to compare the effectiveness of our algorithm against the currently available algorithms used by popular applications, we performed the same test on Microsoft Word Sinhala Spell Checker, LibreOffice SpellChecker(Hunspell) and Subasa Spell Checker developed by UCSC.

For 200 misspelled words

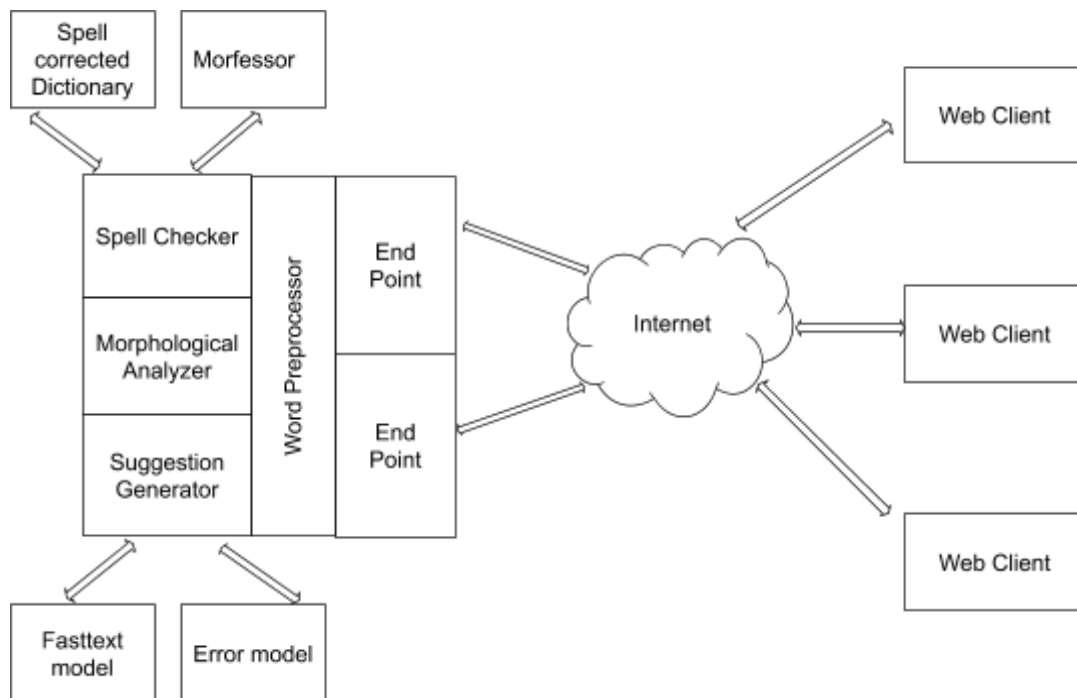
	Number of Misspelled Words	Correctly classified	First suggestion accuracy	MRR
Word	200	188	52.12%	0.5717
Hunspell	200	171	58.4%	0.6227
Subasa	200	94	77.65%	0.7765

For blog words

	Number of Misspelled Words	Correctly classified	First suggestion accuracy	MRR
Word	100	98	35.71%	0.4319
Hunspell	100	48	60.41%	0.7176
Subasa	100	38	73.68%	0.7368

4.1.8 Web Application

We have developed a web based application integrating the above mentioned modules as our final product. The following diagram illustrates the overall architecture of the web application.



According to the architectural system diagram, the overall system is based on multi-client single-server architecture. Also, asynchronous Javascript XMLHttpRequests are used to manage real-time client-server communication. The web application is not just a traditional spell checker. It is 100% interactive with the client. Whenever the client writes a new word in the application's text editor, the application sends AJAX requests to the server for the spell checking. After the spell checking processing inside the server, responses are sent back to the client-side. These requests contain a data object which includes all the data requires to pass to the client-side.

Based on the received data objects, the core script behind the spell checker perform the rest of the tasks to identify and highlight misspelled data. Also for the suggestion generation, similar

client-server communication are made totally based on the client's requests. The following table shows all the HTTP endpoints used in the spell checker.

Host URL(HURL):<http://helabasa.projects.uom.lk/morphy/>

Endpoint	Request Type	Request Parameters	Respond Data
HURL /foma/dicexecute	POST	[Param]: words Data Structure ex: [["සමාජභාවය", "සමාජභාවය", "අධිවාචනය", "අධිවාචනය", "නැවීම", "නැවීම"]]	Array of spell checked data Data Structure ex: [["සමාජභාවය", "සමාජභාවය", 1], ["අධිවාචනය", "අධිවාචනය", 0], ["නැවීම", "නැවීම", 1]]
HURL //fasttext/execute	GET	[Param]:word Data Structure ex: රසඥතාවයක්	Array of suggested word Data Structure ex: [["රසඥතාවයක්", 1], ["රසඥතාවයක්", 3.58351893845611], ["රසඥතාවයේ", 2.8903717578961645], ...]

5. Conclusion

Helabasa is a project that was focused on developing high accurate spell checking engine for the Sinhala language. The application is able to identify the misspelled words interactively while giving suggestions for the misspelled words. Over the past few years various attempts have been made to develop a well-covered spell checker for Sinhala language. Examples of such attempts are microsoft word which is commercialized application, hunspell which is an open source application and Subasa which was developed by UCSC. due to the morphological richness and the resource scarcity all of the above mentioned applications have fairly low accuracy in spell checking.

So as Sinhala is a morphologically rich language, as our first approach we followed a methodology based on the morphological analysis of the words. We developed a morphological analyzer using FOMA, and used the analyzer for spell checking. Simply if the word can be analysed by the morphological analyzer we considered as a correctly spelled word while if the analyzer was unable to analyse the word we considered as a misspelled word. That methodology was able to give quite high accuracy for the word which root word is in the morphological analyzer. So in order to increase the accuracy, we have to feed more root words to the morphological analyzer. But the main drawback of this approach is the performance issue. As FOMA can perform only one I/O operation at a time, it takes a considerable amount of time for the spell checking. So instead of using the morphological analyzer directly we preprocessed the morphological analyzer to generate all the words which are analyzed by the morphological analyzer.

So as our next approach, we used these pre generated words with some other spell corrected word list to build a dictionary with about 1 million spell corrected Sinhala words. Used the dictionary data structure in Python to store the above mentioned dictionary. Dictionary lookup was way faster than the morphological analyzer based approach. But the drawback was updating the dictionary over the time and unable to include all of the inflections of a particular root word to the dictionary. So as a solution to that we introduced a machine learning based approach to analyse the misspelled words. Morfessor helps to analyse the words in to root and the inflections and if the root is already in the spell corrected dictionary we mark the word as a potentially correct word using yellow color.

After a word is classified as a misspelled word by the spell checking module, the suggestion generation module is responsible for producing the best suggestions for the user. For suggestion generation we did not use typical minimum edit distance or bi-gram based methods which are quite low in accuracy as well as consumes lot of time according to the size of the dictionary. As our approach, we trained a machine learning model named fasttext which incorporates n-gram and subword information of the words to determine the word vectors. The main advantage of using this model is because it has the ability to generate similar words(syntactically and semantically) for a particular given word in a matter of milliseconds. Due to that we could gain higher performance speed up for our suggestion

generation module. But the suggested words by the fasttext model is not very accurate for Sinhala language. As some of the spelling mistakes are more common than others, some kind of priority should be given to those types of errors. So we have used an error model which is used to score the words given by the fasttext model based on factors such as the type of spelling error, minimum edit distance and frequency of the word. After that top 5 highest scored words are shown to the user as the suggestions.

As our final outcome, we integrated these 2 modules into a web application to give an interactive experience in spell checking for the user. Users are given a window to type interactively or copy paste a document to check the spelling errors. When a user type something or paste some document, although the browser renders the words and showed as a correctly spelled word, there can be some unicode errors which are not visible. So we have developed a unicode cleaner module which can detect those types of errors and give suggestions for the user with the correct word.

The spell checker and the suggestion generation module was evaluated alongside currently implemented spell checkers such as microsoft word, Hunspell and Subasa. For the spell checker we were able to achieve 96.9% true positive accuracy and 91% false negative accuracy. For the suggestion generation module 90% accuracy was able to achieve which is 70% of the suggestions are first suggestions. These accuracies are quite high compared to currently implemented spell checkers.

As we followed data driven approach, the accuracy of the application is solely based on the accuracy of the dataset used for the implementation. One of the main challenges we faced during the implementation of the Helabasa project was finding a spell corrected corpus for Sinhala language. As well as finding the root words for the morphological analyzer out of these misspelled words was another challenge we faced. As we have identified accuracy of the false negatives can be improved with a proper spell corrected dataset. As Sinhala words can be written in multiple formats with a proper preprocessing module also we can improve the accuracy of the system.

6. References

- [1] R. W. R. P. C. L. a. E. J. Asanka Wasala, "*A Data-Driven Approach to Checking and Correcting Spelling Errors in Sinhala*," *The International Journal on Advances in ICT for Emerging Regions*, vol. 3.
- [2] A. W. ., W. Eranga Jayalatharachchl, "Data-Driven Spell Checking: The Synergy of Two Algorithms for Spelling Error Detection and Correction," *The International Conference on Advances in ICT for Emerging Regions - iCTer* , pp. 7-13, 2012.
- [3] Subhagya, L.G., Ranathunga, L., Nimasha, W.H., Jayawickrama, B.R., & Mahaliyanaarchchi, K.L. (2018). Data Driven Approach to Sinhala Spell Checker and Correction. *2018 18th International Conference on Advances in ICT for Emerging Regions (ICTer)*, 01-06.S
- [4] H. Kaur, G. Kaur and M. Kaur, "Punjabi Spell Checker Using Dictionary Clustering," *International Journal of Science, Engineering and Technology Research*, vol. 4, no. 7, pp. 2369-2374, July 2015.
- [5] G. U. M. Rao, P. A. Kulkarni and C. Mala, "A Telugu Morphological Analyzer," *International Telugu Internet Conference Proceedings*, 28th - 30th September 2011.
- [6] B. C. Dongxu Han, "A Maximum Entropy Approach to Chinese Spelling Check," *Proceedings of the Seventh SIGHAN Workshop on Chinese Language Processing*, pp. 74-78, 2013.
- [7] B, Premjith & Kp, Soman & Kumar, M. (2018). A deep learning approach for Malayalam morphological analysis at character level. *Procedia Computer Science*. 132. 47-54. 10.1016/j.procs.2018.05.058.

- [8] Sooraj, S & K, Manjusha & Kumar, M & Kp, Soman. (2018). Deep learning based spell checker for Malayalam language. *Journal of Intelligent & Fuzzy Systems*. 34. 1427-1434. 10.3233/JIFS-169438.
- [9] Bojanowski, P., Grave, E., Joulin, A. and Mikolov, T., 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5, pp.135-146.
- [10] GitHub. (2019). *mhulden/foma*. [online] Available at <https://github.com/mhulden/foma> [Accessed 7 Aug. 2019].
- [11] [TechRep] (1, 2, 3) Sami Virpioja, Peter Smit, Stig-Arne Grönroos, and Mikko Kurimo. Morfessor 2.0: Python Implementation and Extensions for Morfessor Baseline. Aalto University publication series SCIENCE + TECHNOLOGY, 25/2013. Aalto University, Helsinki, 2013. ISBN 978-952-60-5501-5.
- [12] [Creutz2002] Mathias Creutz and Krista Lagus. Unsupervised discovery of morphemes. In *Proceedings of the Workshop on Morphological and Phonological Learning of ACL-02*, pages 21-30, Philadelphia, Pennsylvania, 11 July, 2002.
- [13] [Kohonen2010] Oskar Kohonen, Sami Virpioja and Krista Lagus. Semi-supervised learning of concatenative morphology. In *Proceedings of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology*, pages 78-86, Uppsala, Sweden, July 2010. Association for Computational Linguistics.
- [14] Welgama Viraj, Weerasinghe Ruwan, and Mahesan Niran (2015) 'Defining the Gold Standard Definitions for the Morphology of Sinhala Words', *Research in Computing Science* , 90(), pp. .

- [15] V. Welgama, R. Weerasinghe, and M. Niranjana, “Evaluating a Machine Learning Approach to Sinhala Morphological Analysis,” in *Evaluating a Machine Learning Approach to Sinhala Morphological Analysis*, n.d..
- [16] Mikolov, T., Chen, K., Corrado, G. and Dean, J., 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.
- [17] Fasttext.cc. (2019). fastText. [online] Available at: <https://fasttext.cc/> [Accessed 21 Mar. 2019].
- [18] Github.io. (2011). *MorphologicalAnalysisTutorial - foma - A self-contained tutorial for building morphological analyzers. - finite-state compiler and C library*. [online] Available at: <https://fomafst.github.io/morphutut.html> [Accessed 7 Dec. 2019].
- [19]Hulden, M., 2009, April. Foma: a finite-state compiler and library. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics: Demonstrations Session* (pp. 29-32). Association for Computational Linguistics.
- [20] Analytics Vidhya (2017). *Understanding Word Embeddings: From Word2Vec to Count Vectors*. [online] Analytics Vidhya. Available at: <https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/> [Accessed 4 Nov. 2019].
- [21] mhulden (2016). mhulden/foma. [online] GitHub. Available at: <https://github.com/mhulden/foma/blob/master/foma/docs/simpleintro.md> [Accessed 7 Dec. 2019].
- [22] En.wikipedia.org. (2019). *Spell checker*. [online] Available at: https://en.wikipedia.org/wiki/Spell_checker [Accessed 11 Apr. 2019].

