**Interim Report**

**Level 4**

**Automated Evaluation of Algebraic Word Problems in Mathematics**

Team 789

|  |  |
| --- | --- |
| Name | Index Number |
| 154187F | W.M.S.A. Panditharatne |
| 154183N | L.A.D.C.U. Rathnaweera |
| 154079C | C.M.P.P.D.V. Perera |

Faculty of Information Technology

University of Moratuwa

2020

**Final Report**

**Level 4**

**Automated Evaluation of Algebraic Word Problems in Mathematics**

Team 789

|  |  |
| --- | --- |
| Name | Index Number |
| 154187F | W.M.S.A. Panditharatne |
| 154183N | L.A.D.C.U. Rathnaweera |
| 154079C | C.M.P.P.D.V. Perera |

Supervised by: Dr. L. Ranathunga

Signature: …………………….

Faculty of Information Technology

University of Moratuwa

2020

**Dedication**

We would like to dedicate our project,

“Automated Evaluation of Algebraic Word Problems in Mathematics”

To,

The Dean of Faculty of Information Technology, Mr. B. H. Sudantha ,

Supervisor of the Project, Dr. Lochandaka Ranathunga,

All the academic and non-academic staff in the Faculty of Information Technology who inspired and motivated us in numerous ways.

**Acknowledgement**

We would like to acknowledge and extend our heartfelt gratitude to the following people who have helped and guided us in numerous ways to make the project a success.

Our supervisor Dr. Lochandaka Ranathunga, for the immense assistance and guidance he has always given us and motivating us towards achieving the aim of the project.

All the academic staff at the Faculty of Information Technology of University of Moratuwa, who have given us knowledge and guidance throughout our undergraduate years.

All the non-academic staff of Faculty of Information Technology of University of Moratuwa, members of our families, our friends and batch mates who have supported us in numerous ways.

**Abstract**

Table of Contents

[Chapter 1 12](#_Toc33841090)

[Introduction 12](#_Toc33841091)

[1.1 Introduction 12](#_Toc33841092)

[1.2 Background & Motivation 12](#_Toc33841093)

[1.3 The Problem in Brief 13](#_Toc33841094)

[1.4 Aim & Objectives 14](#_Toc33841095)

[1.5 Proposed Solution 15](#_Toc33841096)

[**1.6 Summary** 15](#_Toc33841097)

[Chapter 2 17](#_Toc33841098)

[Other’s Work 17](#_Toc33841099)

[2.1 Introduction 17](#_Toc33841100)

[2.2 Handwritten OCR System 17](#_Toc33841101)

[2.3 Algebraic Word Problem Analyzer and Equation Generation Module 22](#_Toc33841102)

[2.4 Dataset Repository and Performance Analysis 25](#_Toc33841103)

[Chapter 3 27](#_Toc33841104)

[Technology Adopted 27](#_Toc33841105)

[3.1 Introduction 27](#_Toc33841106)

[3.2 Technologies Adopted for Implementation 27](#_Toc33841107)

[3.2.1 Broad Technologies 27](#_Toc33841108)

[3.2.2 Programming Languages 27](#_Toc33841109)

[3.2.3 Development Tools 27](#_Toc33841110)

[3.2.4 Version Controlling System 28](#_Toc33841111)

[3.3 Summary 28](#_Toc33841112)

[Chapter 4 29](#_Toc33841113)

[Approach 29](#_Toc33841114)

[4.1 Introduction 29](#_Toc33841115)

[4.2 Overall System 29](#_Toc33841116)

[4.3 Handwritten Optical Character Recognition module 29](#_Toc33841117)

[4.3.1 Users 29](#_Toc33841118)

[4.3.2 Input 29](#_Toc33841119)

[4.3.3 Output 30](#_Toc33841120)

[4.3.4 Scope limitations 31](#_Toc33841121)

[4.3.5 Process 31](#_Toc33841122)

[4.4 Algebraic word problem Analyzer. 31](#_Toc33841123)

[4.4.1 Users 31](#_Toc33841124)

[4.4.2 Input 31](#_Toc33841125)

[4.4.3 Output 32](#_Toc33841126)

[4.4.4 Process 32](#_Toc33841127)

[4.5 Answer generating Module 32](#_Toc33841128)

[4.5.1 Input 33](#_Toc33841129)

[4.5.2 Output 33](#_Toc33841130)

[4.5.3 Process 33](#_Toc33841131)

[Chapter 5 34](#_Toc33841132)

[Methodology 34](#_Toc33841133)

[5.1 Introduction 34](#_Toc33841134)

[5.2 High-level Architecture of the Overall System 34](#_Toc33841135)

[5.3 Handwritten Optical Character Recognition module 36](#_Toc33841136)

[5.3.1 Pre-Processing 36](#_Toc33841137)

[5.3.2 Line Segmentation 38](#_Toc33841138)

[5.3.4 Character Classification 53](#_Toc33841139)

[5.4 NLP Preprocessing Module. 53](#_Toc33841140)

[5.4.1 Raw Data 54](#_Toc33841141)

[5.4.2 Tokenizer 55](#_Toc33841142)

[5.4.3 Stemming 55](#_Toc33841143)

[5.4.4 Filtering 58](#_Toc33841144)

[5.4.5 POS Tagging 60](#_Toc33841145)

[5.4.6 Fetch Data for Models 63](#_Toc33841146)

[5.5 Answer Generating Module 66](#_Toc33841147)

[5.5.1 Simple Equation Solving Module 67](#_Toc33841148)

[5.5.2 Complex Simultaneous Equations Solving Module 71](#_Toc33841149)

[5.6 Summary 81](#_Toc33841150)

[Chapter 6 82](#_Toc33841151)

[Implementation 82](#_Toc33841152)

[6.1 Introduction 82](#_Toc33841153)

[6.2 Handwritten Optical Character Recognition Module 82](#_Toc33841154)

[6.2.1 Preprocessing 82](#_Toc33841155)

[6.2.2 Line Segmentation 82](#_Toc33841156)

[6.2.3 Initial Segmentation 82](#_Toc33841157)

[6.3 NLP Preprocessing Module 86](#_Toc33841158)

[6.3.1 Raw data Extraction 86](#_Toc33841159)

[6.3.2 Tokenizer 88](#_Toc33841160)

[6.3.3 Stemming 89](#_Toc33841161)

[6.3.4 Filtering Module 89](#_Toc33841162)

[6.3.5 Part of Speech Tagging 91](#_Toc33841163)

[6.3.6 Fetch Data for Models 91](#_Toc33841164)

[6.4 Equation Generation Module 93](#_Toc33841165)

[6.4.1 Simple Problem Module 94](#_Toc33841166)

[6.4.2 Complex Problem Module 97](#_Toc33841167)

[Evaluation 101](#_Toc33841168)

[7.1 Introduction 101](#_Toc33841169)

[7.2 Evaluation of the Handwritten OCR Module 101](#_Toc33841170)

[7.2.1 Module wise evaluation 101](#_Toc33841171)

[7.2.1 Evaluation on real life data 102](#_Toc33841172)

[7.2 Evaluation of Preprocessing Module 104](#_Toc33841173)

[7.3 Evaluation of Equation Generation Module 106](#_Toc33841174)

[7.3.1 Evaluation of Classification models 106](#_Toc33841175)

[7.3.1 Evaluation of Rule-Based module 108](#_Toc33841176)

[Chapter 8 115](#_Toc33841177)

[Further Work 115](#_Toc33841178)

[8.1 Handwritten Recognition Module 115](#_Toc33841179)

[8.2 Preprocessing Module 115](#_Toc33841180)

[8.2.1 Limitations of Preprocessing Module 115](#_Toc33841181)

[8.2.2 Future works of Preprocessing Module 116](#_Toc33841182)

[8.3 Equation Generation Module 116](#_Toc33841183)

[8.3.1 Limitations of Equation Generation Module 116](#_Toc33841184)

[8.3.2 Future works of Equation Generation Module 116](#_Toc33841185)

[8.4 Conclusion 117](#_Toc33841186)

[References 119](#_Toc33841187)

[Appendixes 121](#_Toc33841188)

[Appendix A – Individual contribution 121](#_Toc33841189)

[Appendix B 123](#_Toc33841190)

[Figure 1- Line segmentation using vertical projection (a)Rotated Image (b)Projection profile (c)Line segmentation by peaks [4] 18](https://d.docs.live.net/3779b0908a60b80a/Documents/FYP%20Final%20Report.docx#_Toc33841037)

[Figure 2 - Touching and overlapping characters [1] 19](https://d.docs.live.net/3779b0908a60b80a/Documents/FYP%20Final%20Report.docx#_Toc33841038)

[Figure 3- Flowchart of the algorithm proposed by Lu et al. [9] 20](#_Toc33841039)

[Figure 4 Example feature points detected by the algorithm of Lu et al. [9] 20](#_Toc33841040)

[Figure 5- Samples of the datasets used by Zhan et al [10]. 21](#_Toc33841041)

[Figure 6 Accuracy of statistic-based and tree-based methods in solving arithmetic problems. 26](#_Toc33841042)

[Figure 7 Example input image to Handwritten OCR module 30](#_Toc33841043)

[Figure 8 Example output text from Handwritten OCR module 30](https://d.docs.live.net/3779b0908a60b80a/Documents/FYP%20Final%20Report.docx#_Toc33841044)

[Figure 9 Example of problem sentence 32](#_Toc33841045)

[Figure 10 High level architecture of the overall System 35](#_Toc33841046)

[Figure 11 High level architecture of Handwritten OCR Module 36](#_Toc33841047)

[Figure 12 Handwritten OCR Pre Processing steps (a) Input Image (b)Gaussian Blurring and Gray Level Transform (c)Black Hat Transform (d)OTSU Thresholding 37](#_Toc33841048)

[Figure 13 Line Segmentation Design 38](#_Toc33841049)

[Figure 14 Line Segmentation Process (a)After applying closing operation (b)-(e)Detected Lines 39](#_Toc33841050)

[Figure 15 Initial character segmentation 40](#_Toc33841051)

[Figure 16 Segmented Characters from lines 40](#_Toc33841052)

[Figure 17 Samples and their labels 42](#_Toc33841053)

[Figure 18 Explicit Segmentation overall architecture 44](#_Toc33841054)

[Figure 19 Scatter plot of image widths(x) and image white pixel counts(y) 45](#_Toc33841055)

[Figure 20 Samples of the generated dataset 46](#_Toc33841056)

[Figure 21 (a,c)Touching/Overlapping Character Image (b,d)Background thinned image 46](#_Toc33841057)

[Figure 22 (a)Touching character image (b)Background thinned image (c)The middle section outlined in red (d) Candidate points found (Red- candidate point in upper part, Green- Candidate point in lower part) (e)Candidate points in the background image (d)Final image with connected candidate points 47](#_Toc33841058)

[Figure 23 Convolution kernel to find candidate points in the upper part 48](#_Toc33841059)

[Figure 24 Types of candidate points found by the kernel 48](#_Toc33841060)

[Figure 25 Convolution kernel to find candidate points in the lower part 48](#_Toc33841061)

[Figure 26 Types of candidate points found by the kernel 49](#_Toc33841062)

[Figure 27 Flowchart for splitting characters using contour hierarchy 51](#_Toc33841063)

[Figure 28 Images that can be split using contour hierarchy 52](#_Toc33841064)

[Figure 29 Contour hierarchy 52](#_Toc33841065)

[Figure 30 Preprocessing Module 53](#_Toc33841066)

[Figure 31 PDF Extraction or get raw data string from given source 54](#_Toc33841067)

[Figure 32 Raw String with Question 55](#_Toc33841068)

[Figure 33 POS tagged Corpus 62](#_Toc33841069)

[Figure 34 POS tagged Output 63](#_Toc33841070)

[Figure 35 A input with requirement to replace තෙගුණයක් with value 3 64](#_Toc33841071)

[Figure 36 Flow chart of feature extraction module. 65](#_Toc33841072)

[Figure 37- Answer Generating Module 67](#_Toc33841073)

[Figure 38- Flow chart of Simple Equation Solving Module 68](#_Toc33841074)

[Figure 39- Summary of Classification ANN of Simple problems 70](#_Toc33841075)

[Figure 40- Output of Type Classification Model for Simple questions 70](#_Toc33841076)

[Figure 41- Output of Simple Equations Generating and Solving Module 71](#_Toc33841077)

[Figure 42- Flowchart of Rule Based approach of Complex Simultaneous Equations Solving Module 72](#_Toc33841078)

[Figure 43- Flowchart of Dynamic approach of Complex Simultaneous Equations Solving Module 74](#_Toc33841079)

[Figure 44- Summary of Classification ANN of Complex problems 77](#_Toc33841080)

[Figure 45- Output of Classification module of Complex problems 77](#_Toc33841081)

[Figure 46- Output after execution of rules in the template 78](#_Toc33841082)

[Figure 47- Noise in the result. 79](#_Toc33841083)

[Figure 48- Example for Rule in a template (Template 01 – rule 08) 98](#_Toc33841084)

[Figure 49- Example for Rule in a template (Template 03 – rule 06) 99](#_Toc33841085)

[Figure 50- Example for Rule in a template (Template 04 – rule 04) 99](#_Toc33841086)

[Figure 27 Single character classifier accuracy 101](#_Toc33841087)

[Figure 28 Single or multiple character classification (binary) 102](#_Toc33841088)

[Figure 29 Single or multiple character segmentation and classification accuracy 102](#_Toc33841089)

# Chapter 1

# Introduction

## 1.1 Introduction

Vast growth of technology causes to development of many fields such as in the fields of Medicine, Engineering, Transportations and Education. Technologies are used in every industry reaching to the higher level of accuracy and to achieve expected results efficiently. Education field is one of the most important sectors that should be integrated with the technology. But so far evaluating academic performance of school children has not much approached by the technology. Most of the times in schools’ teachers correct homework in primary classes. But when entering to the secondary education (from grade 6) checking homework and marking corrections in each and every book is not an easy task for teachers due to the increasing number of subjects and the number of students. Subjects like mathematics should be given a special attention as it helps improve and enhance the logical thinking skills of students. Therefore, an effective method of solving this problem is much valuable to the education sector.

In our project we focus on algebraic equations in grade 6, 7, 8 mathematics. The technologies planning to use in completing this project are Image Processing, Semantic Analysis Natural Language processing and Deep Learning.

The scope of our solution can be simply categorized under 3 areas.

* Implementing a module to process algebraic expression questions. (Classifying parameters and variables)
* Implementing a module read handwritten answers using image processing and Machine Learning.
* Implementing a module to generate answers based on the processed algebraic equation and check them against the written answers using Semantic Analysis.

## 1.2 Background & Motivation

Currently education field faces many issues regarding various aspects. Approaching these problems with effective solutions from technology will be more useful and contributes to positive outcomes. As we believe students should pay more attention to mathematic subject as they enter to their secondary education from grade 6. Improving logical thinking and analyzing skills of students from the beginning is much useful their future. This can be achieved from continuous assessment of students’ daily works. Now a days there are 5 or 6 classes for each grade in schools and teachers have less time to correct and mark all the books. It’s time consuming and requires much effort for the task. Therefore, it’s hard for teachers to pay attention separately to each student and their weaknesses as they have to spend their time only to teach and just mark and correct books.

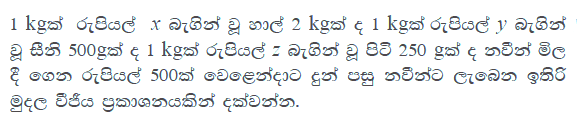
But now a days there are technologies such as NLP, Image processing and Machine learning that hold much potential in education sector. These technologies can be producing an effective solution for the above problem. Automated marking and corrections of algebraic expressions is a better approach for this problem. Also achieving higher level of accuracy in this task can be much useful. Because then the teachers have time to pay much attention to the each and every individual separately, recognize their weaknesses and take necessary steps to improve those weaken areas.

Therefore, our motive is to develop a solution for automated evaluation of algebraic word problems.

## 1.3 The Problem in Brief

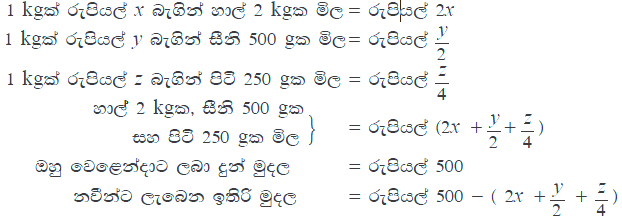
Algebraic expressions are a fundamental section in mathematics. These expressions are needed to solve various kinds of problems. For an example, following is an algebraic expressions question from Grade 8 Mathematics book [1].

Question:



The following is the expected answer to the above question.

Answer:



Currently there exists no method to automate checking of answers that are in written format or text format to algebraic expression questions. Countless hours spent by teachers to mark these questions and answers can be saved by automating this process.

## 1.4 Aim & Objectives

#### 1.4.1 Aim

The aim of this project is to implement a system that can process algebraic expression related questions and validate answers handwritten answers.

#### 1.4.2 Objectives

* To study NLP algorithms
* To study Image Processing and Machine Learning
* To implement a module to process the algebraic expression questions
* To implement a module to generate answers based on the processed algebraic equation
* To implement a module to read handwritten answers and check them against the generated answers.

## 1.5 Proposed Solution

Addressing the above-mentioned problems, we propose a solution which can read and understand the given questions as in sentences and generate all the correct algebra equations for those given question and use those generated answers to check answers which are written by students. In this way, correction of algebraic questions can be automated through this solution.

The system has the following three main areas,

1. NLP Module which performs entity identification to identify variables, operators and parameters

When the question is uploaded in pdf format this module will process Sinhala characters and perform entity identification which will identify variables, operators and parameters.

As an example, if question has said that “The price of the paper is *10*” this model has to identify that *price of the paper* is a variable and it is equal to 10.

1. Equation Generation Module.

This module will use semantic analysis techniques to generate correct algebraic equation from the entities, variables and parameters that are identified by the first module. Then the generated answers will be checked against the hand-written answers of students that will be retrieved from the next module.

1. Image processing and Machine learning model to read handwritten answers of students.

The images of the hand-written answers uploaded to this module will be processed to recognize handwritten characters using image processing and Machine Learning techniques.

**1.6 Summary**

This chapter has discussed an introduction to the research area, background and motivation to conduct this research, the aim and objectives of the research. The rest of the report as follows. Chapter 2 discusses about already implemented systems related to Mathematical equation solving. Chapter 3 describes the technologies adopted during the implementation of the system. Chapter 4 illustrates on the approach to developing the system. Chapter 5 depicts the analysis and design of the proposed system and its sub modules. Chapter 6 discusses about the implementation and the current progress of the system. Next chapter, chapter 7 give the conclusion of the report and the future works of the system. Finally, the report will conclude with a list of references that was used during the implementation of the system.

# Chapter 2

# Other’s Work

## 2.1 Introduction

This chapter explores existing literature carried out on similar problem domains. Other’s work is categorized by the two main modules of the proposed system.

## 2.2 Handwritten OCR System

Handwritten text is still a widely used method of communication in todays world. Even with the advancements in digital text applications, handwritten text is still the medium for text in fields such as education. Thus, it’s vital to make advancements in recognizing such texts.

There exist many researches regarding handwritten text recognition. Even though many advancements have been made, hand written character recognition is still considered an open problem. The usual process of OCR is to Pre-Process the document image, segment lines, segment characters and finally classify the segmented characters [1].

1. **Pre-Processing**

Colored images are transformed into gray level images in order to address issues of colored pens and colored backgrounds. Thresholding is applied to remove unwanted background and highlight the text. Thinning and skeletonization is used to achieve constant stroke width which resolves using of different pens and varying stroke width. Noise removal process is applied to reduce unwanted high spatial frequencies [2]. Skewness or slant correction is sometimes applied in case of words being slanted.

1. **Line Segmentation**

Text line extraction is the first step in document processing. The accuracy of the extracted lines directly correlates with the accuracy of correct recognition of characters. Due to diversity in handwriting such as tilt, adhesion, overlap etc. there many approaches to tackle the issue [3].

In [4], Manmatha et al. discusses a method of using horizontal projection to detect the amount of text in a row of the image. The horizontal projection is the sum of gray values in rows in the image.

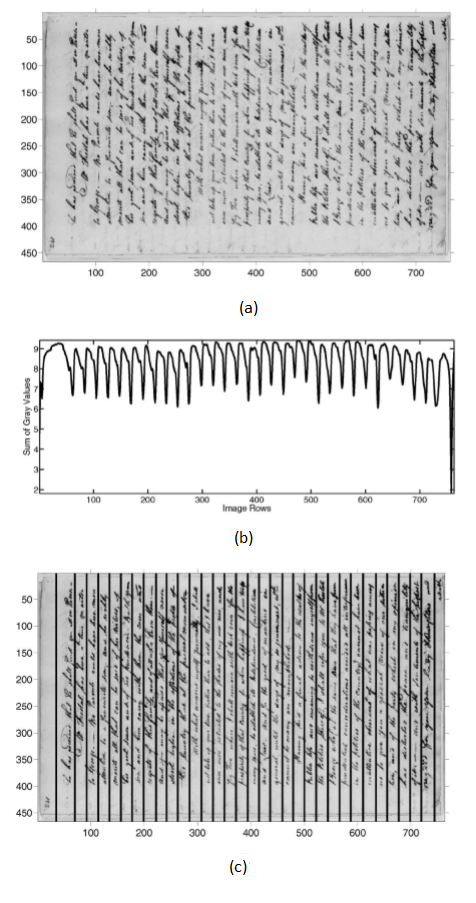
The troughs of the projection will represent areas of text while the peaks of the projection will represent the areas of white space.

Figure - Line segmentation using vertical projection (a)Rotated Image (b)Projection profile (c)Line segmentation by peaks [4]

To reduce the number of false local maxima, the projection function is smoothed using the Gaussian filter. This eliminates false alarms and reduces the noise sensitivity. Further, this technique is robust to variation of line size. However, in documents where text line skewness, overlapping and touching lines are present this method fails to correctly segment the lines.

In [5], Wu et al. proposes a method of using morphology-based algorithm to extract text from images. Text lines usually have a high contrast to the background with uniform widths and heights. Thus, morphology operations can be applied to extract candidate contrast regions as possible text lines. This method is robust against different lighting conditions and geometrical image transformations like skewing, translation, rotation and scaling.

1. **Character Segmentation**

Character segmentation is considered one of the most crucial steps in handwritten OCR [6]. Correct segmentation of characters is imperative for correct classification of characters in later stages.

A close up of a sign

Description automatically generatedAlthough characters that are separated by spaces can be easily segmented there are challenges in segmenting touched or overlapping characters. The below figure demonstrates some common ways in which characters can interfere [1]. Touching, connected and intersecting characters will all be referred as touching characters from here onwards for simplicity.

Figure - Touching and overlapping characters [1]

In research, character segmentation is broadly classified into two main techniques [7].

* Implicit segmentation – Recognition based segmentation method where character recognition and segmentation are achieved simultaneously. Segmentation can be regarded as a byproduct of the recognition process [7]. Thus, the recognizer decides on the best segmentation hypothesis.
* Explicit segmentation – A classical approach where an image of a word sequence is partitioned to several, individual character images.

**Explicit Segmentation**

In [8] Casey et al. demonstrates a method of using vertical projection (vertical histogram) of a line to find separation of characters. Vertical projection is the count of foreground pixels in each column of the line. The minimums(valleys) of the projection serves as a detector for white space between successive letters thus the segmentation points. However, when touching or overlapping characters are introduced this method fails to correctly segment the characters. Thus, this method serves as a successful method for segmentation for non-cursive handwriting.

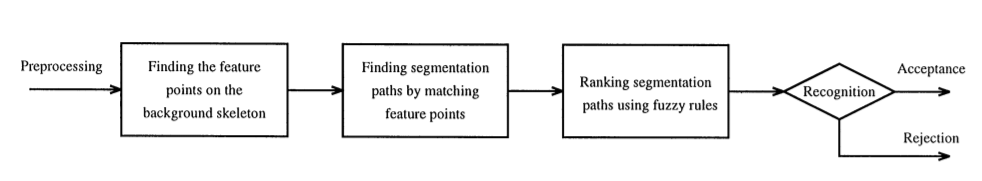
In [9] Lu et al. proposes an approach of segmenting hand-written two-digit strings by thinning the background of the image. 

Figure - Flowchart of the algorithm proposed by Lu et al. [9]

After retrieving the skeleton of the background, the algorithm locates several feature points that can be used to segment the image.

A close up of a map

Description automatically generated

Figure Example feature points detected by the algorithm of Lu et al. [9]

Possible segmentation paths are the constructed between the detected feature points of the upper background contour and the lower background contour. Since there can be multiple possible segmentation paths, they are ranked using fuzzy rules generated from a decision-tree approach. Then the segmentation paths with the highest ranks are tested according to an acceptance criterion.

**Implicit Segmentation**

In [10], Zhan et al. proposes a new architecture based on a CNN to recognize handwritten digit strings. The CNN consists of a feature extraction layer, feature dimension transposition layer and an output layer. A Connectionist Temporal Classification (CTC) layer is used in the output layer to calculate the loss and decode the feature sequence.

A screenshot of a cell phone

Description automatically generated

Figure - Samples of the datasets used by Zhan et al [10].

In [11], Meany et al. trains a CNN on the IAM Handwriting word dataset. They have limited the length of the words to 1-6 characters to achieve best performance. They have trained the dataset on a 1-layer CNN and a 3-layer CNN. They have achieved accuracies of 71.6% and 76.9% respectively for the test set.

1. **Character classification**

In [11], Meany et al. trains a CNN and a ResNet-50 on the EMNIST dataset to achieve character classification. They have trained the model on the EMNIST balanced dataset where there are 47 classes. The CNN reaches an accuracy of 84.8% and the ResNet reaches an accuracy of 88.5% on the test set.

## 2.3 Algebraic Word Problem Analyzer and Equation Generation Module

Knowledge representation methods using NLP has been improved with different research areas. Using schemas for knowledge representation is a novel way in these days. Sentences are analyzed and keywords in each sentence is mapped to a defined schema. It is very rare to find a sentence that explicitly match to template. Therefore, when a keyword is detected it should be adjusted with the schema. Furthermore, this system has followed several steps to simplify the question text. Simplifying the problem involves resolving conjunctions, preprocessing currency, resolving co-references, preprocessing sentences and resolving missing entities. Then the sentence is analyzed to extract information [12].

D.Zhang et al. presents a survey based on Automatic Math problem Solvers. The paper has provided a descriptive analysis about the various methods that have been used so far. Rule based systems are early approach used for Math word Problems (MWP). Answer is derived by applying set of rules on prepositions that are extracted from the problem text. Statistical based methods use classifiers to identify the properties of a text. Advance logic templates are required to solve multi stem mathematical expressions. Logic template contains, entities, relations, attributes and values. Tree based approaches construct and equation tree from the derivations. In the equation tree, operators with higher priority are placed in the lower levels of the tree. Template based methods map extracted information to predefined template. Deep learning-based methods mostly use seq2seq model to transform information to a template. This paper contains preprocessing methods that are commonly used by different systems. Syntactic parsing arranges lexical units and their dependencies to tree structure. Stanford Parser is often used in those systems. Coreference resolution is a subtask of preprocessing stage that recognize and identify nouns that refer to same entity. Quantity related features are used to identify operators and variables associated with them. Quantity pair features are useful to identify the relationship between two quantities and which operator can be used to connect them. Question related features and verb related features are also extracted for Information extraction in Natural language processing is used for various tasks such as, text summarization, Machine translation, question answering systems etc. Several subtasks that are involved with information extraction are Named Entity Recognition, coreference resolution, Named Entity Linking, Relation Extraction etc. Accuracy of extracted information is highly depended on the preprocessing stage. Part-of-speech tagging is used to annotate words with grammatical meaning. Rule based and statistical approaches are used to perform part-of-speech tagging. Named Entity Recognition is used to identify the entities in a text such as persons, organizations, locations etc. Named Entity Linking is used to recognize the references for a particular entity. Event extraction and relation extraction is used to identify events and predefined relationships respectively [13].

Kushman et al. proposed a system to solve algebraic word problems, using template-based method. Problem text is mapped to an equation template to present a semantic representation of the text. Mapping is basically contained 2 steps. Those are selecting a template for equation and instantiate the template with nouns and the numbers in the text. Induction of the template is done by learning. The type of the template structure is chosen during the learning process. Slots in the equation template is aligned with the values and variables in the problem text. This proposed system has defined the “space of possible derivations” which has used to define the space of equations by generating set of possible templates for a particular problem. A probabilistic model has used to select the most suitable template. Model has used template canonicalization to avoid ambiguity of expressing the semantically equal equations in different syntaxes. Although the system has achieved high accuracy system contains several drawbacks. It has used weak supervision method which has only selected five most common types of questions. Therefore, the system only covers limited number of templates. Also, the system includes errors due to the lack of understanding of real-world concepts and compositional language [14]

ALGES system present a system to solve algebraic word problems by generating equation trees with the use of integer linear programming and scoring functions. ALGES map problem text to arithmetic operators through a learning process and set of possible trees are generated to solve the problem. System approaches to extract properties of the text as Quantified sets or Qsets which are used to represent the nodes of the trees. Extracting the quantities are done by Qsets during the learning process to map text to equation trees. System consist of main two equation models. Local equation model which is trained to compute the likelihood of combining two Qsets with an operator. Global Equation model is trained to select the best equation tree using a scoring function. ALGES has used an ILP optimization model to select most suitable candidate equation tree. Apart from improvements than the other systems ALGES contains issues in Qset ordering which leads to wrong interpretations. Also, the system contains semantic limitations due to the lack of real-world knowledge [15].

ARIS system presents a methodology for arithmetic word problems solving using verb categorization. System follows a different method than other similar systems to build the equation. Approach of this system consist of 3 main steps. Firstly, problem is grounded in entities and containers which returns entities, attributes, quantities and containers of problems. Entities are objects that are associated with quantities and containers are location where each entity belongs. 2nd step is to categorize the verbs in sentences. Analyzing the problems, 7 verb categories have been identified and the system contains a model that is trained identify the category of the verbs. Forming the equation and derive the solution is the final step of the model. Equation formulation is achieved through state transition progression. Despite the novelty of the system still considerable portion of weaknesses can be recognized due to the irrelevant information, parsing issues, of the used methodology [16].

Approach of this system attempts to solve math word problems by mining fine-grained textual expressions from problem to derive math concepts. Mining Textual expression is proceeding through learning process. System constructs a template sketch which contains the information of the sketch. Values of the expressions are aligned only to N most probable templates. The equation is parsed to a hierarchical tree to derive the template fragments through the subtree that are at least associated with one number slot. Alignment ranking is performed to find out the template that has the highest probability for the N most probable templates.

|  |  |  |
| --- | --- | --- |
| Equation | Template | Phrases |
| 1-.28 | 1-n1 | a discount of [*NUM*]% |
| (1-.28)\* 275 | (1-n1)\*n2 | A discount of [*NUM*]% on [*VAR]* priced at [*NUM*] |

However, the system contains some amounts of drawbacks. Errors in determining the type of quantities. As an example, ’24 male’ has same interpretation of ‘men’. Issues of identifying change of states. Errors due to the lack of external specific mathematical models such as permutation, combination and knowledge of common sense [17].

Automatic mathematical problem solvers use many different approaches in order to maximize results and to increase the efficiency of models. In this system new representation language has defined to do the mapping of text and math expressions. This proposed system called SigmaDolphin has implemented a Semantic parser to transform problems to trees. Semantic parsing based on context-free grammar (CFG). In this approach Natural language text is transformed to semantic representation. System contains a reasoning module that can build math expression from the semantic representation [18].

Automatic math problem solving has used sequence to sequence models recently. The method contains two stages, first maps the significant numbers in the text to number tokens as their order in the text. Second stage is aiming in training SEQ2SEQ models to map to the equation from the text. Equations are further normalized to an expression tree [19]. Similar approach has been carried out by Roth et al. that presents a theory to solve arithmetic problems solving using expression tree. Problem text is input to the system and aimed to produce a read-once expression which use each quantity of the text at most one time [20]

## 2.4 Dataset Repository and Performance Analysis

The accuracy of these math word problems solvers is evaluated on the datasets that are manually harvested and annotated from online math problems related websites which are small-scale and contain hundreds of math problems.

**1) AI2 [25] -** There are 395 single-step or multi-step arithmetic word problems for the third, fourth, and fifth graders. It involves problems that can be solved with only addition and subtraction.

**2)** **IL [30] -** The problems are collected from websites k5learning.com and dadsworksheets.com. The dataset contains 562 single-step word problems with only one operator, including addition, subtraction, multiplication, and division.

**3) CC [30] -** The dataset is designed for multi-step math problems. It contains 600 multi-step problems without irrelevant quantities, harvested from commoncoresheets.com.

**4) SingleEQ [31] -** The dataset contains both single-step and multi-step arithmetic problems and is a mixture of problems from a number of sources. Each problem involves operators of multiplication, division, subtraction, and addition over non-negative rational numbers.

**5) AllArith [32] -** The dataset is a mixture of the data from AI2, IL, CC and SingleEQ. All mentions of quantities are normalized into digit representation.

6) **MAWPS [49]** – This is another testbed for arithmetic word problems with one unknown variable in the question.

**7) Dolphin-S.** - This is a subset of Dolphin18K which originally contains 18,460 problems and 5,871 templates with one or multiple equations. It contains 115 problems with single operator and 6,955 problems with multiple operators.

**8) Math23K [14]** - The dataset contains Chinese math word problems for elementary school students and is crawled from multiple online education websites. Initially, 60,000 problems with only one unknown variable are collected.

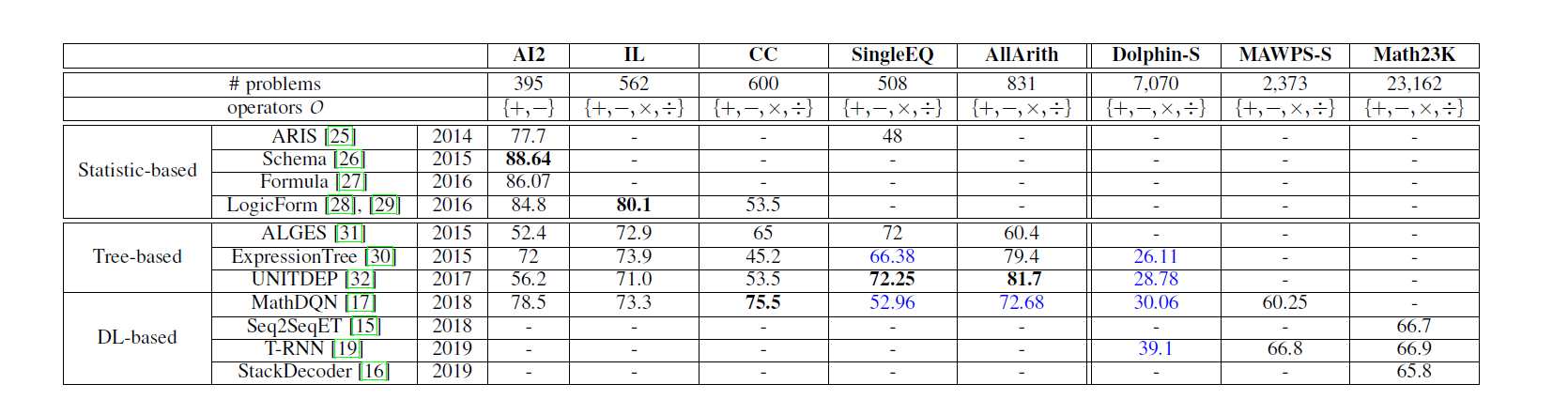
****

Figure Accuracy of statistic-based and tree-based methods in solving arithmetic problems.

# Chapter 3

# Technology Adopted

## 3.1 Introduction

This chapter briefly discusses various technologies that are adapted to implement the modules and why they are chosen.

## 3.2 Technologies Adopted for Implementation

### 3.2.1 Broad Technologies

* Image Processing – Usage of various computer algorithms to perform image processing tasks such as classification, feature extraction and pattern recognition. Image processing techniques are used to recognize the handwritten answers to the algebraic word problems.
* Natural Language Processing – Usage of various algorithms (mainly mathematical) to process human language. Natural Language Processing is used to process the algebraic word questions and generate answers based on semantic analysis and named entity recognition.

### 3.2.2 Programming Languages

Python programming language will be the core language used to develop the system. The main reason to implement the system using Python is the support for numerous standard libraries in mathematical, natural language processing and image processing contexts.

### 3.2.3 Development Tools

* OpenCV - OpenCV is an open source cross platform computer vision library that greatly eases image processing tasks. Since the handwriting recognition module is relying on image processing, OpenCV library is used
* Scikit Learn - A machine learning library that will be used to for classification algorithms. This is used to evaluate the type classification of equations.
* NLTK (Natural Language Toolkit) – A collection of libraries that supports Natural Language Processing using Python Programming. NLTK is a free and open source community driven project that contains text processing libraries for tokenization, stemming, tagging, parsing, semantic reasoning and classification which will be used to implement the module that processes algebraic word questions.
* Stanford Core NLP
* UCSC Lexical Resources – A collection of corpuses/datasets related to Sinhala language put together by Language Research Lab at University of Colombo.
* Jet Brains PyCharm is used as an IDE for development because of its user-friendliness.
* Pandas - Pandas is a software library written for the Python programming language for data manipulation and analysis. Its fast and efficient DataFrame object, reading and writing data tools, etc. are used for the development of the Domain Detection Component.
* NumPy - NumPy is a library for the Python programming language. It adds support for large, multi-dimensional arrays and matrices with a large collection of high-level mathematical functions to operate on these arrays [42]. As the Domain Detection is with highly sparsed multi-dimensional vectors NumPy is used for easy manipulation of such vectors.

### 3.2.4 Version Controlling System

GitHub is as a distributed version controlling, collaboration and source code management tool. GitHub will be used because of its familiarity.

## 3.3 Summary

This chapter briefly discussed the technologies adapted to develop the system along with the reasons for choosing them. The next chapter will discuss about the approach taken to develop the system.

# Chapter 4

# Approach

## 4.1 Introduction

This chapter discusses how the technologies mentioned in the previous chapter are adopted to tackle the challenges of the proposed system.

## 4.2 Overall System

The proposed solution tackles Automated Evaluation of Algebraic Word Problems using two major components. The components are divided according to the main two broad technologies that are used to tackle the main challenges.

1. Handwritten Optical Character Recognition module
2. Algebraic word problem analyzer and relevant answer generating module

## 4.3 Handwritten Optical Character Recognition module

The handwritten optical character recognition module uses Image Processing algorithms to tackle the challenge of recognizing the handwritten answers to algebraic word problems.

### 4.3.1 Users

This module will be used by the persons who need to evaluate their handwritten answers to a given algebraic word problem.

### 4.3.2 Input

Answers to algebraic word problems are usually written on singled ruled or square ruled pages. Images of these answers obtained by means of scanning or by camera are the inputs to this module. An example input image is illustrated in Figure 7.

A close up of text on a whiteboard

Description automatically generated

Figure Example input image to Handwritten OCR module

### 4.3.3 Output

The Handwritten OCR module processes the uploaded image and outputs machine understandable character lines. For an example, for the above input, the following character lines are generated ( Figure 8).

1. i. x + y + z

ii. 5x + 5y + 5z

iii. y – z

iv. x + y

Figure Example output text from Handwritten OCR module

### 4.3.4 Scope limitations

The Handwritten OCR module’s scope is limited to recognizing maximum of two touching or two overlapping characters. Most of the mathematical documents we have observed usually have around maximum of two-three touching or overlapping characters.

### 4.3.5 Process

The following process is used to transform the input image into the output text.

* The input image is first pre-processed to remove inconsistencies and is applied various operations to make the image suitable for later stages in the process.
* Then the image is segmented to identify the written lines of interest.
* Individual characters are then segmented by analyzing the segmented lines.
* Classification of characters using machine learning algorithm

## 4.4 Algebraic word problem Analyzer.

This NLP module will process the question an identify the entities, variables, operators and parameters. Process in simplifying the question to represent information in a formal way which can be understand by machines.

### 4.4.1 Users

Users who need to evaluate their answers, will upload the question to the system.

### 4.4.2 Input

Algebraic word problems can be found in grade 6,7,8 Math textbooks. Sinhala questions are uploaded to the system as string which can be extracted from PDF. An example of input Sinhala algebraic question is shown below.

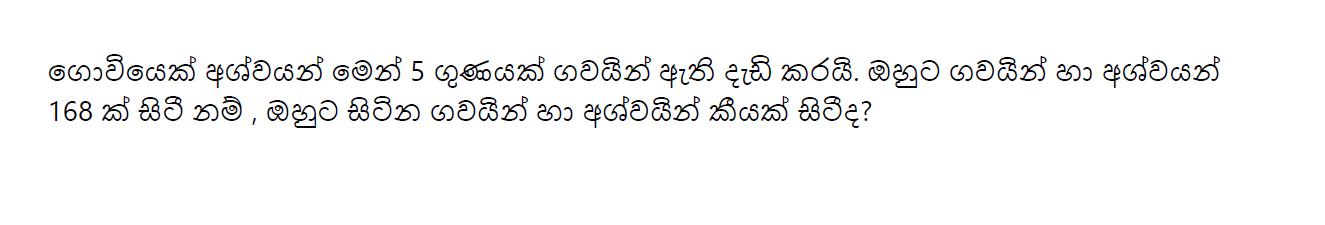


Figure Example of problem sentence

### 4.4.3 Output

NLP module process the questions and recognize the entities, variables, that are needed to build the equation. Output of the module will be given as semantic representation of the entities with the associated quantities.

### 4.4.4 Process

Steps of identifying entities, variables, from a given question will be as follows.

* Input problem text is analyzed to separate different sentence and phrases in the text.
* Preprocessing the text with stemming and filtering.
* Applying POS (Part-of-speech) tagging which represent the grammatical meaning of words.
* Extract entities and values that are needed to build the equation.

## 4.5 Answer generating Module

This module mainly consists of mapping the identified entities and quantities to an equation template. Process basically involves two steps which are selecting a suitable template structure and Instantiate with the numbers from the text.

### 4.5.1 Input

Output of the previous NLP module will be the input to module. The entities and numbers that are given by the feature extraction module will be the input for this system.

### 4.5.2 Output

Output of the system will be the answer for the question entered to the system. Answer will be generated through mapping the quantities to equation templates. An example of an equation template is shown below.

u = An unknown slot

n = A number slot

***u11 + u21− n1 = 0***

***n2 × u21+ n3 × u22− n4 = 0***

### 4.5.3 Process

* Supervise training with problems and relative answer equations with different templates.
* Identify candidate templates that matches to given problem.
* Using template rules and generate the equations.
* Smoothing module is used to clarify and remove noise from equations.
* Generate the answer.

# Chapter 5

# Methodology

## 5.1 Introduction

This chapter discusses in detail the design of the proposed solution. The top-level architecture of the proposed solution and the architectures of individual modules are discussed in this chapter.

## 5.2 High-level Architecture of the Overall System

The implementation of the solution was carried out in major 3 stages, Handwritten Optical Character Recognition module which identifies written answers submitted as images, Preprocessing module which extracted and Answer generating module.

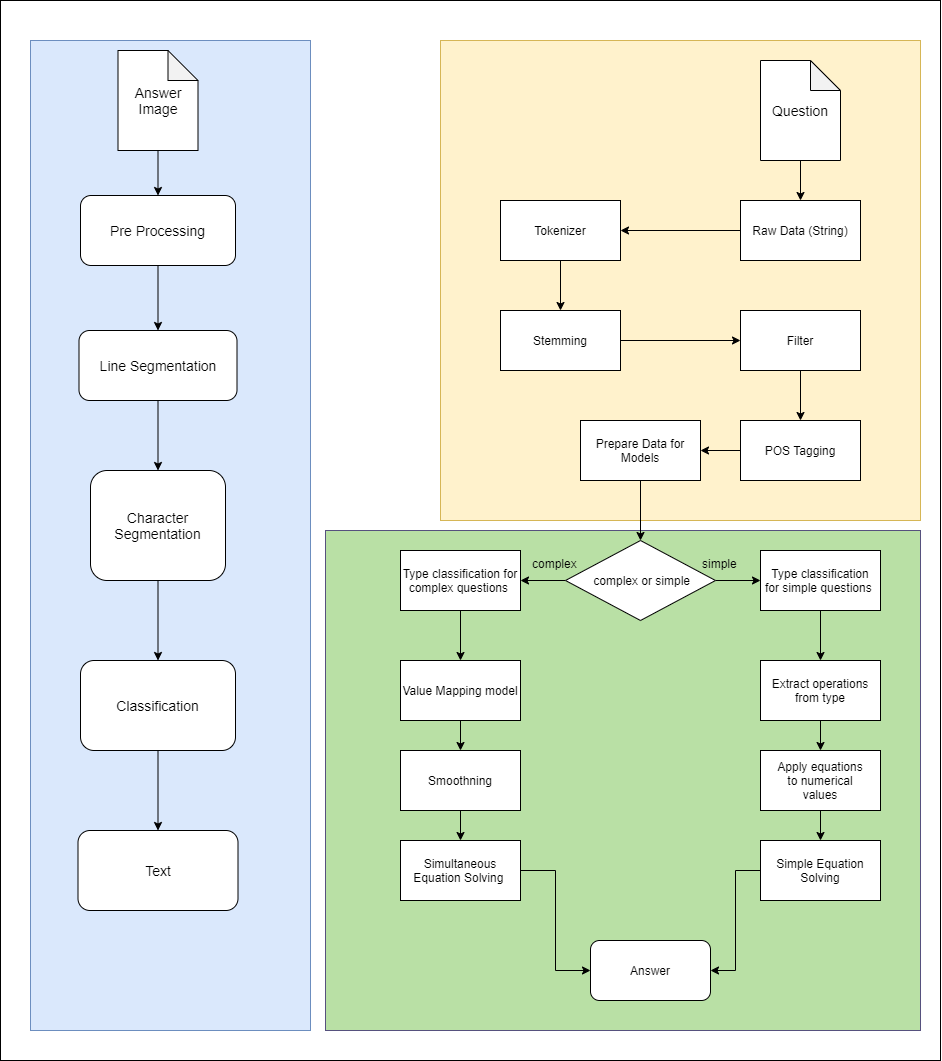


Figure High level architecture of the overall System

## 5.3 Handwritten Optical Character Recognition module

Figure 11 depicts the high-level architecture of the Handwritten Optical Character Recognition module that is used to process and extract the handwritten answers.

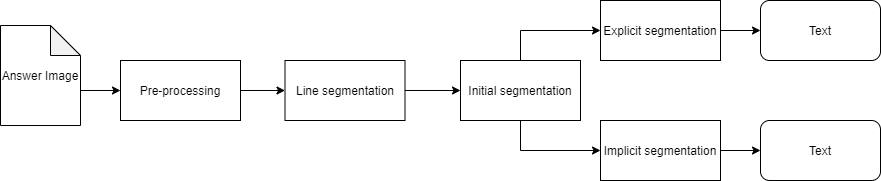


Figure High level architecture of Handwritten OCR Module

The input to this module is an image of a handwritten answer to an algebraic word problem (Ex - Figure 7).

### 5.3.1 Pre-Processing

The image is first resized because there can be different spatial resolutions in images depending on their capturing devices. Since the image can consist of background and writings of different color, the image is converted into Gray Scale. Gaussian blurring is then applied to remove noise and smoothen the image. Then the Blackhat morphological operator is applied to enhance dark objects against light background. Finally, thresholding is applied to separate the background from foreground (written text).

|  |  |
| --- | --- |
| A close up of a door  Description automatically generated  (a) | A close up of a whiteboard  Description automatically generated  (b) |
| A close up of a screen  Description automatically generated  (c) | A close up of a logo  Description automatically generated  (d) |

Figure Handwritten OCR Pre Processing steps (a) Input Image (b)Gaussian Blurring and Gray Level Transform (c)Black Hat Transform (d)OTSU Thresholding

### 5.3.2 Line Segmentation

Line segmentation is the process of differentiating individual lines of handwritten text in a given document.

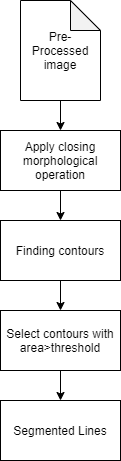


Figure Line Segmentation Design

The preprocessed image is the input to this module and the closing morphological operation is applied to smear the text lines into blobs. Then the contours are retrieved using OpenCV. Since there can be small irrelevant blobs, they need to be filtered using calculating the contour area and selecting blobs with an area larger than a threshold.

|  |  |
| --- | --- |
| A close up of a logo  Description automatically generated(a) | A picture containing object  Description automatically generated  (b)  A picture containing device, gauge, object  Description automatically generated  (c)    (d)  A picture containing device, gauge, object  Description automatically generated  (e) |

Figure Line Segmentation Process (a)After applying closing operation (b)-(e)Detected Lines

#### 5.3.3 Initial segmentation

In initial segmentation, lines of text are segmented into single characters, touching characters or overlapping characters. Due to scope constraints only two touching characters or two overlapping characters are considered.

A picture containing screenshot

Description automatically generated

Figure Initial character segmentation

The segmented lines act as the input of this module. The vertical projection, that is the count of foreground pixels in each vertical line is then calculated. This allows us to easily segment characters that are separated by spaces. Thus, leaving us with single characters, two touching characters or two overlapping characters. These images are then recognized using two methods, implicit segmentation and explicit segmentation. Both methods will be tried in this thesis and the method with the best accuracy will be chosen for the final system.

|  |  |
| --- | --- |
| A picture containing object  Description automatically generated  (a) | A picture containing clipart  Description automatically generated    A picture containing object  Description automatically generated  (b) |
| A picture containing device, gauge, object  Description automatically generated  (c) | A close up of a logo  Description automatically generated   A close up of a logo  Description automatically generated  A picture containing object  Description automatically generated  (d) |

Figure Segmented Characters from lines

#### 5.3.3.2 Implicit segmentation

In implicit segmentation character segmentation and character recognition is achieved simultaneously. This can be achieved using a Convolutional Neural Network. A dataset containing single characters, two touching characters and two overlapping characters will have to be generated as none such datasets were available online. It’s important to note that the scope of this thesis is limited to segmenting maximum of two touching or two overlapping characters.

1. **Generation of the dataset**

The dataset will be generated using the EMNIST dataset (for 52 characters and 10 numbers) and the CROHME dataset (for 10 mathematical symbols). The CROHME dataset is a dataset of Online handwritten characters. However, through offline\_CROHME [21] tool it’s possible to convert the online handwritten characters to offline character images. Only 10 mathematical symbols are chosen from the CROHME dataset as they are the most basic math symbols. The symbols are – (, ), +, -, /, =, [, ], division, \*. The whole merged dataset is converted to binary colors because the pre-processing module of the OCR turns all images to binary colors using OTSU thresholding and the mathematical symbols of the CROHME dataset is in binary color. Thus, the grayscale color depth of EMNIST is sacrificed.

Images are generated by randomly taking a character from the merged dataset and combining it with a another randomly taken character from the merged dataset. The merged dataset (EMNIST + CROHME) should be balanced such that each character class (out of 72) has the equal number of images. Thus, will allow us to generate a dataset that is more balanced, as characters are taken from the merged dataset at random.

Characters are merged by placing the second character where the first character ends. The characters are merged such that some images are touching characters and some images are overlapping characters. This variation between overlapping and touching characters can be achieved by randomly shifting (by around 3 pixels) where the second character is placed.

Each generated image will have two labels. First label for the first character of the image and the second label for the second character in the image. For single character images the first label will contain the character in the image and the second label will contain a number to represent a blank character (an extra character). The combined dataset has 72 (EMNIST+CROHME) types of characters and thus the generated dataset has 5256 (72x73) possible character combinations/classes. Each combination will have at least 600 images thus in total there are around 3 153 600 images (600x5256). The exact number of images in the generated dataset is 3 230 980. Each image in the dataset is sized at 28x56 (height x width) and is in binary colors. The following figure shows some samples of the generated images and their labels.

|  |  |  |
| --- | --- | --- |
| (y, d) | (P, 72) | (6, h) |
| (+, s) | (=, 72) | (E, N) |

Figure Samples and their labels

1. **Training the model**

The generated dataset will be trained on a 4-layer CNN, 6-layer CNN and on ResNet-50. ResNet is an abbreviation for Residual Network. ResNet allows training of extremely deep neural networks by allowing shortcut connections which allows to mitigate the problem of gradient vanishing. ResNet-50 has 50 convolutional layers with shortcut connections.

The problem can be addressed as a multi label classification or a multi class classification.

If the problem is addressed as a multi label classification, the output of the NN will have 145 (72+73) outputs and each image will have two labels (ex: - 0, 8)

If the problem is addressed as a multi class classification, the NN will have 5256 (72x73) outputs. Each class will denote a one combination between two characters.

Train test split of 90/10 is used to train and evaluate the model.

#### 5.3.3.3 Explicit segmentation

In explicit segmentation two touching characters or two overlapping characters must be manually segmented using rule-based methods.

A close up of a map

Description automatically generated

Figure Explicit Segmentation overall architecture

First, a classification must be performed to recognize weather the image is of a single character or two touching/overlapping characters. This will be tried in two different ways in this thesis.

1. By clustering –With the previous output, the width and the white pixel count of each image is calculated. The following figure shows the distribution of image width (x axis) against image white pixel count (y axis).

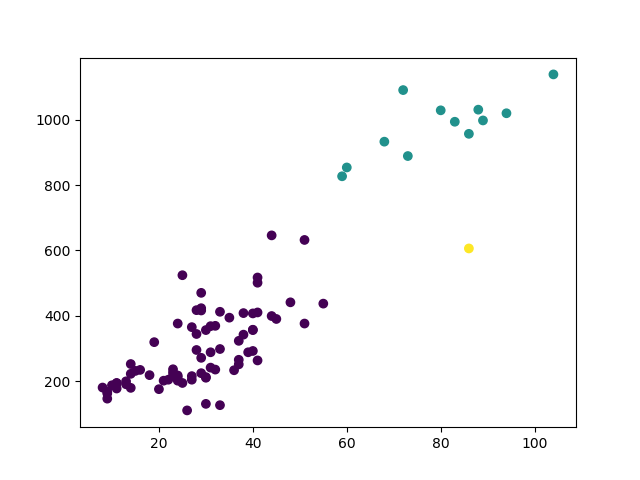


Figure Scatter plot of image widths(x) and image white pixel counts(y)

Through hierarchical clustering of these values, it is possible to classify characters as single characters (purple colored) and touching/overlapping characters (turquoise colored). The cluster (turquoise colored) of the image with the largest white pixel count will be chosen as the touching/overlapping character cluster. The rest will be recognized as single characters.

1. By training a CNN – This classification can be addressed as a binary classification problem where a single character is classified as 0 and touching/overlapping characters classified as 1. Since there are no such public datasets available for this purpose a dataset was generated. This dataset was generated in similar fashion to the dataset generated in the implicit segmentation section. Instead of 5256 possible classes used in the implicit segmentation dataset this dataset has two (binary) classes. The dataset contains 318 672 images where 50% of the images are single images and the rest is touching/overlapping character images.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

Figure Samples of the generated dataset

A CNN with 8 Convolutional Layers will be used to perform the classification.

If the image is recognized as a single character, it can be directly sent to the character classifier to be classified. If the image is of touching/overlapping characters, it will have to be further processed to segment into two characters.

The first step in segmenting the image is to perform background thinning of the input image. The following images are of input images and the background thinned images of the input.

|  |  |
| --- | --- |
| (a) | (b) |
| (c) | (d) |

Figure (a,c)Touching/Overlapping Character Image (b,d)Background thinned image

The next step is to separate overlapping characters from touching characters. This can be achieved by checking the contours of the background image. If there are two separable contours in the image, the image is recognized as two overlapping characters (d image above). Thus, they can be easily segmented by creating masks of the contours. Then the single character images are sent to the character classifier to be classified. This process is described in detail under the Splitting characters using contours section below.

If the image doesn’t contain two separable contours, it’s taken as a touching character image (b image above). In such images, the background thinned image is divided into 3 sections along the x axis (width wise) and the middle section is chosen for further processing (red area in the c image below). This algorithm assumes that the candidate segmentation line between the characters is situated in the before mentioned middle section. Then a kernel is applied to find candidate points in the upper part of the background skeleton (bounded by red in d,e image below). Another kernel is applied to find candidate points in the lower part of the background skeleton (bounded by green in d,e image below). These points are then connected to form a segmentation line (f image below).

|  |  |
| --- | --- |
| (a) | (b) |
| (c) | (d) |
| (e) | (f) |

Figure (a)Touching character image (b)Background thinned image (c)The middle section outlined in red (d) Candidate points found (Red- candidate point in upper part, Green- Candidate point in lower part) (e)Candidate points in the background image (d)Final image with connected candidate points

The above figure demonstrates how an input image (a) is processed to find the candidate points that are to be connected. Which in turn gives the segmentation line upon connecting the points (f image above).

In order to find the candidate point, in the upper part of the background thinned image, the following convolution kernel can be used.

|  |  |  |
| --- | --- | --- |
| 2 | 2 | 2 |
| -10 | 10 | -10 |
| -10 | -10 | -10 |

Figure Convolution kernel to find candidate points in the upper part

Checking this convolution against a threshold of 3060 (255x10 + 255x2) allows us to find following types of candidate points.

|  |  |  |
| --- | --- | --- |
| (a) | (b) | (c) |

Figure Types of candidate points found by the kernel

In order to find the candidate point in the lower part of the background thinned image, the following convolution kernel can be used.

|  |  |  |
| --- | --- | --- |
| -10 | -10 | -10 |
| -10 | 10 | -10 |
| 2 | 2 | 2 |

Figure Convolution kernel to find candidate points in the lower part

Checking this convolution against a threshold of 3060 (255x10 + 255x2) allows us to find following types of candidate points.

|  |  |  |
| --- | --- | --- |
| (a) | (b) | (c) |

Figure Types of candidate points found by the kernel

If no candidate points can be found, then the image is not segment able by the current algorithm.

The most trivial case is when there is only one candidate point each per upper and lower part of the skeleton. They can be directly connected to form the segmentation line. However, a choice must be made in case there are more than one point in each upper and lower parts of the skeleton. Points (x, y) found on the upper part of the skeleton will be referred to as top points and points (x, y) found on the lower part of the skeleton will be referred to as bottom points from here onwards.

The following algorithm can be used to find the candidate connecting points in cases where there are more than one top point or bottom points.

Inputs – Top points, bottom points and middle section of the background skeleton

Output – Weather connecting points were found (Boolean) and the connecting points

1. *boolean, arrayOfPoints getConnectingPoints(topPoints, bottomPoints, middle):*
2. *connectingPoints = [ ] 🡨array to keep the two connecting points between the upper contour and the lower contour*
3. *height = middle.shape[0]*
4. *if length(topPoints) == 1 and length(bottomPoints) == 1*
5. *connectingPoints.add(topPoints[0], bottomPoints[0])*
6. *else if length(topPoints) == 0 and length(bottomPoints) != 0*
7. *connectingPoints.add([bottomPoints [0][0],0], bottomPoints[0])*
8. *else if length(topPoints) != 0 and length(bottomPoints) == 0*
9. *sorted\_topPoints = sortOnY(topPoints) 🡨sort points such that points whose Y values are largest (closest to the bottom) are at the start of the array*
10. *connectingPoints.add(sorted\_topPoints[0], [sorted\_topPoints[0][0], height])*
11. *else if length(topPoints) > 1 and length(bottomPoints) == 1*
12. *distances = [ ] 🡨array to hold the distances between the top points and bottom point*
13. *for each topPoint in topPoints*
14. *y = abs(topPoint[1] - btmPoints[0][1])*
15. *x = abs(topPoint[0] - btmPoints[0][0])*
16. *distances.add(x+y)*
17. *end for*
18. *min\_idx = getIndexOfMinValue(distances) 🡨get the index of the minimum distance value in the array*
19. *connectingPoints.add(topPoints[min\_idx], bottomPoints[0])*
20. *else*
21. *return false, [ ]*
22. *end if*
23. *return True, connectingPoints*

According to the above algorithm, if there are no top points and there is at least one bottom point, the bottom point’s X value is chosen as the top point’s X value and 0 will be chosen as the Y value.

If there are is least one top point and no bottom points, the top point’s X value is chosen as the bottom points X value and the last row of the image is chosen as the bottom points Y value. In case there are multiple top points, the points are sorted such that the points whose Y values are the largest (closest to the bottom of the image) are at the start of the array and the first point is chosen as the top point.

If there are multiple top points and only one bottom point, the distances between the bottom point and the top points are calculated. The top point that gives the lowest distance to the bottom point is chosen as the top point.

Through connecting the top point and the bottom point a segmentation line can be drawn and thus the image can be segmented to two characters using contour hierarchy method.

1. **Splitting characters using contours (contour hierarchy)**

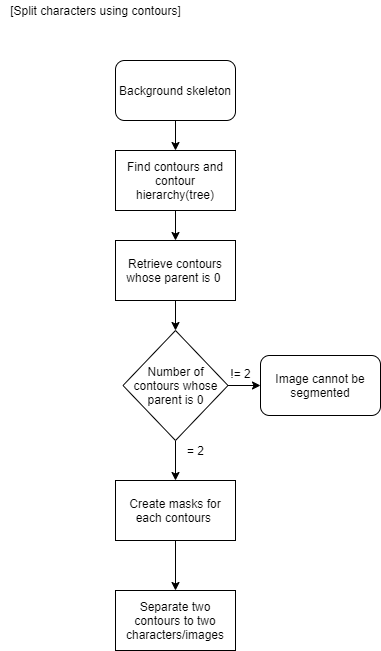


Figure Flowchart for splitting characters using contour hierarchy

If the background skeleton is separated clearly as the following figure (b, d) contour hierarchy can be used to split the image into two characters.

|  |  |
| --- | --- |
| (a) | (b) |
| (c) | (d) |

Figure Images that can be split using contour hierarchy

The first step of the process is to find the contour hierarchies of the background skeleton as a tree. This retrieves the contours and creates a full family hierarchy list. The contour that encircles the whole region of interest is the contour at the top of the hierarchy (parent) and it will be assigned the value 0. The following figure shows the contour hierarchy where (a, e) images are of the background skeleton, (b, f) images are of the parent contour (rank 0) of the background skeleton and (c, d, g, h) images are the child contours of the parent (rank 0) contour.

|  |  |  |  |
| --- | --- | --- | --- |
| (a) | (b) | (c) | (d) |
| (e) | (f) | (g) | (h) |

Figure Contour hierarchy

If there are two child contours under the rank 0 parent contour, then the image can be segmented to two-character images. In case of overlapping characters, the image is segmented along the space between the characters. In case of touching characters, the background skeleton should be sent with the candidate segmentation line (image e).

If two child contours under the rank 0 contour can’t be found, then this algorithm marks the image as being not segmentable.

### 5.3.4 Character Classification

In this module single characters are classified to their respective labels. The inputs are the single characters in the document or the segmented characters. The dataset will be a combination of the EMNIST dataset and the CROHME dataset with a total of 72 characters. These 72 characters are made up of the characters of the alphabet (A-Z and a-z), numbers (0-9) and mathematical symbols ((, ), +, -, /, =, [, ], division, \*.). The CROHME dataset is a dataset of Online handwritten characters. However, through offline\_CROHME [21] tool it’s possible to convert the online handwritten characters to offline character images. The whole dataset will be pre-processed into binary colors because the pre-processing module of the OCR turns the document to binary colors using OTSU thresholding and the offline CROHME dataset is available only in binary color. The dataset will be trained on 4-layer CNN, 6-layer CNN and on ResNet50. The merged dataset has 880 709 training images and 140 862 test images.

## 5.4 NLP Preprocessing Module.

Sinhala is one of the several morphologically rich languages for which currently there are no completed repositories such as WordNet or Subjective lexicons. Sinhala has two varieties: literal and spoken forms. In this research, we are focusing Sinhala problems which are common in grade 6 ,7, 8 textbooks

Preprocessing is applied to the input question to prepare it for later operations.

A screenshot of a cell phone

Description automatically generated

Figure Preprocessing Module

Preprocessing is the tasks carried out on raw data before it is inputted to the main processing flow. Below tasks are identified as the steps in the text preprocessing component which provides required outcomes for the rest of the modules which are shown in Figure 18.

* Extracting Sentences from a Source
* Tokenizing Process
* Stemming Process
* Filtering Process
  + Stop words removal
  + Unnecessary Characters removal
* Part of Speech Tagging
* Prepare and Fetch data for classification models
  + Feature Extraction
  + Noise words Removal
  + Extract Numerical Values

### 5.4.1 Raw Data

When the question is uploaded in PDF format, the question needs to be extracted from the PDF as a string. That string will be proceeded through the module. The main challenge was Sinhala chars are given in UTF-8 or in UTF-16 format. Therefore, when extracting these data this module has to consider and translate data into one of those formats.

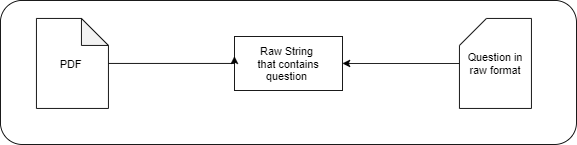


Figure PDF Extraction or get raw data string from given source

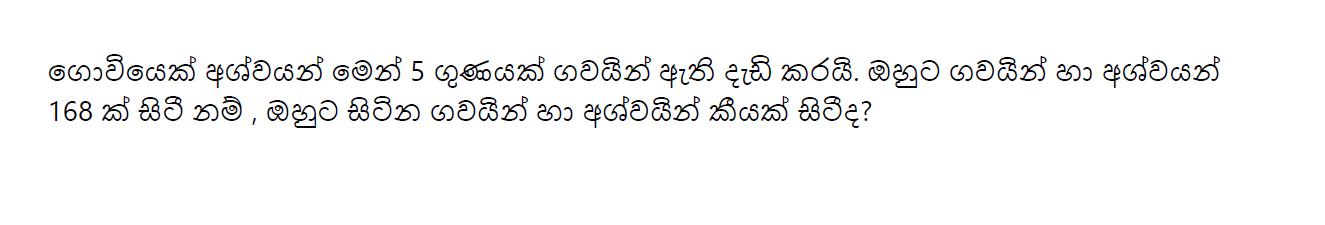


Figure Raw String with Question

### 5.4.2 Tokenizer

Tokenization is the process of breaking a stream of text into words, phrases, symbols, or other meaningful elements called tokens. The purpose of the tokenization is the further study of the words in a sentence. The list of tokens becomes input for advance processing, for example parsing or text mining. Extracted sentence is split into small sentences, phrases and individual words. This process is called Tokenization. This process is done with the help of nltk libabries.

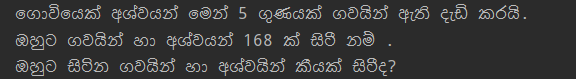


Figure 32 Tokenized Sentences

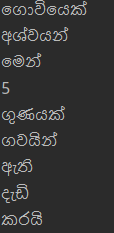
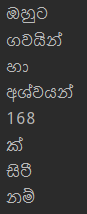
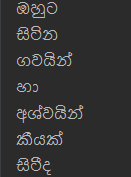
  

Figure 33 Tokenized words of each sentence in the question

### 5.4.3 Stemming

The theory behind stemming is the transformation of morphological forms of a word to its stem by assuming each one is semantically associated. Two points are considered when using a stemmer:

- Words should be kept separate, if they do not have the same meaning

- Morphological forms of a word are assumed to have the same base meaning and therefore it should be mapped to the same stem

In text mining or language processing applications, these two rules are considered good and enough. For languages with comparatively simple morphology, the influence of stemming is less than for the languages with a more complex morphology. Most of the stemming researches done so far are in English and European languages. Stemming is performed to get the root words. For this module two techniques were tried to do the proper stemming. They are using word frequency and stemming word dictionary.

Lemmatization is the process of transforms a word to its actual root word in proper way not by just chopping off the end of the word. Since lemmatization was bit complex and not essential for our module, the stemming was the technique chosen. Next step in the preprocessing is to normalize text. This process may not be much needed for this module as it won’t be contained the noisy words in the question text.

|  |  |
| --- | --- |
| **Stemming** | **Lemmatization** |
| A stem is the part of the word that doesn’t change even when morphologically inflected | A lemma is the base form of the verb |
| Stemmers are normally easier to implement and run faster | Since lemmatizes are related to the semantics and the POS of a sentence, they are difficult to implement |
| A crude heuristic process that cuts off the ends of words | Done in a proper manner with the use of a vocabulary and morphological analysis of words, generally aiming to eliminate inflectional endings only and to return the lemma. |
| Results are not morphologically right forms of words all the time | Comparatively accurate results are received. |
| “gone, going, goes” is mapped to “go”  but “went” won’t map with “go” | “gone, going, goes, went” mapped accurately |

*Table 3: Comparison of Stemming and Lemmatization*

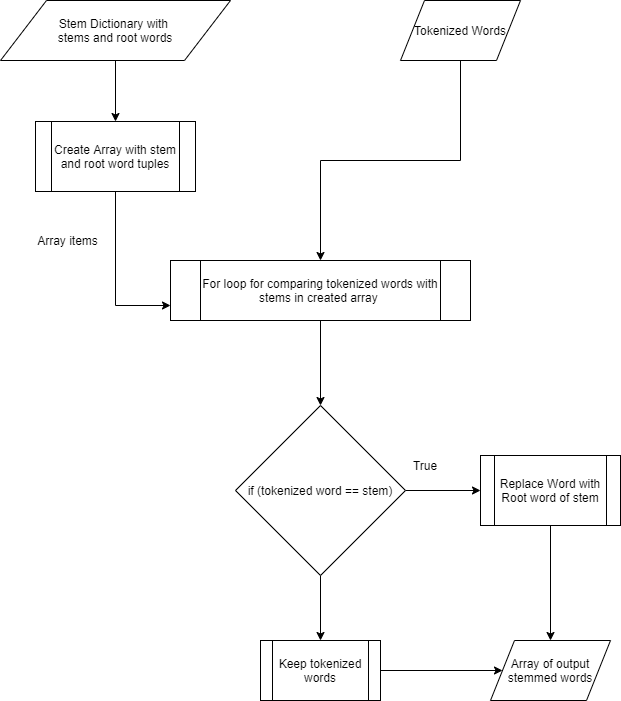


Figure 34 Flow chart of Stemmed Process

A close up of a logo

Description automatically generated

Figure 34 Flow chart of Stemmed Process

### 5.4.4 Filtering

This process is used to filter the words by removing unnecessary words such as stop words and chars. A stop word dictionary has been prepared to recognize the stop words in a question. This will output only the necessary words that are needed.

Stop words are a part of natural language. The reason to eliminate stop words from a text is

that they make the text look heavier and less significant for analysis purposes. Eliminating stop words lessen the dimensionality of term space. The most common words in text documents are articles, prepositions, and pro-nouns, etc. which does not give the meaning of the documents . After cleaning the data and converted into a one language, stop words removing can be done.

It was found that Sinhala is a language which satisfies Ziff’s Law behavior. Ziff’s law states

that, given a text corpus, if f: is word count and r: is rank, when sorted by word count that,

𝑓. 𝑟 ≈ 𝐶𝑜𝑛𝑠𝑡𝑎𝑛t

Which means that the most frequent word will occur approximately twice as often as the second

most frequent word, three times as often as the third most frequent word, etc.

Set of Stop words of Sinhala language is given below.

**බව , යි, ලෙස, හෝ, ඇත, නිසා, මෙම, ව, ගැන, විසින්, විට, එම, තුළ, කොට, යුතු**

But that was not much reliable with this system since domain of this system is mathematical related one. Therefore, referring to predefined set of stop words recorded in an external file, and use that file to create an array of stop words and eliminate words by referring to that array is the approach used in this system. Pseudocode for that process is given below.

In the filtering process, the system needs to eliminate unnecessary punctuations. Punctuations can provide grammatical context but for bag of words-based sentiment analysis punctuation does not add value. The most inclusive approach to pre-processing is simply to clean all text, including numbers, any markup (html) or tags, punctuation, special characters ($, %, &, etc), and extra white-space characters.

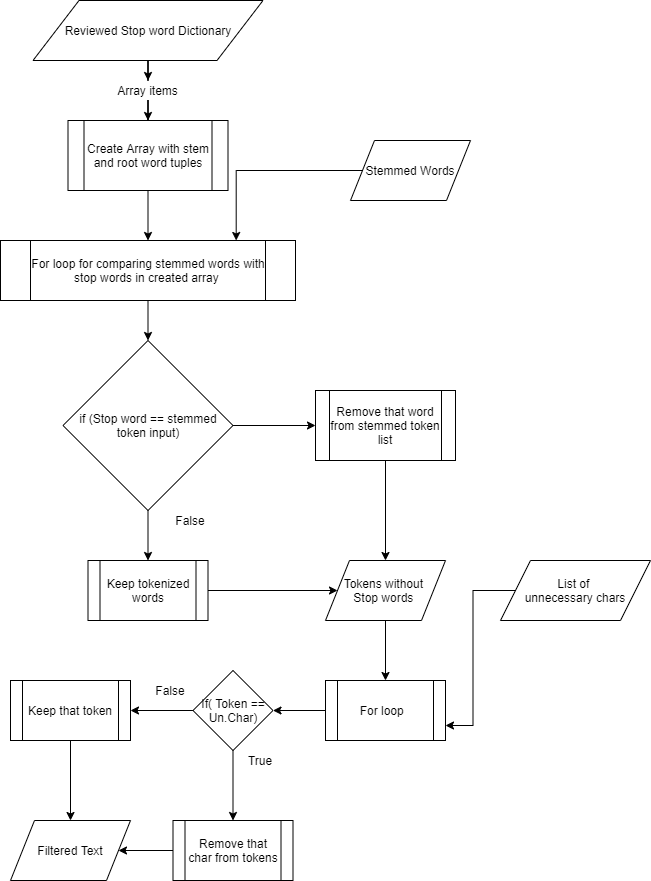


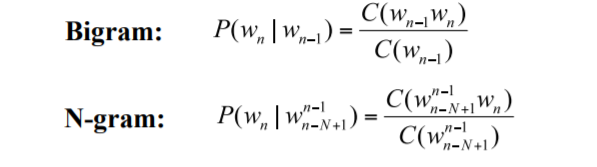
Figure Flow chart of Filtering Process

### 5.4.5 POS Tagging

Part-of-Speech tagging is standardizing the text using a formal representation. Word are annotated to recognize the form of a word such as verb, noun, adjective. There are several POS tagging techniques. Most frequently used ones are Rule based taggers and Stochastic taggers. A large database of hand-written disambiguation rules is used in a Rule based tagger. It entails more human effort as well as outstanding human expertise. A training corpus is used to compute the probability of a given word having a given tag in a given context in Stochastic taggers. HMM taggers, N-gram taggers and Naïve Bayes taggers are examples of stochastic techniques. In this system we have used the N-gram perceptron tagger and HMM tagger to get the most accurate one for the tagging process.

#### 5.4.5.1 Ngram Tagger

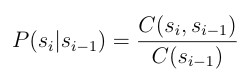
Ngram Tagger has 3 subclasses as Unigram tagger, Bigram tagger and Trigram tagger.



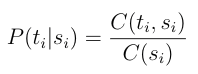
Idea of Ngram Tagger is generally following these instructions. By looking at the previous words and P-O-S tags, part-of-speech tag for the current word can be guessed. Each tagger maintains a context dictionary (Context Tagger parent class is used to implement it).This dictionary is used to guess that tag based on the context. The context is some number of previous tagged words in the case of Ngram Tagger subclasses. In this system this Ngram tagger is developed using NLTK libraries where it defines the perceptron tagger.

#### 5.4.5.2 Hidden Markov Model Tagger

A Markov model is a stochastic (probabilistic) model used to represent a system where future states depend only on the current state. For the purposes of POS tagging, we make the simplifying assumption that we can represent the Markov model using a finite state transition network. In HMM model-based POS tagging each POS tag and word should have transition probability and emission probability. The transition probabilities can be calculated using the maximum likelihood estimate.

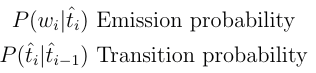


Emission probabilities can also be calculated using maximum likelihood estimates:

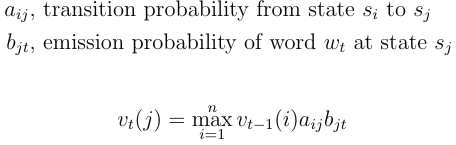


For likelihood estimations HMM models use “Forward Algorithm”. To find the sequence of POS tags with the highest probability given a sequence of words we use the forward algorithm.





The HMM forward algorithm gives us probabilities but what system wants is the actual sequence of tags. We need an algorithm that can give us the tag sequence with highest probability of being correct given a sequence of words. An intuitive algorithm for doing this, known as greedy decoding, goes and chooses the tag with the highest probability for each word without considering context such as subsequent tags. Viterbi is used to find the most likely sequence of states that led to this sequence.



With Viterbi algorithm, system is able to receive the most probabilistic sequence of POS tags for a given sentence.

#### 5.4.5.3 Corpus

To train the perceptron tagger and HMM tagger both, generally required tagged corpus with large number of words. Since this system is a mathematical related one, tagged dataset should be a domain specific data set. Therefor the dataset was manually created by our team referring USCS tagged corpuses.

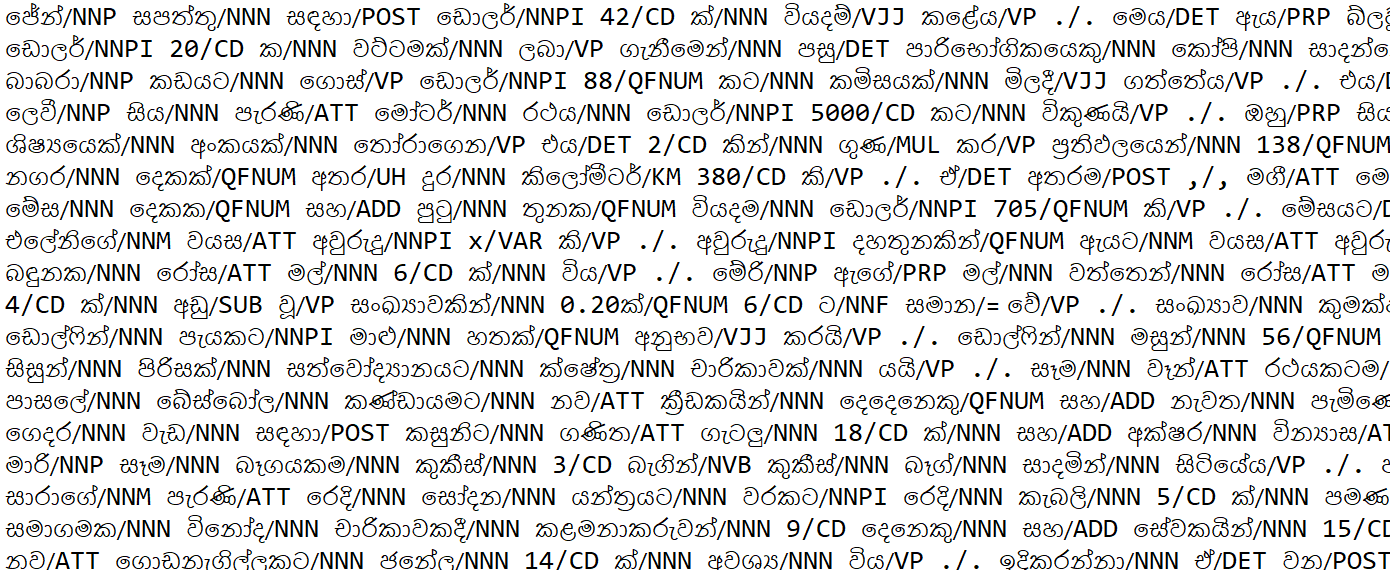


Figure POS tagged Corpus

Other than usually used well known tags such as NNP, DET, VP, JJ in this system new set of tags were introduced such as ATT, ADD, MUL, EQ to future feature extraction processing purposes. All the tags in our system given below.

|  |  |
| --- | --- |
| NN | Noun |
| NNP | Proper Noun |
| PRP | Pronoun |
| JJ | Adjective |
| VAUX | Verb Auxiliary |
| RB | Adverb |
| CC | Conjuncts |
| QFNUM | Number |
| VFM | Verb Finite math |
| VJJ | Verb Non-Finite Adjectival |
| NVB | Light verbs |
| SYM | Special |
| DIV | Division keywords |
| MUL | Multiplication Keywords |
| ADD | Addition Keywords |
| SUB | Subtraction Keywords |
| ATT | Attributes |
| EXP | Algebraic Expressions |
| QNUM | Question Numbers |
| EQ / = | Equivalence |
| CD | Numerical Value |
| VAR | Variable in English |

*Table 4: Tags and meanings (Colored ones are the newly introduced ones)*

The output of part-of-tagging module is received as given below.

Ex:



Figure POS tagged Output

### 5.4.6 Fetch Data for Models

The tagged problem is the input for this module and in this module number of processes are carried out. In this system an ANN is used for classification. Before the given input enters to the system number of feature extraction processes and data fetching processes are required to be performed on these data. That purposes are fulfilled with this module. Several steps are used to prepare the data. It Removes the unnecessary words such as “මහතා”, “මහත්මිය” etc. Also, it processes Sinhala value words such as “දෙගුණයක්”, “තෙගුණයක්” and “තිදෙනෙක්” etc.

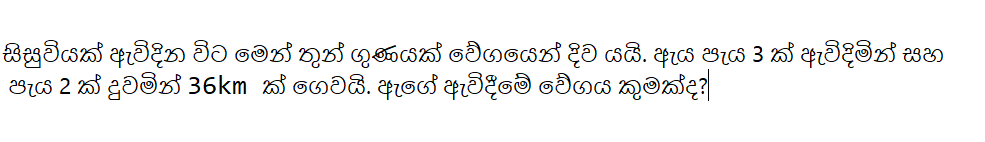


Figure A input with requirement to replace තෙගුණයක් with value 3

Then extract the numerical values from question. Then with the help of POS tags it gathers the keywords from the problem and create an array of word list.

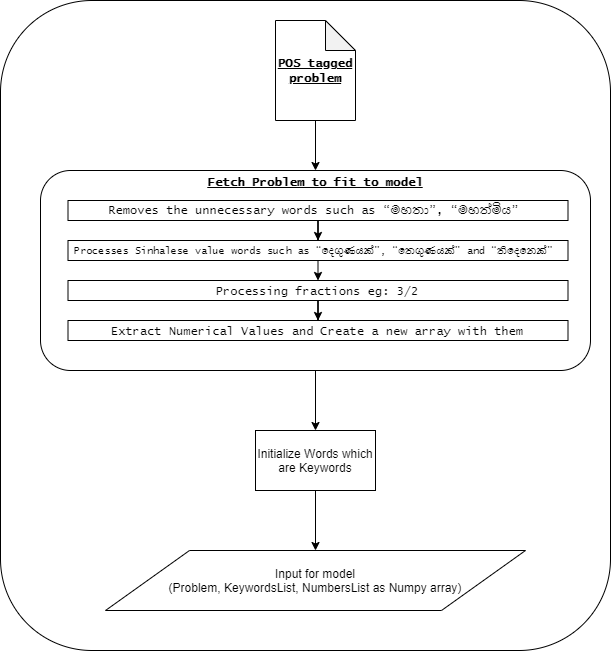


Figure Flow chart of feature extraction module.

A screenshot of a computer screen

Description automatically generated

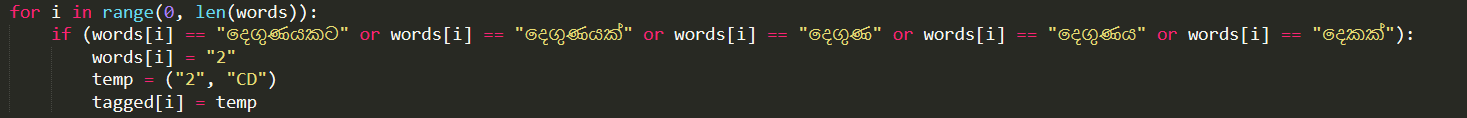


Figure 31 Implementation of Rules

## 5.5 Answer Generating Module

Answer generating module is the most challenging module that this system contained. This system can solve two type of equations. But in the GUI the user is supposed to provide whether the question is simple algebraic equation or simultaneous algebraic equation which is a limitation of this system.

* 1. ***Simple equations with single variable –*** Terms that have the same variable are said to be like terms, and you can add, subtract, multiply or divide them as if they were simple numbers. For example: The equation 2x + 3x is equal to 5x, simply 2 lots of x plus 3 lots of x to make 5 lots of x (5x).
  2. ***Simultaneous Equations with two variables -*** This is a process which involves removing or eliminating one of the unknowns to leave a single equation which involves the other unknown. The method is best illustrated by example. Example Solve the simultaneous equations

3x + 2y = 36

5x + 4y = 64

y = 6 Hence the full solution is x = 8, y = 6.

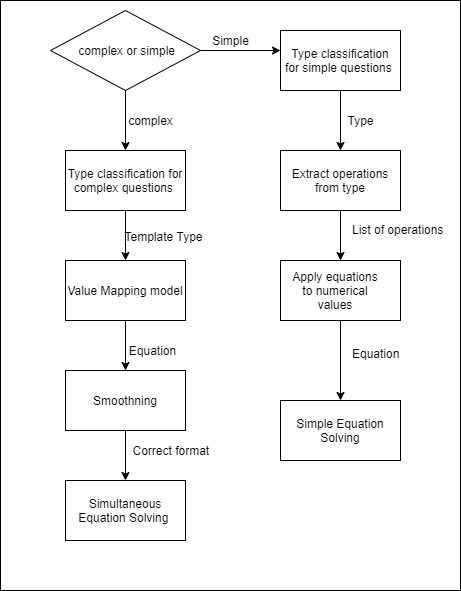


Figure - Answer Generating Module

### 5.5.1 Simple Equation Solving Module

Simple equation solving module solve Sinhala sentences which are algebraic equations with one variable. Since, these questions have one variable there is no need to maintain reference of variables but to maintain the reference of operations and values that operations should be performed.

* *කීත්ට පොත් 20 ක් ඇත. ජේසන්ට පොත් 21 ක් ඇත. ඔවුන් සතුව පොත් කීයක් තිබේද?*

*X = 20+21 = 41*

* *පියුමි දොඩම් ගෙඩි 50 ක් ගෙන සාරාට දොඩම් ගෙඩි 20 ක් දුන්නේය. පියුමිට දැන් දෙහි කීයක් තිබේද?*

*X = 50 – 20 = 30*

* *බෝට්ටු 5 ක් ගඟක ගමන් කරයි. සෑම බෝට්ටුවකම පුද්ගලයින් 3 දෙනෙකු සිටී. බෝට්ටුවල සිටින මුලු මිනිසුන් ගණන කීයද?*

*Y = 3 \* 5 = 15*

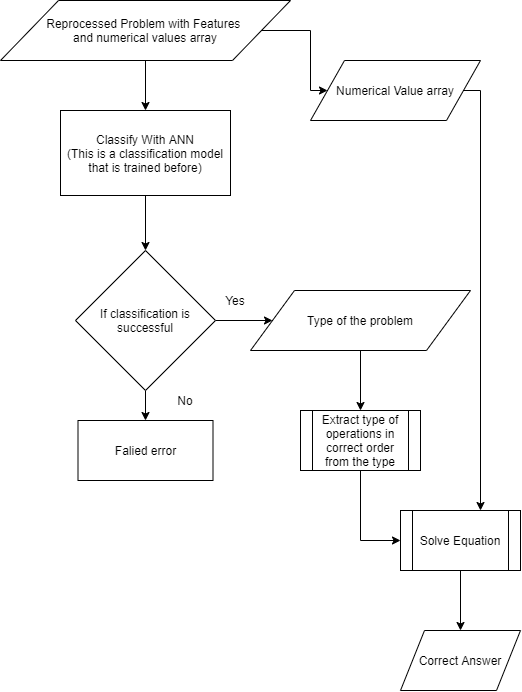


Figure - Flow chart of Simple Equation Solving Module

#### 5.5.1.1 Type Classification Model for Simple questions

There can be different types of simple algebraic arithmetic questions which contains one variable. In this module, it has the potential to solve problems which has 3 arithmetic operations in any combinations. In simple mathematics there are four type of arithmetic operations as Addition, Subtraction, Division and Multiplication. Therefore, there can be 64 types of simple type equations that this model has to identify. Some of those types and their identifiers given in following table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | Operation\_01 | Operation\_02 | Operation\_03 | Type Identifier |
| 01 | ADD | NULL | NULL | ADD/NULL/NULL |
| 02 | SUB | ADD | MUL | SUB/ADD/MUL |
| 03 | MUL | SUB | NULL | MUL/SUB/NULL |
| 04 | DIV | ADD | MUL | DIV/ADD/MUL |

*Table 5: Examples for types in simple questions module*

To get the relevant t question type when problem is given this module use Artificial Neural Network (ANN). An ANN is a computational model based on the structure and functions of biological neural networks. Information that flows through the network affects the structure of the ANN because a neural network changes - or learns, in a sense - based on that input and output. ANNs are considered nonlinear statistical data modeling tools where the complex relationships between inputs and outputs are modeled or patterns are found.

A screenshot of a cell phone screen with text

Description automatically generated

Figure - Summary of Classification ANN of Simple problems

After processing this fetched new problem in this ANN classification it will provide Type\_Identifier of the type of the question as shown in figure ().



Figure - Output of Type Classification Model for Simple questions

#### 5.5.1.2 Simple Equations Generating and Solving Module

After the process of previous model this model will receive type of the questions and list of numerical values in the same order that they are needed to process as inputs. Then those numerical values will properly be mapped into equation with operations and then it will solve the equation and will provide the result of the equation as the output.

1. Input - Numerical value array created from feature extraction module,

Type\_identifier of the problem type given from Classification Module

1. Output – Result of Equation
2. problemType <----------#Type\_identifier of type of problem
3. numbers <----------#numerical value array
4. ops <--------- list of opertaions extracted from Type\_identifier
5. temp1 <-------Result after first operation execution
6. temp2 <-------Result after second operation execution
7. temp3 <-------Result after third operation execution
8. operation <-------- Items in ops array
9. for operation in ops: <----------- Three iterations
10. if (operation == "ADD"):

Perform Addition

1. else if (operation == "SUB"):

Perform Subtraction

1. else if (operation == "MUL"):

Perform Multiplication

1. else if (operation == "DIV"):

Perform Division

1. else if (operation == "NULL"):

No action

1. end if
2. end for
3. result <------ Last result from equations
4. return result

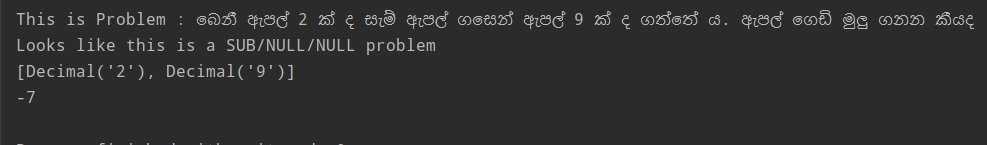


Figure - Output of Simple Equations Generating and Solving Module

### 5.5.2 Complex Simultaneous Equations Solving Module

Complex Simultaneous Equations Solving Module solve Sinhala sentences which are algebraic equations with two variables. Since, these questions have two variables it is very complex to process these types of questions because it requires to maintain reference of variables, how operations and applied. There several challenges involve when processing these types of questions.

* Sinhala is very ambiguous language and therefore there are multiple ways to represent same idea in Sinhala.
* Extract variables from the questions by differentiating them from other nouns and attributes.
* Identifying nouns with attributes such as රතු පාට මල්, කුඩා ඉටිපන්දම් , ගැහැණු ළමයි
* Maintain relationship between variables.
* Identifying operations given in the questions.
* Identifying Equivalencies and implications that implies equation in Sinhala languages and mapped that equal sign to correct position

To overcome these challenges our team has tried and came up with many solutions and there were two approaches we tried while implementing this module.

* **Rule Based approach** – Use an ANN classification module to predict the type of question and according to the type the question will go through a template which contains series of rules to generate the equation. In this approach extracting variables from the questions by differentiating them from other nouns and attributes , identifying nouns with attributes such as රතු පාට මල්, maintaining relationship between variables, identifying operations given in the questions, and identifying Equivalencies and implications that implies equation in Sinhala languages and mapped that equal sign to correct position all processes will be executed through the set of rules defined in the template according to its respective type. Then use smoothing techniques to get the required two equations and then solve that simultaneous equations and get the answers for variables.

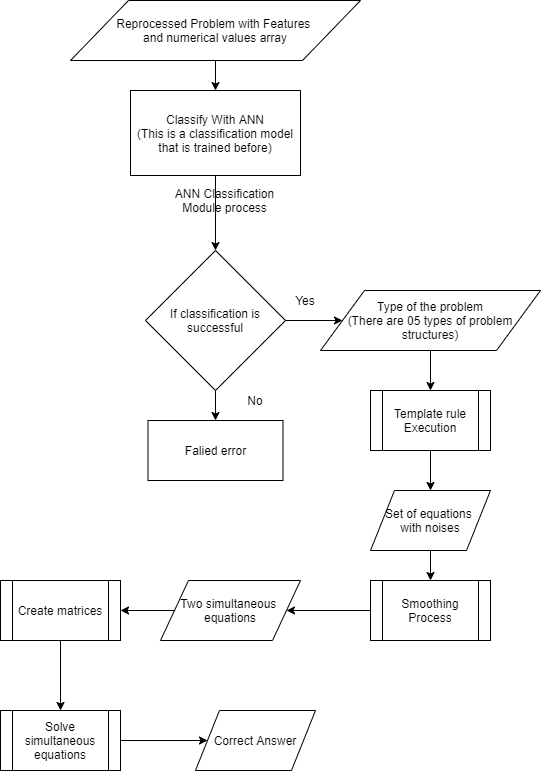


Figure - Flowchart of Rule Based approach of Complex Simultaneous Equations Solving Module

* **Dynamic Model Approach** – Use complex level of feature extraction module and gather all the variables and numerical values. For this task, dataset should contain the word Problems with correct math expressions. A problem can have set of equations. Equations are generalized to a template by replacing variables and numbers with unknown slots and numbers slots respectively. Numbers that are not available in the text are considered as constants. For a given problem T number of possible templates are generated from training data. Use more detailed dataset to train an ANN model, then use that ANN to get the equation as the input. In this approach extracting variables from the questions by differentiating them from other nouns and attributes , identifying nouns with attributes such as රතු පාට මල් , identifying operations given in the questions, and identifying Equivalencies and implications that implies equation in Sinhala languages and mapped that equal sign to correct position will be done by the ANN. Then use relationship extraction module to extract and maintain relationship between variables. After that use smoothing techniques to get the required two equations and then solve that simultaneous equations and get the answers for variables.

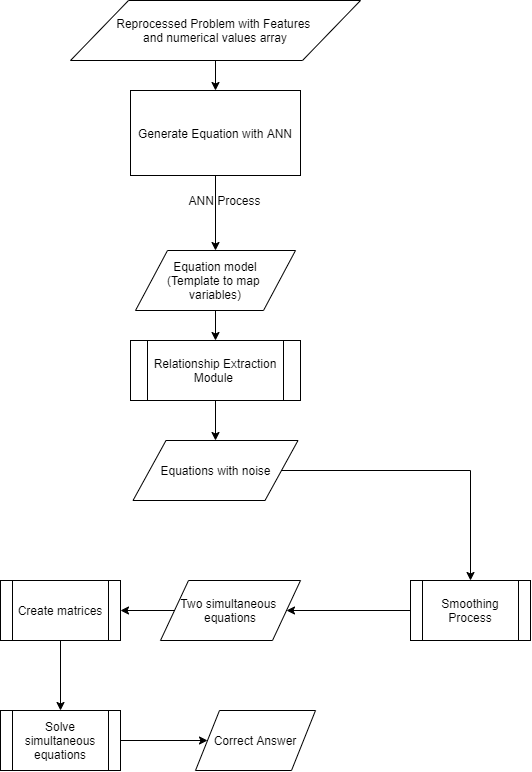


Figure - Flowchart of Dynamic approach of Complex Simultaneous Equations Solving Module

Our team had proceeded with both approaches and when we reached to relationship extraction module of dynamic approach the accuracy of the system was very law related to the rule-based approach. Since, Sinhala is a very law level and new language for Natural Language Processing domain there is no sufficient resources and knowledge exist yet to proceed with the dynamic approach. Therefor system was continued with the rule-based approach.

#### 5.5.2.1 Type Classification Model for Complex questions

Questions with single variable are categorized as simple and questions with two variables are categorized as complex questions. User will be instructed to insert the question type when uploading the question. According to the question type equations generating process will differ. Output that is given from the preprocessing module will be the input for the classification model.

This process includes supervised and semi-supervised learning with set of algebraic word problems and answers. This module will receive problem, features list which contains numerical values and keywords extracted from question in previous modules. This system can solve 5 types of sentence structures. Therefore, system need to have a module which is trained with those type of questions to accurately predict the most suitable sentence structure (Template) for a given problem.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Question | Type | Equation Template | | Equation | |
| **Equation\_**  **Template\_1** | **Equation\_**  **Template\_2** | **Equation\_1** | **Equation\_2** |
| රෙදි සෝදන යන්ත්රයක් සහ වියළන යන්ත්රයක් සඳහා ඩොලර් 600 ක් වැය වේ. රෙදි සෝදන යන්ත්රය වියළන යන්ත්රයේ මිල මෙන් 3 ගුණයක් වේ. වියළන යන්ත්රයේ පිරිවැය කුමක්ද? | **Type 01** | u1+u2=n1 | n2\*u2=u1 | x+y=600 | 3\*y=x |
| පියුමි සතුව තැපැල්පත් 30 ක් ඇත. ඇය සතුව කුඩා තැපැල්පත් මෙන් 4 ගුණයක් විශාල තැපැල්පත් තිබේ. ඇය සතුව කුඩා තැපැල්පත් ගණන හා විශාල තැපැල්පත් ගණන සොයන්න | **Type 01** | u1+u2=n1 | n2\*u2=u1 | x +y=30 | 4\*y=x |
|  | | | | | |
| පැන්සලක් සහ මකනයක් සඳහා රුපියල් 15 ක් වැය වන අතර , පැන්සල් 2 ක් සහ මකන 3 ක් සඳහා රුපියල් 35 ක් වැය වේ. මකනයක සහ පැන්සලක මිල ගණනය කරන්න | **Type 02** | (n1\*u1) +(n2\*u2)=n3 | u1+u2=n4 | (2\*x) +(3\*y) =35 | x+y=15 |
| කසුනි සතුව ටොෆි සහ චොකලට් වල එකතුව 50 ක් වේ. ටොෆියක මිල රුපියල් 5 ක් වන සහ චොකලට් එකක මිල රුපියල් 3 ක් නම් එ සඳහා ඇයට රුපියල් 120 ක් වැය වුණි.විකිණී ඇති ටොෆි සහ චොකලට් ගණන වෙන වෙනම සොයන්න. | **Type 02** | (n1\*u1) +(n2\*u2) =n3 | u1+u2=n4 | (5\*x) +(3\*y) =120 | x+y=50 |
|  | | | | | |
| ශ්‍රීමා මහත්මියගේ පන්තියේ පිරිමි ළමයින් හා ගැහැණු ළමයින් 24 ක් සිටියි. එහි පිරිමි ළමයින්ට වඩා ගැහැණු ළමයින් 6 ක් ඇත. පිරිමි ළමයින් හා ගැහැණු ළමයින් කී දෙනෙක් සිටීද? | **Type 03** | u1+u2=n1 | n2+u1=u2 | x+y=24 | 6+y=x |
| පෙට්ටියක රතු බෝල හා සුදු බෝල 27 ක් ඇත. එහි රතු බෝල වලට වඩා සුදු බෝල 9 ක් ඇත. පෙට්ටියේ ඇති රතු බෝල සහ සුදු බෝල ගණන සොයන්න | **Type 03** | u1+u2=n1 | n2+u1=u2 | x+y=27 | 9+y=x |
|  | | | | | |
| ගොවියෙක් අශ්වයන් මෙන් 4 ගුණයකට වඩා 8 ක් වැඩියෙන් එළදෙනුන් ඇති දැඩි කරයි. ඔහුට අශ්වයින් හා ගවයින් 168 ක් සිටී නම් ඔහුට සිටින ගවයින් හා අශ්වයින් කීයක් සිටීද? | **Type 04** | u1+u2=n1 | n2+(n3\*u1) =u2 | x+y=168 | (4\*x) +8=y |
| ධාන්‍ය ගබඩාවක කඩල ගෝනි සහ කවුපි ගෝනි 21 ක් ඇත. එහි කඩල ගෝනි මෙන් 2 ගුණයකට වඩා 3 ක් කවුපි ගෝනි ඇත. ගබඩාවෙහි ඇති කඩල ගෝනි සහ කවුපි ගෝනි ගණන සොයන්න | **Type 04** | u1+u2=n1 | n2+(n3\*u1) =u2 | x+y=21 | (2\*x) +3=y |
|  | | | | | |
| නිමාලි සතුව ඇති නිල් පෑන් සහ රතු පෑන් ගණන අතර වෙනස 8 කි. ඇය සතුව ඇති මුලු පෑන් ගණන 20 ක් නම්, නිල් පෑන් සහ රතු පෑන් ගණන වෙන වෙනම සොයන්න | **Type 05** | u1+u2=n1 | u2-n2=u1 | x+y=20 | x-y=8 |
| සපත්තු සාප්පුවක දුඹුරු සපත්තු සහ කළු සපත්තු ගණන අතර වෙනස 26 කි. මුළු සපත්තු සංඛ්‍යාව 66 කි. දුඹුරු සපත්තු හා කළු සපත්තු වෙන වෙනම කීයක් තිබේද? | **Type 05** | u1+u2=n1 | u2-n2=u1 | x+y=66 | x-y=26 |

*Table 5: Examples for types in Complex questions module*

A dataset will feed into ANN in order to train that classification ANN to predict type of problem correctly and accurately.

A screenshot of a cell phone

Description automatically generated

Figure - Summary of Classification ANN of Complex problems

After processing this fetched new problem in this ANN classification it will provide Template type which is the structure of the of the question that it should follows in the templates as shown in figure ().

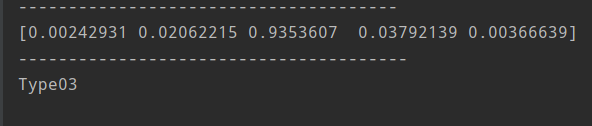
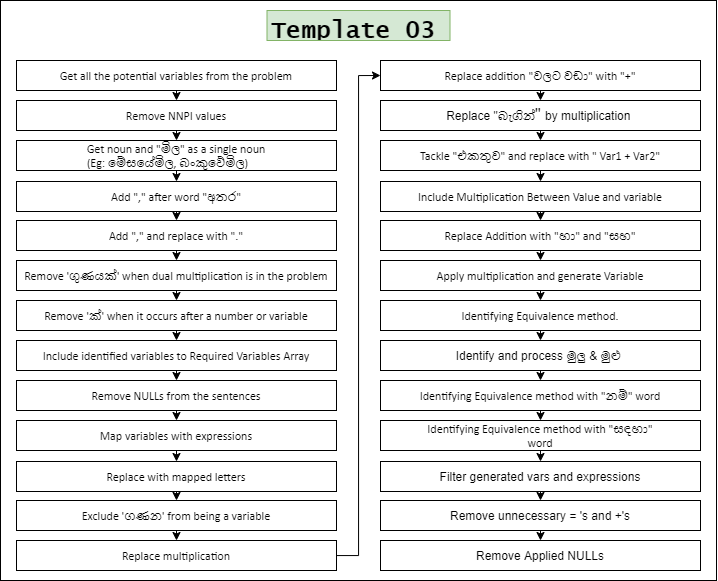


Figure - Output of Classification module of Complex problems

#### 5.5.2.2 Template Based Solving Module

Equation set is generated using input template-based information which is extracted from the problem text. Template based information contains entities, variables, numerical values, and operator information in a semantic representation which is understandable to machines. NLP module will output the required template-based information to map to the equation. This process which involves rules of different kinds based mostly on POS tags of the words and key words of the common algebraic linguistics used in Sinhala language. But output will sometimes contain some noises due because extracting universally applied and true rules for Sinhala language is a difficult process since Sinhala language is very ambiguous language.



After the rule execution of the template the input sentence will be generated into equations. But equations are contained noise also. It occurs because when generating universally valid

there can be unnecessarily generated expressions and that is the noise for this module’s output.

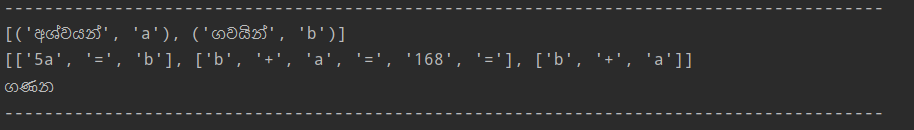


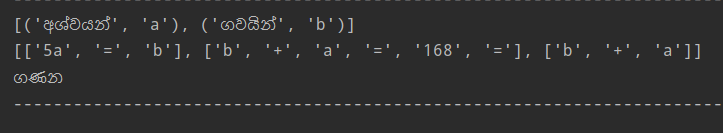
Figure - Output after execution of rules in the template



Figure - Noise in the result.

#### 5.5.2.3 Smoothing Module

Complex equation solving module gives an output as following .

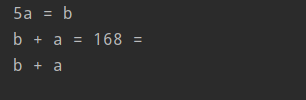


Above output contains noises which are not relevant to the answer. This can be happened due to the structure of the questions. Sinhala language is ambiguous in many situations. Therefore, when applying universal rules to a particular question, there is a high probability of interpreting those rules in different ways. Therefore, output can contain unnecessary parts that shouldn’t be included in the answer.

Ex:- ගොවියෙක් අශ්වයන් මෙන් 5 ගුණයක් ගවයින් ඇති දැඩි කරයි. ඔහුට ගවයින් හා අශ්වයන් 168 ක් සිටී නම් , ඔහුට සිටින ගවයින් හා අශ්වයින් කීයක් සිටීද?

In above question highlighted part has been taken to the output. Because when processing the question “ගවයින් හා අශ්වයින්” is taken as an Addition as those nouns are combined with “හා” .

But for the solving process input should only contain the two equations. Therefore, smoothing process should be performed to remove noises and clean the output. When studying the output results of questions, it is shown that each type of question has unique set of noises. Therefore, each question type needs to have set of rules to remove noises.

First resulted array is split to extract each array separately.

*Figure 33 Output after splitting*

Then the relevant equations should be extracted from the output. After smoothing the output each od expressions will be checked for unnecessary chars and remove them from the given expressions.



*Figure 33 Final Output of smoothing module*

#### 5.4.5.4 Simultaneous Equations Solving Model

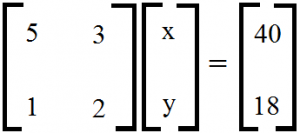
After getting these equations from previous module it should be mapped into a matrix set before solving these equations.  For our example we will be using the following two equations:

5x + 3y =40

x + 2y = 18

These equations should be Convert into the system of equations to matrix form:

For our example we have:

[](http://dwightreid.com/blog/wp-content/uploads/2015/09/simulEqn.png)

Then define coefficient and results matrices as NumPy arrays in order to solve the problem.

*A = np.array([[5,3],[1,2]])*

*B = np.array([40,18])*

Then use NumPy’s linear algebra solve function to solve the system.

*C = np.linalg.solve(A,B)*

## 5.6 Summary

In this chapter, the analysis and design of the system is discussed. In the next chapter, the implementation of the system will be discussed.

# Chapter 6

# Implementation

## 6.1 Introduction

This chapter illustrates the current experiments carried out in implementing the solution.

## 6.2 Handwritten Optical Character Recognition Module

### 6.2.1 Preprocessing

First the image is resized, and gray level transformed. Then Gaussian Blurring is applied to smoothen the image. Black Hat transformation is then applied to enhance the foreground against the background. Finally, thresholding is applied to separate the relevant foreground text. All these functions are applied using OpenCV.

### 6.2.2 Line Segmentation

The preprocessed image is applied with the closing morphological operator to smear the text line to individual blobs. Then contour areas are checked to extract blobs with area bigger than a certain threshold. This threshold was determined to be 100.

### 6.2.3 Initial Segmentation

The vertical projections of individual lines are calculated. Then the characters are segmented using the minimums of the projection.

#### 6.2.4 Implicit segmentation

##### 6.2.4.1 Generation of the dataset

Inputs – Merged dataset (EMNIST + CROHME)

Outputs – Generated dataset

1. *mergedDatasetX 🡨 image array of the merged dataset*
2. *mergedDatasety 🡨 label array of the merged dataset*
3. *generatedDatasetX 🡨 image array of the generated dataset*
4. *generatedDatasety🡨 label array of the generated dataset*
5. *targetSize = 3 000 000 🡨 target size of the generated dataset*
6. *for each i < targetSize*
7. *if i % 72 == 0: 🡨 Add a single character if i%72 = 0*
8. *idx = randomIndexes(mergedDatasetX.length , 1) 🡨 get one random value between 0 and merged dataset size*
9. *image3 = mergedDatasetX[idx]*
10. *newImage = emptyImage(28,56) 🡨 create an empty image of given height, width*
11. *newImage[0:image3.height, 0:image3.length] = image3🡨 Place the image in the new blank image*
12. *generatedDatasetX.add(new\_img)*
13. *label = [mergedDatasety[idx], 72]*
14. *generatedDatasety.add(label)*
15. *else: 🡨 Add two touching/overlapping charactesr if i%72 != 0*
16. *idx = randomIndexes(mergedDatasetX.length , 2) 🡨 get two random values between 0 and merged dataset size*
17. *image1 = mergedDatasetX[idx[0]]*
18. *image2 = mergedDatasetX[idx[1]]*
19. *bottomX1 = getMaxXValue(image1) 🡨 get X value of rightmost pixel of image1*
20. *topX2 = getMinXValue(image2) 🡨 get X value of leftmost pixel of image2*
21. *bottomX2 = getMaxXValue(image2) 🡨 get X value of rightmost pixel of image2*
22. *r = random(0, 3) 🡨 get a random value between 0 and 3*
23. *newImage = emptyImage(28,56)*
24. *newImage[0:image1.height, 0:image1.length] = image1🡨 Place the first image in the new blank image*
25. *new\_img[0:image2.height, bottomX1-r:(bottomX1-r)+(bottomX2- topX2)+1] += image2[0:image2.height, topX2:bottomX2+1] 🡨 Place the second image at the end of first image shifted by r number of pixels*
26. *generatedDatasetX.add(new\_img)*
27. *label = [mergedDatasety[idx[0]], mergedDatasety[idx[1]]]*
28. *generatedDatasety.add(label)*
29. *endif*
30. *endfor*

##### 6.2.4.2 Training the model

The problem was addressed as a multi class classification because the training of the model showed better accuracies than multi label classification in the initial stages.

#### 6.2.5 Explicit segmentation

##### 6.2.5.1 Split characters using contours

The following algorithm is used to split touching/overlapping characters using contour hierarchies.

Inputs – Two touching/overlapping characters, background skeleton of the image

Outputs – Weather the image is segmentable, array of segmented images

1. *boolean,ImageArray splitCharactersUsingContours(image, skeleton) {*
2. *image 🡨 two touching/overlapping characters*
3. *skeleton 🡨 background skeleton of the image*
4. *contours 🡨 stores all the contours found*
5. *hierarchy 🡨 stores the hierarchy information*
6. *new\_images = [ ] 🡨 stores segmented images*
7. *parent0contours = [ ] 🡨 stores the contours whose parent is 0*
8. *contours, hierarchy = findContours(skeleton, TREE) 🡨 retrieve the contours and hierarchies of the skeleton with TREE mode*
9. *for i in range(length(contours)):*
10. *if hierarchy[i][3] == 0: 🡨 check if the contour’s parent is 0*
11. *parent0contours.append(i)*
12. *endif*
13. *endfor*
14. *if length(parent0contours) == 2:*
15. *for each i in parent0contours:*
16. mask = emptyImage(image) *🡨 create an empty image with similar dimensions to the passed image*
17. *drawContour(mask, contours, i, 255, -1) 🡨fill the mask with contour i*
18. new\_image = maskImage(mask, image) *🡨obtain the character image using the mask*
19. *new\_images.add(new\_image)*
20. *endfor*
21. *return True, new\_images*
22. *else*
23. *return False, [ ]*
24. *endif*
25. *}*

##### 6.2.5.2 Segment touching characters

Inputs – Images classified as two touching or two overlapping characters

Outputs – Weather the image is segmentable, segmented images

1. *image 🡨two touching/overlapping characters*
2. *inverted\_image = invert(image) 🡨invert image such that white pixels are the background*
3. *skeleton = skeletonize(inverted\_image) 🡨get the background skeleton*
4. *connectingPoints = [ ] 🡨array to keep the two connecting points between the upper contour and the lower contour*
5. *splittable, images = splitCharactersUsingContours(image, skeleton) 🡨segment the image if it’s two overlapping characters*
6. *if splittable*
7. *return True, images*
8. *else*
9. *outer\_skeleton = emptyImage(image) 🡨 create an empty image with similar dimensions to the passed image*
10. *contours = findContours(skeleton, TREE)*
11. *drawContour(outer\_skeleton, contours, 1, 255, 1) 🡨draw the outermost contour of the background skeleton*
12. *height, width = outer\_skeleton.shape*
13. *w = int(width/3)*
14. *middle = outer\_skeleton[0:height, w:2\*w] 🡨divide the skeleton to three sections and get the middle section*
15. *topKernel = [[2,2,2],[-10,10,-10],[-10,-10,-10]]*
16. *bottomKernel = [[-10,-10,-10],[-10,10,-10],[2,2,2]]*
17. *topPoints = getPoints(middle, topKernel, 3059) 🡨get candidate points(x ,y) whose convolution value is larger than 3059*
18. *bottomPoints = getPoints(middle, bottomKernel, 3059) 🡨get candidate points(x, y) whose convolution value is larger than 3059*
19. *if length(topPoints) == 1 and length(bottomPoints) == 1*
20. *connectingPoints.add(topPoints[0], bottomPoints[0])*
21. *else if length(topPoints) == 0 and length(bottomPoints) != 0*
22. *connectingPoints.add([bottomPoints [0][0],0], bottomPoints[0])*
23. *else if length(topPoints) != 0 and length(bottomPoints) == 0*
24. *sorted\_topPoints = sortOnY(topPoints) 🡨sort points such that points whose Y values are largest (closest to the bottom) are at the start of the array*
25. *connectingPoints.add(sorted\_topPoints[0], [sorted\_topPoints[0][0], height])*
26. *else if length(topPoints) > 1 and length(bottomPoints) == 1*
27. *distances = [ ] 🡨array to hold the distances between the top points and bottom point*
28. *for each topPoint in topPoints*
29. *y = abs(topPoint[1] - btmPoints[0][1])*
30. *x = abs(topPoint[0] - btmPoints[0][0])*
31. *distances.add(x+y)*
32. *end for*
33. *min\_idx = getIndexOfMinValue(distances) 🡨get the index of the minimum distance value in the array*
34. *connectingPoints.add(topPoints[min\_idx], bottomPoints[0])*
35. *else*
36. *return false, [ ]*
37. *end if*
38. *drawLine(outer\_skeleton, (connectingPoints[0][0] + w, connectingPoints[0][1]), (connectingPoints[1][0] + w, connectingPoints [1][1]), 255, 1) 🡨 draw a one pixel wide white(255) line between the two connecting points in the skeleton*
39. *splittable, images = splitCharactersUsingContours(image, outer\_skeleton) 🡨segment the image along the segmentation line*
40. *if splittable*
41. *return True, images*
42. *else*
43. *return False, [ ]*
44. *end if*
45. *end if*

##### 6.2.5.3 Character classifier

The problem is addressed as a multi class classification of 72 labels and the model is trained on the merged dataset of EMNIST + offline CROHME.

## 6.3 NLP Preprocessing Module

### 6.3.1 Raw data Extraction

First of all, the question needs to be extracted from the source as a string. If the source is an “.txt” (Text) file type simple file open function can be used to extract data from file.

But if the source is a “.pdf ” (PDF) file it follows the given process.

1. Inputs - File path that problem is excisted that need to solve
2. Output - Array of content as Strings
3. import PyPDF2 <------ Import PDF reading package
4. # Create File Object
5. filePath <------- File path location of file needs to read
6. pdfFileObj <------- File object
7. details<--------- Content of a single page of PDF
8. Output <------ Array of contents orders according to pages of PDF
9. pdfReader = PyPDF2.PdfFileReader(pdfFileObj) <-------- content of the object Get
10. for page in pdfReader :
    1. details = pageObj.extractText(page)<------- Extract text from page
    2. output.append(details)
11. end for
12. return output <-------- Return array of content

import PyPDF2  
  
def readFile(filePath):  
 path = str(filePath)  
 outPut = []  
 try:  
 if (".pdf" in path):  
 pdfFileObj = open(filePath, 'rb')  
 # Create Reader  
 pdfReader = PyPDF2.PdfFileReader(pdfFileObj)  
 numofPages = pdfReader.numPages  
 print(numofPages)  
  
 # Create Page Object  
 for i in range(numofPages):  
 pageObj = pdfReader.getPage(i)  
 print(pageObj)  
 # Extract text from the Page  
 details = pageObj.extractText()  
 print(details)  
 outPut.append(details)  
  
 elif (".txt" in path):  
 with open(path, "r", encoding="utf-8") as sentences\_file:  
 # delete\_punctuation(sentences\_file)  
 for line in sentences\_file:  
 outPut.append(line)  
  
 except:  
 print("File Type is Not supported")  
 print(outPut)  
 return outPut

### 6.3.2 Tokenizer

The extracted data as string objects should split into sentences and then each sentence should split into token of words.

def tokenize\_texts(text):  
 tokens = nltk.word\_tokenize(text)  
 return tokens  
  
def tokenize\_sents(text):  
 tokens = nltk.sent\_tokenize(text)  
 return tokens

Input :

නිමාලි ගෙ උපන් දින සාදයට පැමිණ සිටි ගණන 25 කි. එහි පිරිමින් මෙන් 3 ගුණයකට වඩා 1 ක් වැඩියෙන් ගැහැණුන් පැමිණ සිටියාය. පැමිණ සිටි ගැහැණුන් හා පිරිමින් ගණන වෙන වෙනම සොයන්න

Output of Sentence Tokenizer :

Output : ['නිමාලි ගෙ උපන් දින සාදයට පැමිණ සිටි ගණන 25 කි.', 'එහි පිරිමින් මෙන් 3 ගුණයකට වඩා 1 ක් වැඩියෙන් ගැහැණුන් පැමිණ සිටියාය.', 'පැමිණ සිටි ගැහැණුන් හා පිරිමින් ගණන වෙන වෙනම සොයන්න']

Output of Word Tokenizer :

Output : ['නිමාලි', 'ගෙ', 'උපන්', 'දින', 'සාදයට', 'පැමිණ', 'සිටි', 'ගණන', '25', 'කි', '.']

Output : ["'එහි", 'පිරිමින්', 'මෙන්', '3', 'ගුණයකට', 'වඩා', '1', 'ක්', 'වැඩියෙන්', 'ගැහැණුන්', 'පැමිණ', 'සිටියාය', '.']

Output : ['පැමිණ', 'සිටි', 'ගැහැණුන්', 'හා', 'පිරිමින්', 'ගණන', 'වෙන', 'වෙනම', 'සොයන්න']

### 6.3.3 Stemming

In stemming the system refers a dictionary that consists stems and their root words to perform stemming process. That array of stems compares with the tokens that extracted from the tokenizer when the input sentence is given and follow the designed flow in order to perform stemming process.

Inputs - Tokenized texts which contains list of words

Output - Stemmed String as array

stem\_dict <------ # Get the sinhala stem dictionary from the file location

stem\_array = []<------ # Array contains all the stemming word tuples

tokens <--------Tokenized Texts

for each tokens:

i <------ # Single Tokens

for i in range(0, len(stem\_array)):<-----#Check tokens against stemming words

if (stem\_array[i][0] == tokens[k]):

tokens[k]<-----#One word of the tokenized text

stem\_array[i][1]<-----#Root word for that word

tokens[k] = stem\_array[i][1]<-----#Swap root form of word to the tokens set

end if

end for

end for

### 6.3.4 Filtering Module

Filtering module supposed to perform two tasks. Those are

* Stop words removal
* Unnecessary Punctuations removal

#### 6.3.4.1 Stop words Removal

The list of stop words published by UCSC contained 480 words in Sinhala. But some of those words are essentials in future modules since those words have meanings in mathematical domain context. Therefore, that USCS stop words list were reviewed created a new set of data to make it compatible for our system .

Inputs - Stemmed Sentences with root words

Output - Filtered sentence without stop words

stopWordsPath <------#Path of the StopWords Dictionary file

stopwords <------#Array of the StopWords

StopWordsDic = [] <------#Dictionary of the StopWords

Word <------#Each element in StopWords Array

stemmedText <------#Stemmed Sentences with root words

token<------#Stemmed tokens

for word in stopwords:

for token in stemmedText:

if (word == token):

remove that token from text

else:

Keep that token in the text

End if

end for

end for

#### 6.3.4.2 Unnecessary Punctuations Removal

In the filtering process, the system needs to eliminate unnecessary punctuations.

Unnecessary\_punctuations = ['(', ')', "\*\*\*", ';', ',', '.', '!', '"', "'", "#", "...", "..", "-", "!", "@", "^"]

Inputs – Stop words removed Token array

Output - Filtered sentence without unnecessary punctuations

sentences < ------------- Stop words removed tokens

Unnecessary\_punctuations = [] <------#Array of unnecessary punctuations

item <------#Each element in s Array

stemmedText <------#Stemmed Sentences with root words

token<------#Stemmed tokens

for item in Unnecessary\_punctuations:

for token in stemmedText:

if (word == token):

remove that punctuations from text

else:

Keep that token in the text

End if

end for

end for

### 6.3.5 Part of Speech Tagging

In this system we have used the N-gram perceptron tagger approach and HMM tagger approach to get the most accurate one for the tagging process. But evaluation results showed that N-gram perceptron tagger approach has the higher accuracy than HMM tagger approach. Therefore, system is developed with N-gram perceptron tagger.

\*Both N-gram tagger and HMM tagger Implementation is attached in Appendix

POS Tagged sentence :

['නිමාලි/NNP ගෙ/NNN උපන්/NNN දින/NNN සාදයට/NNN පැමිණ/VP සිටි/VP ගණන/NNN 25/CD කි/VP ./. එහි/DET පිරිමින්/NNN මෙන්/MUL 3/CD ගුණයකට/MUL වඩා/ADD 1/CD ක්/NNN වැඩියෙන්/ADD ගැහැණුන්/NNN පැමිණ/VP සිටියාය/NNN ./. පැමිණ/VP සිටි/VP ගැහැණුන්/NNN හා/ADD පිරිමින්/NNN ගණන/NNN වෙන/DT වෙනම/DT සොයන්/VP']

### 6.3.6 Fetch Data for Models

This model is implemented to perform 3 types of processes in it. Those are

* Feature Extraction
* Noise words Removal
* Extract Numerical Values

Inputs - words, numcounter, nums, tagged, forANN

Output – set of data ready to fetch

words <------------Tokenized tokens from Sentence

numcounter <------- Numbers Counter initialized value

nums < ------ Numerical value array initialization

tagged <------- Preprocessed and tagged tokens after stemming and filtering

forANN <------ Boolean to check whther fetching for train or not

def prepare\_single\_data <----------- Fetch data to models

length <------------- Length of the words array

for i in range(0,k):

if(Check for words such as "මහතා"):

Remove those words

end if

if ( given as a fraction "3/2"):

Replace it as fraction to the token list

end if

if (tagged set has Numerical tags):

Add that number to nums array

end if

if (Words equals to these types of sinhala vals තෙගුණයකට,පස්ගුණයක්):

replace it with corresponding numerical val

end if

end for

return feature extracted and fetched data

def build\_phrases(): <-------- Build phrses for problems in dataset

words<--------- dictionary of Key words

phrasesinsentences <----------- phrases in the sentences

problem <------ single problem in datset

strings <------ List of problem in dataset

nWords <------ Starting pointer

phraseList <------- List of phrases

for problem in strings:

for j in range (len(problem) - nWords + 1):

phraseList = [problem[j]]

for k in range(1,nWords):

phraseList.append(problem[j + k])

phrase<--------- detokend items

Detokening process <---------- Use Moses Derokenizer to detoken

end for

if(phrase not in phrasesinsamesentence):

Add phrase to phraseList

if (phrase not in words):

check

else:

continue

end if

end for

end for

sorting phrses process <------ Sorting of extracted

return phraselist

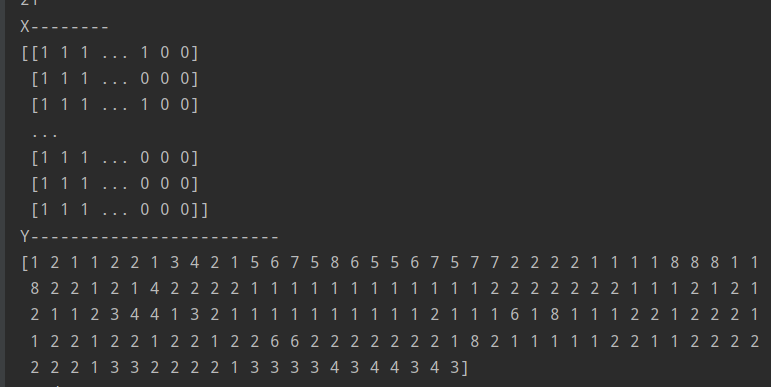
## 6.4 Equation Generation Module

### 6.4.1 Simple Problem Module

Questions with single variable is processed by this module. This module will get the preprocessed question as input. Then the classification module will process the input data and output the type of the simple question

#### 6.4.1.1 Classification Module

Simple problems can have 4 types of operations Addition, Subtraction, Division and Multiplication. Also has the potential of solving questions with any combination of 3 operations. Preprocessed data is fetched to the classification model and then type of the question is given as order of operations that needed to be proceed. Classification is performed using ANN. Preprocessed questions will be encoded converted into an array. As shown in below figure X, Y will be the input of the model.



#### 6.4.1.2 Simple Problem Solve Module

simple word solving module get the array of numerical values. It will process data and out put the answer.

1. Input - Numerical value array created from feature extraction module,

Type\_identifier of the problem type given from Classification Module

1. Output – Result of Equation
2. problemType <----------#Type\_identifier of type of problem
3. numbers <----------#numerical value array
4. ops <--------- list of opertaions extracted from Type\_identifier
5. temp1 <-------Result after first operation execution
6. temp2 <-------Result after second operation execution
7. temp3 <-------Result after third operation execution
8. operation <-------- Items in ops array
9. for operation in ops: <----------- Three iterations
10. if (operation == "ADD"):

Perform Addition

1. else if (operation == "SUB"):

Perform Subtraction

1. else if (operation == "MUL"):

Perform Multiplication

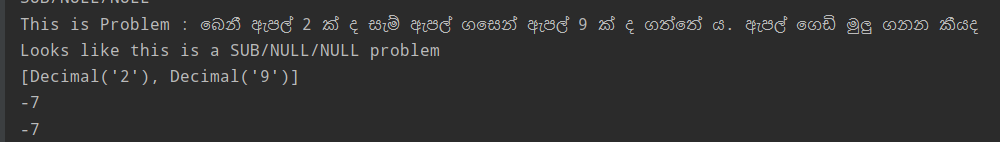
1. else if (operation == "DIV"):

Perform Division

1. else if (operation == "NULL"):

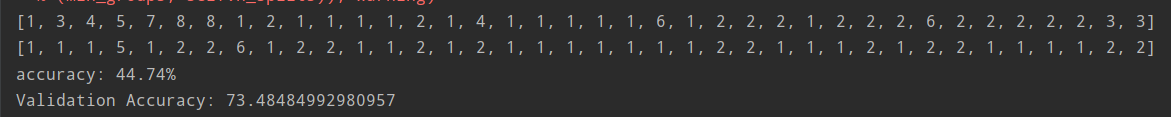
No action

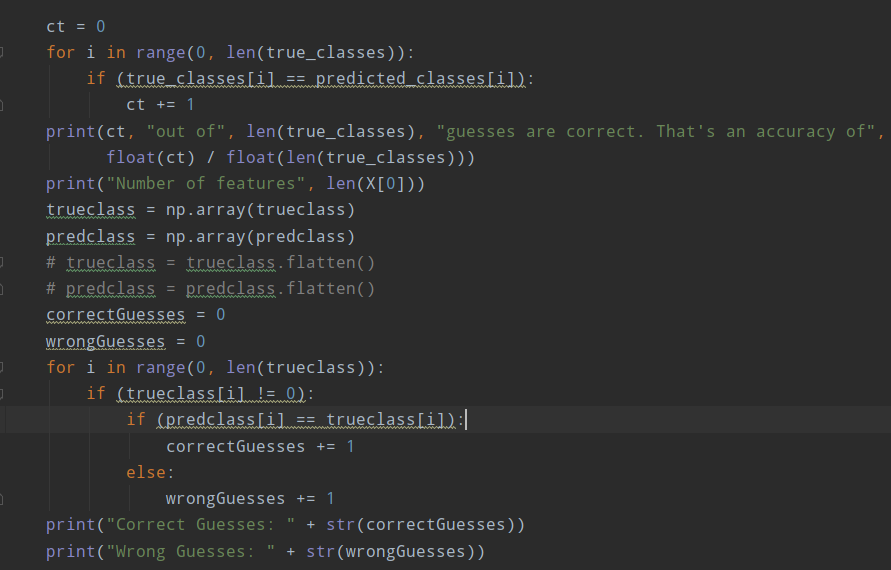
1. end if
2. end for
3. result <------ Last result from equations
4. return result



#### 6.4.1.3 Simple Problem Classification Evaluation

Simple problem classification evaluation has performed using K-fold cross validation. K value of this model is 5. K-fold techniques is used due to the small dataset and its model skill of resulting in an unbiased output. Dataset is shuffled randomly and split into K groups. One group is set as a holdout group or test set and the remaining groups will be set as training set. Every group will be given an opportunity to be used as a test set once and to be the training set K-1 times. It will output the validation accuracy, true class and the predicted class of dataset in each iteration.





*Figure 41 Model output is summarized get total evaluation*

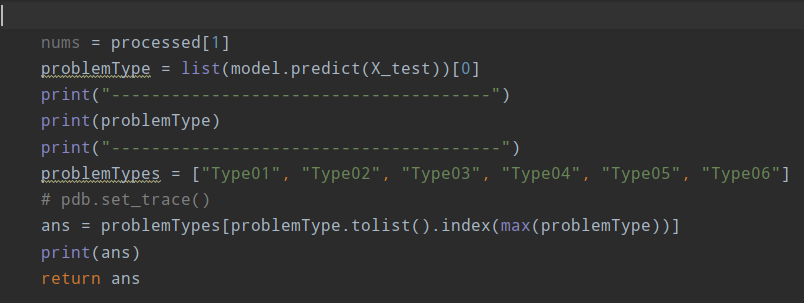
1. correctclass = 0
2. for i in range((0 , len(trueclasses))
3. if ( trueclass [i]== predictedclass[i])
4. correctclass = correctclass + 1
5. trueClass < -------------- np.array of tureclass
6. predclass < --------------- np.array of predclass
7. correctguesses < ------------- number of correct guesses
8. wrongguesses < -------------- number of wrong guesses
9. for i in range (0, len(trueclass))
10. if (trueclass[i] != 0)
11. If (predclass[i] == trueclass[i]
12. correctguesses += 1
13. else
14. wrongguesses +=1

### 6.4.2 Complex Problem Module

Questions with two variables are processed using this module. It’s more complex than the simple problem module as it needs to be generated simultaneous equations. This module contains a classification module to classify the type of the equation and the value mapping module to map values from the questions to equation template. This module has followed a rule-based approach.

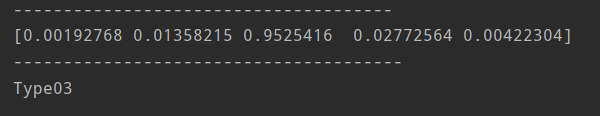
#### 6.4.2.1 Classification Model

Classification model of complex problem module is supervised, and semi supervised learning model with set of algebraic questions and answers. Keywords, numerical values that were extracted from the questions in previous module are taken as the inputs to the classification model. Complex problem module is capable of solving 5 different types of questions. Classification model is used to classify the type of the equation by processing the question.



*Figure 41 Getting the type of the question*

The type that get high probability for particular question will be assign as the type of the question .



*Figure 41 Generated output of the complex type classifier*

As shown in the above figure there is 0.95 probability that the processed question to be a type 03 question.

#### 6.4.2.1 Value Mapping Model.

This module is used to map values that were extracted from questions to equation templates. This follows a rule-based process. Each type of questions consists of set of rules to map to the relevant equation template. There are common rules for all 5 templates and set of unique rules for each template. But due to the ambiguity of Sinhala language, applying those rules won’t give the correct answer for all the questions.

\* Template rules for the type 1 is attached in the appendix

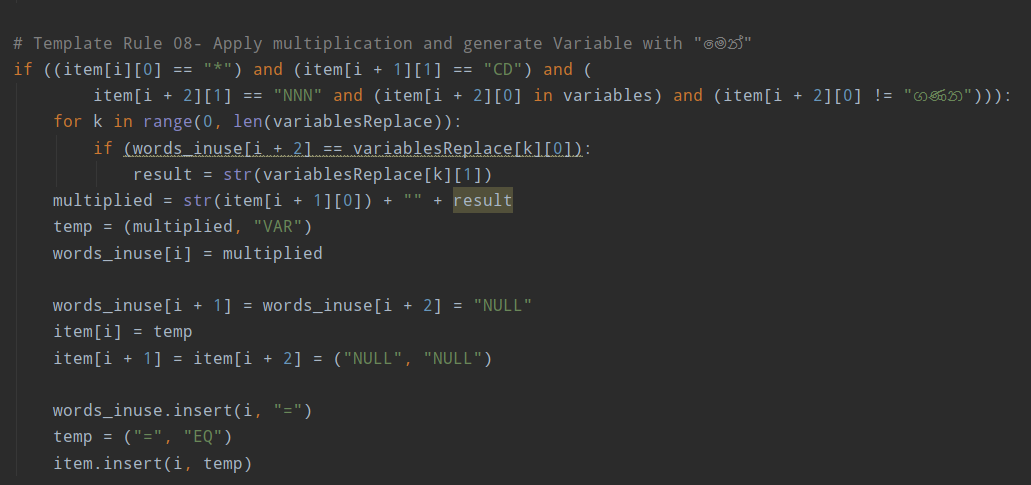


Figure - Example for Rule in a template (Template 01 – rule 08)

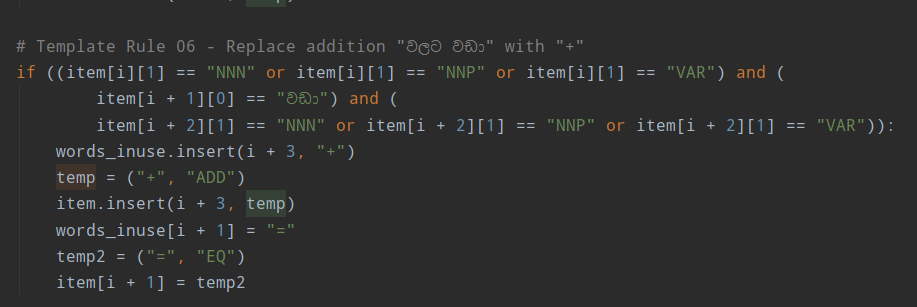


Figure - Example for Rule in a template (Template 03 – rule 06)

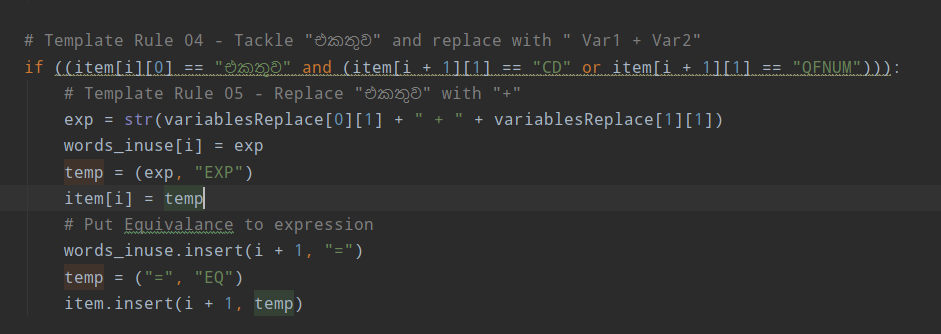
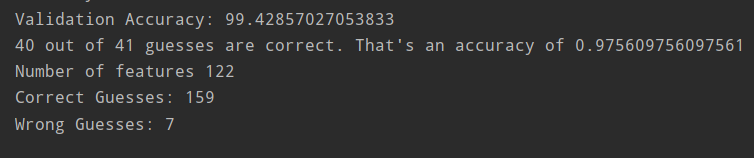


Figure - Example for Rule in a template (Template 04 – rule 04)

#### 6.4.2.1 Simultaneous Equation Solving.

#### 6.4.2.1 Complex Problem Classification Evaluation

Classification model of the Complex problem module is also evaluated using K-fold validation.



#### 6.4.2.1 Dataset preparation complex problem Module

Preparation of dataset for the complex problem module was a complicated task. Dataset had to be prepared manually in a manner that it can be processed by the model to classify the type. Also, in value mapping module rules should be implemented according to the type of questions in the dataset. Although there are only five main types , each type can have different questions that indicate the same type in whole different way. Certainly, there are some limitations when applying rules for the questions specially for Sinhala language. Therefore, questions were thoroughly studied before insert into the dataset. Evaluation of the rules of value mapping module was performed using the dataset.

Link of the dataset :

<https://drive.google.com/file/d/1SDGcVm4ljdDxT6nayDo8uQotrcl1zxI2/view?usp=sharing>

Chapter 7

# Evaluation

## 7.1 Introduction

This chapter will describe the experiments that were conducted in each module. The results obtained in each experiment are explained with the level of accuracy of the module.

## 7.2 Evaluation of the Handwritten OCR Module

The Handwritten OCR module is evaluated sub module wise and all together as a whole system.

### 7.2.1 Module wise evaluation

The following are the evaluations of each module.

1. Single character classifier

This is a sub module in the explicit segmentation module. Segmented single characters are classified using this classifier. The test set has 140 862 (EMNIST+ offline CROHME) images belonging to 72 classes.

|  |  |
| --- | --- |
| Model | Test Accuracy |
| 4-layer CNN | 87.73% |
| 6-layer CNN | 88.11% |
| ResNet-50 | 88.01% |

Figure 27 Single character classifier accuracy

1. Recognizing characters to be segmented (binary classification)

This is a sub module in the explicit segmentation module. This classifier is used to recognize weather an image is of a single character or multiple (2) touching/overlapping characters. This module is tested on the dataset that is generated. There are 31 868 test images in the dataset.

Hierarchical clustering failed to correctly cluster the data in this dataset. Many irrelevant points were being recognized as clusters. Therefore, K-Means clustering with K=2 was used to cluster the data.

|  |  |
| --- | --- |
|  | Test Accuracy |
| 8-layer CNN | 98.57% |
| K-Means clustering (K=2) | 92.07% |

Figure 28 Single or multiple character classification (binary)

1. Single or touching/overlapping character classification

Here the images of single characters or images of two touching/overlapping characters are segmented and classified. The segmentation and classification of characters is achieved simultaneously in the implicit segmentation module.

For explicit segmentation, 8-layer CNN was used to recognize touching/overlapping characters followed by background thinning based segmentation. Finally, the segmented characters are recognized using the single character 6-layer CNN.

This evaluation is performed using the generated dataset mentioned in 5.3.1. A. Out of 3 230 980 images in the dataset 323 098 images are of the test set.

|  |  |  |
| --- | --- | --- |
|  | | Test Accuracy |
| Implicit segmentation | 4-layer CNN | 63.85% |
| 6-layer CNN | 63.77% |
| ResNet-50 | 96.01% |
| Explicit segmentation | | 31.61% |

Figure 29 Single or multiple character segmentation and classification accuracy

### 7.2.1 Evaluation on real life data

The OCR as a whole was tested on 213 handwritten characters. These characters were sourced from 3 documents containing characters, numbers and math symbols. There are 151 single characters and 30 touching/overlapping character pairs in the test set.

For testing implicit segmentation ResNet-50 model was used.

For explicit segmentation hierarchical clustering was used to recognize touching/overlapping characters from single characters. Although the 8-layer CNN was more accurate on the generated dataset, hierarchical clustering was more accurate on real life data. After being recognized as touching/overlapping characters, they are segmented by the background thinning based segmentation. Finally, the segmented characters are recognized using the single character 6-layer CNN.

|  |  |
| --- | --- |
|  | Accuracy |
| Implicit Segmentation (ResNet-50) | 53.99% |
| Explicit segmentation (Hierarchical clustering+backgroundthinning segmentation + 6-layer CNN) | 60.09% |

## 7.3 Evaluation of Preprocessing Module

Evaluation of preprocessing module has performed by taking the POS tagging accuracy. POS tagging has implemented in two approaches. They are perception tagger method and the Hidden Marcov method. HMM method achieve only 72 % accuracy. Therefore, perception tagger method approach which achieves 99% accuracy is proven better. Perception tagger has implemented in Unigram, bigram and trigram models. Evaluation of the unigram POS tagger has shown below.

* HMM Tagger Accuracy – 0.716787834502303
* Unigram Perceptron Tagger Accuracy – 0.9833871712044301
* Trigram Perceptron Tagger Accuracy - 0.7843562528841717

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tag | precision | recall | f1-score | support |
| ‘ | 0.93 | 1.00 | 0.96 | 38 |
| . | 0.98 | 1.00 | 0.99 | 123 |
| = | 1.00 | 1.00 | 1.00 | 45 |
| ADD | 0.94 | 1.00 | 0.97 | 48 |
| ATT | 1.00 | 1.00 | 1.00 | 84 |
| CD | 0.97 | 1.00 | 0.99 | 77 |
| CD, | 0.00 | 0.00 | 0.00 | 1.00 |
| DET | 1.00 | 0.98 | 0.99 | 107 |
| DIV | 1.00 | 1.00 | 1.00 | 2 |
| DT | 0.00 | 0.00 | 0.00 | 0.00 |
| EXP | 1.00 | .98 | 0.99 | 53 |
| G | 1.00 | 1.00 | 1.00 | 12 |
| JJ | 1.00 | 0.99 | 1.00 | 140 |
| KG | 1 | 1.00 | 1.00 | 17 |
| KM | 1.00 | 1.00 | 1.00 | 2 |
| LTR | 1.00 | 1.00 | 1.00 | 3 |
| METER | 1.00 | 1.00 | 1.00 | 6 |
| MUL | 1.00 | 1.00 | 1.00 | 35 |
| NNF | 1.00 | 0.80 | 0.89 | 5 |
| NNM | 1.00 | 1.00 | 1.00 | 15 |
| NNN | 0.99 | 1.00 | 0.99 | 564 |
| NNP | 1.00 | 1.00 | 1.00 | 5 |
| NNPA | 1.00 | 1.00 | 1.00 | 10 |
| NNPI | 0.99 | 1.00 | 0.99 | 66 |
| NVB | 1.00 | 0.98 | 0.99 | 44 |
| NVM | 1.00 | 1.00 | 1.00 | 3 |
| NONE | 0 | 0 | 0 | 4 |
| OP | 1.00 | 1.00 | 1.00 | 9 |
| POST | 0.99 | 0.98 | 0.98 | 84 |
| PRP | 1.00 | 1.00 | 1.00 | 22 |
| QFNUM | 1.00 | 1.00 | 1.00 | 72 |
| QNUM | 1.00 | 1.00 | 1.00 | 49 |
| RP | 1.00 | 1.00 | 1.00 | 28 |
| SUB | 1.00 | 1.00 | 1.00 | 6 |
| UH | 1.00 | 1.00 | 1.00 | 3 |
| VAR | 1.00 | 0.95 | 0.97 | 76 |
| VNM | 1.00 | 1.00 | 1.00 | 8 |
| VP | 0.95 | 0.99 | 0.97 | 237 |
| Accuracy | 0.99 |  |  | 2165 |
| Macro avg | 0.90 | 0.87 | 0.88 | 2165 |
| Weighted avg | 0.98 | 0.99 | 0.98 | 2165 |

Therefore, as the POS tagging module, we have selected the N-gram perceptron tagger approach.

The Stemming approach was evaluated using 200 words extracted from grade 08 math book. The characteristics of the dataset were that they consist of many Sinhala words which cannot be found in normal news corpuses or any corpus that can found on USCS web site .

For the evaluation, the preprocessed dataset was tagged using the trained perceptron tagger. The proposed algorithm was performed on test words and all the existing words on the stem dictionary was given correct root words. From the experimental dataset, 49 words produced incorrect results because they were not existed in the stem dictionary. Accuracy of the approach was calculated as 80.5% using the following equation.

𝐴𝑐𝑐𝑢𝑟𝑎𝑐𝑦= ×100%

Comparison of the results using other existing approaches for Sinhala language are limited since required details and manually created social media domain specific lexicons which is needed for applying those algorithms are not published.

## 7.4 Evaluation of Equation Generation Module

Equation generation module has implemented for simple questions which has single variable and the complex questions which contain two variables. Performance of each model is evaluated separately.

### 7.4.1 Evaluation of Classification models

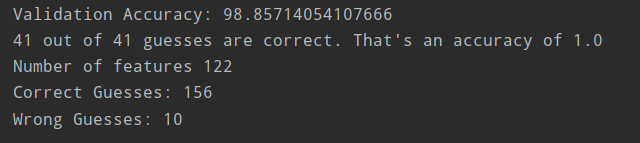
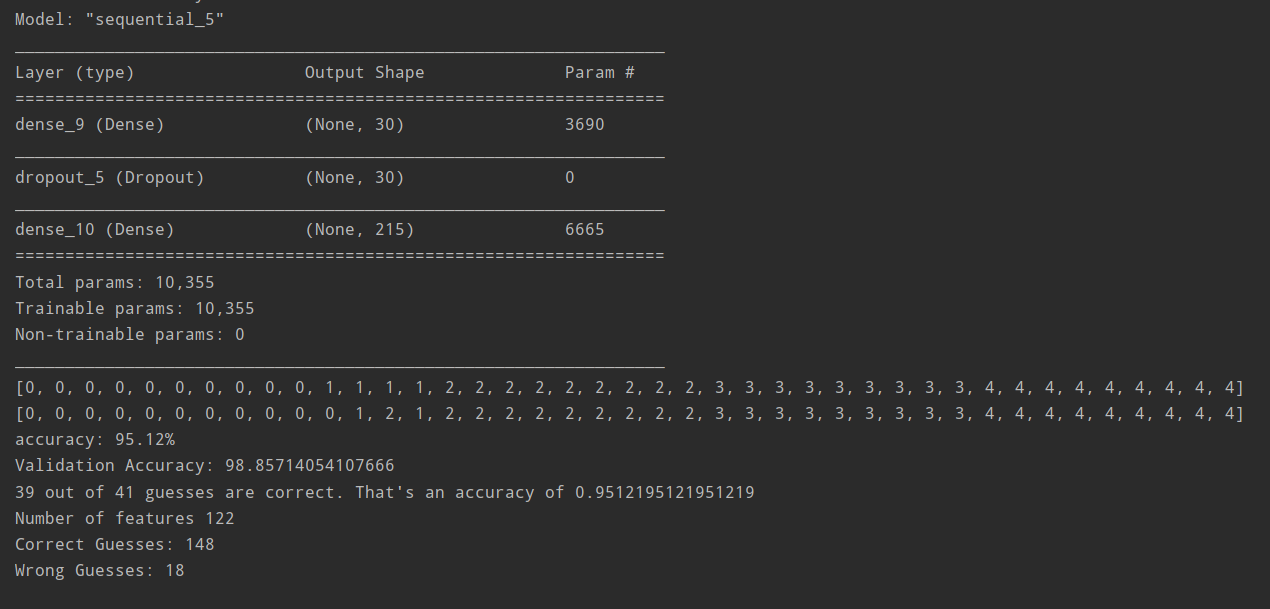
Classification models of both Simple equation model and the Complex equation model has implemented using K-fold cross validation method.

#### 7.4.1.1 K-Fold Cross Validation

In K-Fold Cross Validation the data set is partition into several subsets. Then hold out a set at a time and train the model on the remaining set. Then the model is tested from the holdout set. Purposes of using K-fold cross validation for our classification model are it uses all of its data for training as well as for testing. As we have relatively small data set it is better to use K-fold technique to get an unbiased output. As we have chosen 5 as the K value it trains five models from the dataset. From the evaluation of five models we get an idea about accuracy and the consistency of data. Using this method will help to reduce underfitting and overfitting which are common scenarios that occur in machine learning. K-fold technique shows better performance for small data sets due to the reasons explained above.

#### 7.4.1.2 Evaluation of Classification Model of Complex Equations.

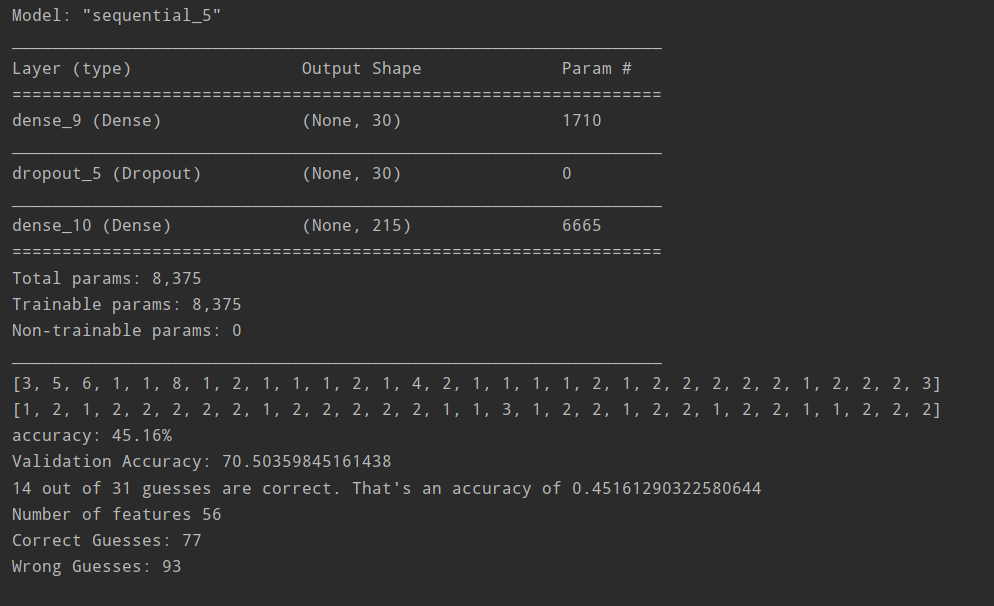
#### Accuracy of the classification model of complex equations achieve nearly 99%.



*Figure 34 Complex Problem Classifier Evaluation*

#### 7.4.1.2 Evaluation of Classification Model of Simple Equations.

The dataset of the Simple problem-solving module contains 64 types of questions. But number of questions in the dataset are not enough to make accurate predictions of the question type. Therefore, model evaluation resulted in less accuracy.



*Figure 35 Simple Problem Classifier Evaluation*

### 7.4.1 Evaluation of Rule-Based module

Rule based approach is used to map the values of the questions to equation template. The dataset of the complex equation module contains equal sets of questions for each type of question. Evaluation of this was performed manually. Each question is processed and validated the result to evaluate the accuracy of rule-based module. Sample of evaluated questions are shown below.

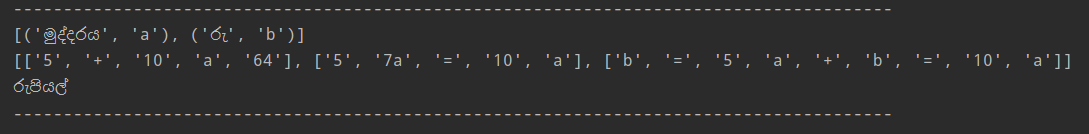
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Question** | **Type** | **Received Solution** | **Equation** | |
|  | **Equations** | **Correct /wrong** |
| රෙදි සෝදන යන්ත්රයක් සහ වියළන යන්ත්රයක් සඳහා ඩොලර් 600 ක් වැය වේ. රෙදි සෝදන යන්ත්රය වියළන යන්ත්රයේ මිල මෙන් 3 ගුණයක් වේ. වියළන යන්ත්රයේ පිරිවැය කුමක්ද? | Type 01 |  | x+y=600  3\*y=x | Correct |
| පියුමි සතුව තැපැල්පත් 30 ක් ඇත. ඇය සතුව කුඩා තැපැල්පත් මෙන් 4 ගුණයක් විශාල තැපැල්පත් තිබේ. ඇය සතුව කුඩා තැපැල්පත් ගණන හා විශාල තැපැල්පත් ගණන සොයන්න | Type 01 |  | x+y=30  4\*y=x | correct |
| පැන්සලක් සහ මකනයක් සඳහා රුපියල් 15 ක් වැය වන අතර , පැන්සල් 2 ක් සහ මකන 3 ක් සඳහා රුපියල් 35 ක් වැය වේ. මකනයක සහ පැන්සලක මිල ගණනය කරන්න | Type 02 |  | (2\*x)+(3\*y)=35  x+y=15 | correct |
| කසුනි සතුව ටොෆි සහ චොකලට් වල එකතුව 50 ක් වේ. ටොෆියක මිල රුපියල් 5 ක් වන සහ චොකලට් එකක මිල රුපියල් 3 ක් නම් එ සඳහා ඇයට රුපියල් 120 ක් වැය වුණි.විකිණී ඇති ටොෆි සහ චොකලට් ගණන වෙන වෙනම සොයන්න. | Type 02 |  | (5\*x)+(3\*y)=120  x+y=50 | correct |
| ශ්‍රීමා මහත්මියගේ පන්තියේ පිරිමි ළමයින් හා ගැහැණු ළමයින් 24 ක් සිටියි. එහි පිරිමි ළමයින්ට වඩා ගැහැණු ළමයින් 6 ක් ඇත. පිරිමි ළමයින් හා ගැහැණු ළමයින් කී දෙනෙක් සිටීද? | Type 03 |  | x+y=24  6+y=x | correct |
| පෙට්ටියක රතු බෝල හා සුදු බෝල 27 ක් ඇත. එහි රතු බෝල වලට වඩා සුදු බෝල 9 ක් ඇත. පෙට්ටියේ ඇති රතු බෝල සහ සුදු බෝල ගණන සොයන්න | Type 03 |  | x+y=27  9+y=x | correct |
| ගොවියෙක් අශ්වයන් මෙන් 4 ගුණයකට වඩා 8 ක් වැඩියෙන් එළදෙනුන් ඇති දැඩි කරයි. ඔහුට අශ්වයින් හා ගවයින් 168 ක් සිටී නම් ඔහුට සිටින ගවයින් හා අශ්වයින් කීයක් සිටීද? | Type 04 |  | x+y=168  (4\*x)+8=y | correct |
| ධාන්‍ය ගබඩාවක කඩල ගෝනි සහ කවුපි ගෝනි 21 ක් ඇත. එහි කඩල ගෝනි මෙන් 2 ගුණයකට වඩා 3 ක් කවුපි ගෝනි ඇත. ගබඩාවෙහි ඇති කඩල ගෝනි සහ කවුපි ගෝනි ගණන සොයන්න | Type 04 |  | x+y=21  (2\*x)+3=y | correct |
| නිමාලි සතුව ඇති නිල් පෑන් සහ රතු පෑන් ගණන අතර වෙනස 8 කි. ඇය සතුව ඇති මුලු පෑන් ගණන 20 ක් නම්, නිල් පෑන් සහ රතු පෑන් ගණන වෙන වෙනම සොයන්න | Type 05 |  | x+y=20  x-y=8 | coreect |
| සපත්තු සාප්පුවක දුඹුරු සපත්තු සහ කළු සපත්තු ගණන අතර වෙනස 26 කි. මුළු සපත්තු සංඛ්‍යාව 66 කි. දුඹුරු සපත්තු හා කළු සපත්තු වෙන වෙනම කීයක් තිබේද? | Type 05 |  | x+y=66  x-y=26 | Correct |

#### 7.4.1.1 Fault Results

When evaluating the questions, some of the questions didn’t give the accurate output of equation. In Sinhala language even the same type questions can be presented in various ways due to the ambiguity of the language words can be interpreted in different meanings. Some of the questions that resulted in fault outputs are explained below.

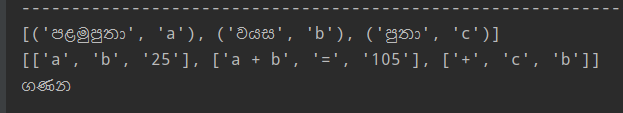
Ex:

1. නිශාන් සතුව ඇති රුපියල් 5 සහ රුපියල් 10 මුද්දර ගණන 64 ක් ඇත. ඔහු සතුව රුපියල් 5 මුද්දර මෙන් 7 ගුණයක් රුපියල් 10 මුද්දර ඇත. රු 5 මුද්දර ගණන හා රු 10 මුද්දර ගණන වෙන වෙනම සොයන්න



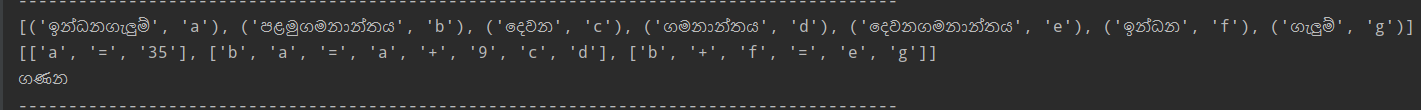
As shown in the above figure question doesn’t give the expected output. The reason for this is question contains words such as “රුපියල් 5 මුද්දර”, “රුපියල් 10 මුද්දර”. Those words were not recoganized as nouns.

2. රීසාගේ පළමු පුතා ඉපදෙන විට රීසාට වයස අවුරුදු 25 කි. අද ඔවුන්ගේ වයස්වල එකතුව 105 කි. රිසාට සහ ඇගේ පුතාට දැන් වයස කීයද



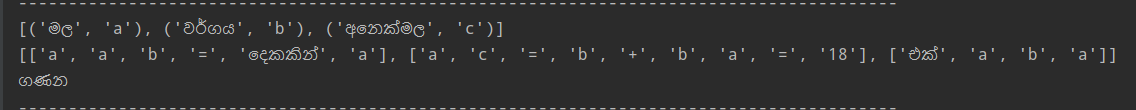
This module also fails to process these kinds of questions which contain extremely implicit information about variables. According to the above example when it says “රීසාගේ පළමු පුතා ඉපදෙන විට රීසාට වයස අවුරුදු 25” , it doesn’t directly indicate that “රීසා” is 25 years older than “පුතා”.

3 ගුවන් යානයක් ඉන්ධන ගැලුම් 35 ක් සහිතව ගමන් ආරම්භ කරයි. පළමු ගමනාන්තය වන විට වැය වු ඉන්ධන ගැලුම් ගණනට වඩා ඉන්ධන ගැලුම් 9 ක් ඉන් පසු දෙවන ගමනාන්තය තෙක් වැය වුනි. පළමු ගමනාන්තය සහ දෙවන ගමනාන්තය සඳහා වැය වු ඉන්ධන ගැලුම් ගණන සොයන්න



In a question like this it doesn’t recognize the correct variable. This happens when there are too much nouns in a question. Therefore, it processes ඉන්ධන, පළමු ගමනාන්තය and දෙවන ගමනාන්තය all three as variables.

1. මල් කඩය දවස අවසානයේ මල් වර්ග දෙකකින් මල් 56 ක් ඉතිරිවී තිබුණි. ඉන් එක් මල් වර්ගයක මෙන් 5 ගුණයකට වඩා 8 ක් වැඩියෙන් අනෙක් මල් වර්ගයෙන් ඉතිරි වී තිබුනි. එක් එක් මල් වර්ගයෙන් ඉතුරු වී තිබු මල් ගණන වෙන වෙනම සොයන්න



In a question like this it has not used much different words for variables. Variables of this questions are “එක් මල් වර්ගය” and “අනෙක් මල් වර්ගය”. In this question “එක්” word is an adjective for “මල් වර්ගය” . But “එක්” is most of the times used as a determiner. It is tagged in the corpus as a determiner than as an adjective. Therefore, it makes a conflict to the system when choosing variables.

#### 7.4.1.2 Performance of Rule based Module

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type | No of questions processed | Number of  Correct outputs | Number of Incorrect outputs | Error percentage | Correct Percentage |
| 01 | 50 | 46 | 04 | 8% | 92% |
| 02 | 30 |  |  |  |  |
| 03 | 49 | 39 | 10 | 20.14% | 79.59% |
| 04 | 38 | 32 | 6 | 15.789% | 84.21% |
| 05 | 41 | 35 | 6 | 14.634% | 85.36% |

# Chapter 8

# Further Work

## 8.1 Handwritten Recognition Module

## 8.2 Preprocessing Module

### 8.2.1 Limitations of Preprocessing Module

1. Less amount of domain specific keywords.

As this module process algebraic word questions, it certainly needs mathematical related word set to recognize the word forms correctly. But due to the time limitation, there is around 8000 words in the corpus, and it was mannually annotated.

2 UTF-8 encoding doesn’t represent some characters properly.

This module has used UTF-8 encoding. But it doesn’t represent some of the Sinhala characters properly.



1. Module has implemented only in two methods.

This module has followed a lexical based approach and hidden Marcov approach. But there are other methods such as deep learning methods and rule-based methods.

### 8.2.2 Future works of Preprocessing Module

This module can be improved by using large corpus to implement the module. Accuracy of the module can be increased by using large corpus. Because the frequency of a word is increased. Also, module can be implemented in various methods. Then comparing results and proceed with the best method will help to increase the accuracy of the whole system.

## 8.3 Equation Generation Module

### 8.3.1 Limitations of Equation Generation Module

1. Only 5 types of questions can be solved using Complex Problem Module.

Complex problem Module has the potential to solve only 5 types of questions. This module map values to templates using rule-based process. It’s impossible to implement universal rules that can be applied to every question due to the ambiguity of Sinhala Language.

2 . Fails to process implicit information in questions.

When the relationship between variables represents in the questions implicitly module fails to recognize variables separately.

Ex : රීසාගේ පළමු පුතා ඉපදෙන විට රීසාට වයස අවුරුදු 25 කි.

3 When the question consists of too much nouns, module fails to recognize the variables.

4 When the question contains too much number other than variables , it fails to recognize variables.

Ex: නිශාන් සතුව ඇති රුපියල් 5 සහ රුපියල් 10 මුද්දර ගණන 64 ක් ඇත. ඔහු සතුව රුපියල් 5 මුද්දර මෙන් 7 ගුණයක් රුපියල් 10 මුද්දර ඇත

### 8.3.2 Future works of Equation Generation Module

#### 8.3.2.1 Future works of Simple Problem Module

Simple Problem module generate equation type using classification model. This can be improved by identifying simple question types and implement rules for simple questions. The accuracy of the question totally depend on the size of the data set as this approach follows supervised learning method. But using rule-based approach , high accuracy can be achieved from a small dataset.

#### 8.3.2.1 Future works of Complex Problem Module

This module contains above limitations that results in fault outputs when the question is processed. Those weaknesses of the module can be improved by implementing more rules. Also, one type can be divided into more types by studying the forms of words often used in a particular question type. This module also can be implemented in a dynamic approach which involves relationship extraction. Initially our approach was to implement it in a dynamic way. But after proceeded to a certain extent, it was realized that more accuracy can be achieved in rule-based way. As future work this module can also be implemented in dynamic approach. We have implemented this module only for 5 types of simultaneous questions. But module can be improved to process many types of questions.

## 8.4 Conclusion

“ Evaluation of Algebraic Word Problems” system is implemented to solve Sinhala algebraic word questions and mark students’ hand written answers. It contains 3 main modules. Handwritten Optical Character Recognition Module, Preprocessing module and Equation Generation Module. Handwritten Optical Character Recognition Module process the students handwritten answer and extract the answer using image processing techniques. When user uploads the question in a PDF the Sinhala question is processed to extract variables and numerical values from the question. Techniques in NLP has used for this module. Equation Generation Module contains two main modules, Simple problem module which contains questions with one variable and Complex solving module which contains questions with two variables. Module will classify the question type and map values to equation template in the complex module. Simple problem module classifies the question type and solve according to the operation order which was given by the classifier. This module has followed a rule-based approach to generate equations. Evaluation of the Handwritten Optical Character Recognition Module has performed using………… Performance of the POS tagger was measured to evaluate the Preprocessing Module. Classification modules of Equation Generation Module was evaluated using K-Fold cross validation method. Value mapping model was evaluated mannually by processing questions. This System has the capability of solving only limited types of questions. As the further word system can be improved to solve may types of questions with more accuracy.

# References

|  |  |
| --- | --- |
| [1] | T. Saba, G. Sulong and A. Rehman, "A Survey on Methods and Strategies on Touched Characters Segmentation," *Characters Segmentation,* vol. 1, June 2010. |
| [2] | A. Choudhary, R. Rishi and S. Ahlawat, "A New Character Segmentation Approach for Off-Line Cursive Handwritten Words," in *International Conference on Information Technology and Quantitative Management*, 2013. |
| [3] | H. Liulei, K. Moydin, A. Dawut and A. Hamdulla, "The Algorithms For Segmentation Of Text-Lines In Handwriting Images," in *International Conference on Smart City and Systems Engineering* , 2018. |
| [4] | R. Manmatha and J. L. Rothfeder, "A Scale Space Approach for Automatically Segmenting Words from Historical Handwritten Documents," *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE,* vol. 27, 2005. |
| [5] | J. C. Wu, J.-W. Hsieh and Y.-S. Chen, "Morphology-based text line extraction," *Machine Vision and Applications,* 2007. |
| [6] | A. KAUR, S. BAGHLA and S. KUMAR, "STUDY OF VARIOUS CHARACTER SEGMENTATION TECHNIQUES STUDY OF VARIOUS CHARACTER SEGMENTATION TECHNIQUES," *International Journal of Advances in Science Engineering and Technology,* vol. 3, no. 3, 2015. |
| [7] | A. Rehman, D. Mohamad and G. Sulong , "Implicit Vs Explicit based Script Segmentation and Recognition: A Performance Comparison on Benchmark Database," *Int. J. Open Problems Compt. Math,* vol. 2, 2009. |
| [8] | G. R. Casey and L. Eric, "A Survey of Methods and Strategies in Character Segmentation," *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE,* vol. 18, 1996. |
| [9] | Z. Lu, Z. Chi, W.-C. Siu and P. Shi, "A background-thinning-based approach for separating and recognizing connected handwritten digit strings," *Pattern Recognition,* vol. 32, pp. 921-933, 1999. |
| [10] | H. Zhan, S. Lyu and Y. Lu, "Handwritten Digit String Recognition using Convolutional Neural Network," in *International Conference on Pattern Recognition*, Beijing, 2018. |
| [11] | C. Meany and M. Arola, "Optical Character Recognition via Deep Learning," Deep Learning, Stanford University, CA, 2018. |
| [12] | S. S. Sundaram and D. Khemani, "Natural Language Processing for Solving SimpleWord Problems". |
| [13] | S. Singh, "Natural Language Processing for Information Extraction," 6 Jul 2018. |
| [14] | N. Kushman, Y. Artzi, L. Zettlemoyer and R. Barzilay, "Learning to Automatically Solve AlgebraWord Problems," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, Baltimore, Maryland, USA,, 2014. |
| [15] | R. K. Kedziorski, H. Hajishirzi, A. Sabharwal, O. Etzioni and S. D. Ang, "Parsing AlgebraicWord Problems into Equations," 2015. |
| [16] | J. M. Hosseini, H. Hajishirzi, O. Etzioni and N. Kushman, "Learning to Solve ArithmeticWord Problems with Verb Categorization," in *EMNLP*, 2014. |
| [17] | D. Huang, S. Shi, J. Yin and C.-Y. Lin, "Learning Fine-Grained Expressions to Solve MathWord Problems," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, Copenhagen, Denmark, September 7–11, 2017.. |
| [18] | S. Shi, Y. Wang, C.-Y. Li, X. Liu and Y. Rui, "Automatically Solving Number Word Problems by Semantic Parsing and Reasoning," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing,*, Lisbon, Portugal, September 2015.. |
| [19] | L. Wang, Y. Wang, D. Cai, D. Zhang and X. Liu, "Translating a MathWord Problem to a Expression Tree," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, 2018. |
| [20] | S. Roy and D. Roth, "Solving General ArithmeticWord Problems," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, Lisbon, Portugal, 2015. |
| [21] | D. Huynh, "Converting CROHME dataset for Online-handwritting recognition to Offline-handwritting recognition.," 14 December 2018. [Online]. Available: https://github.com/vndee/offline-crohme. [Accessed 8 September 2019]. |

# Appendixes

## Appendix A – Individual contribution

**154187F – W.M.S.A. Panditharatne**

Implementation of the handwritten optical character recognition module is assigned to me. I have researched about various methods of tackling the challenges associated with the above module.

In order to implement the understood literature, I had to learn about OpenCV, Image Processing techniques and Python programming language.

Through my research, I was able to implement up to character segmentation in the Handwritten Optical Character Recognition Module.

**154079C – C.M.P.P.D.V Perera**

I was assigned to implement some modules in the equation generation Module. I have implemented the Simple problem module. Simple problem modules consist of classification model , Problem solving model and the classification evaluation model. Classification model is a Ann model train the dataset and the predict the type of a new question as an order of operations. Then in the problem-solving module it will process the question type and extracted operations will applied to the numerical values that were given from the previous module. I implemented the classification evaluation module which measure the performance and the accuracy of the model. Classification model of the complex problem model was implemented by me. It classifies the type of the question. I also implemented the problem-solving module of complex problem module. It involves solving simultaneous equations. I implemented the type classification of complex problem module in dynamic approach. I implemented Classification evaluation model for the complex problem module. I prepare the datasets for both complex problem module and the simple problem module. Both datasets were mannually made in a manner that can be processed by the module. When preparing datasets, I had to make the questions for the relevant types. I did the evaluation of value mapping module of complex problem module by processing each question mannually.

**154183N – L.A.D.C.U. Rathnaweera**

Implementation of the Sinhala word problem preprocessing, and feature extraction module and simultaneous equation generation based on rules are assigned to me. I have researched about various methods of tackling the challenges associated with the above modules.

In order to implement the understood literature, I had to learn about NLTK, Natural Language processing techniques and Python programming language.

Through my research, I was able to implement up to Sinhala preprocessing module, feature extraction of Sinhala sentences module and simultaneous equation generation from Sinhala sentences module using rules-based templates for the respective types of problems.

## Appendix B

1. N-Gram Tagger Implementation

Tagger Definition

import pickle  
import nltk  
nltk.download('averaged\_perceptron\_tagger')  
from nltk.corpus import TaggedCorpusReader  
  
  
def ngramTagger(train\_sents, n=3, defaultTag='NN'):  
 t0 = nltk.DefaultTagger(defaultTag)  
 if (n <= 0):  
 return t0  
 elif (n == 1):  
 t1 = nltk.UnigramTagger(train\_sents, backoff=t0)  
 return t1  
 elif (n == 2):  
 t1 = nltk.UnigramTagger(train\_sents, backoff=t0)  
 t2 = nltk.BigramTagger(train\_sents, backoff=t1)  
 return t2  
 else:  
 t1 = nltk.UnigramTagger(train\_sents, backoff=t0)  
 t2 = nltk.BigramTagger(train\_sents, backoff=t1)  
 t3 = nltk.TrigramTagger(train\_sents, backoff=t2)  
 return t3  
  
#Get Corpus from resources  
corpus\_1 = TaggedCorpusReader("C:/Users/Chamindui/Desktop/New folder (2)/project/MathSolverRecre/Preprocessing/Resources/Tagger", r" .\*\.txt", encoding="utf-16")  
  
training\_corpus = list(corpus\_1.tagged\_sents('Test\_Corpus.txt'))  
tagger = nltk.PerceptronTagger(load=True)  
#  
# Unigram perceptron tagger initiation  
tagger.train(training\_corpus)  
save\_tagger = open("perceptron\_tagger.pickle", "wb")  
pickle.dump(tagger, save\_tagger)  
save\_tagger.close()

Tagger Usage

import functools  
import operator  
import pickle  
  
import nltk  
  
from MathSolverRecre.Preprocessing.Stemmer.StopWordsFilter import get\_Stop\_Words, filter\_text  
from MathSolverRecre.Preprocessing.Tagger.PosPerceptronTagger import training\_corpus, ngramTagger  
from MathSolverRecre.Preprocessing.Utils.RemoveUnnecessaryChars import delete\_punctuation  
  
nltk.download('state\_union')  
from nltk.corpus import state\_unionUTF  
  
punctuation = ['(', ')', '?', ':', ';', ',', '.', '!', '/', '"', "'", "#", "...", "..", "-"]  
  
testSetPath = "C:/Users/Chamindu/Desktop/New folder (2)/project/MathSolverRecre/Preprocessing/Resources/Book.txt"  
  
sample\_text = state\_unionUTF.raw(testSetPath)  
# print(train\_text)  
  
def convert\_tuples(tup):  
 str = functools.reduce(operator.add, (tup))  
 return str  
  
stopWords = []  
stopWords = get\_Stop\_Words()  
  
# Unigram perceptron tagger Run  
tagger\_f = open("perceptron\_tagger.pickle", "rb")  
tagger = pickle.load(tagger\_f)  
  
# Trigran Tagger initiation  
taggerNgram = ngramTagger(training\_corpus, 3, "NNN")  
  
  
def pos\_tagging(sentence):  
 # POS Tagging with Trigram Tagger  
 tagged = (taggerNgram.tag(sentence))  
  
 # pos Tagging with unigram tagger  
 # tagged = (tagger.tag(sentence))  
  
 # print(tagged) # whole sentence  
 posTaggedOutput = " ".join(word + "/" + tag for word, tag in tagged)  
  
 return posTaggedOutput  
  
  
def pos\_tagging\_array\_type(sentence):  
 # POS Tagging with Trigram Tagger  
 outputTagged = []  
 tagged = (taggerNgram.tag(sentence))  
 for word, tag in tagged:  
 set = [word, tag]  
 output = tuple(set)  
 outputTagged.append(output)  
 return (outputTagged)  
  
  
def pos\_tagging\_list\_return(sentence):  
 # POS Tagging with Trigram Tagger  
 tagged = (taggerNgram.tag(sentence))  
  
 # print(tagged) # whole sentence  
 posTaggedOutput = []  
 for word, tag in tagged:  
 taggedWord = word + "/" + tag  
 posTaggedOutput.append(taggedWord)  
  
 return posTaggedOutput  
  
  
def pos\_tagging\_array\_list\_return(sentence):  
 # POS Tagging with Trigram Tagger  
 tagged = (taggerNgram.tag(sentence))  
  
 # print(tagged) # whole sentence  
 posTaggedOutput = []  
 for word, tag in tagged:  
 taggedWord = []  
 taggedWord.append(word)  
 taggedWord.append(tag)  
 posTaggedOutput.append(taggedWord)  
  
 return posTaggedOutput  
  
  
sentencess = []  
with open(testSetPath, "r", encoding="utf-8") as sentences\_file:  
 for line in sentences\_file:  
 # remove linebreak which is the last character of the string  
 statement = line[:-1]  
 statement = filter\_text(statement)  
 statement = pos\_tagging(statement)  
 sentencess.append(statement)  
  
  
# print(sentencess[2])  
  
def sentence\_tag(sentence):  
 sents = []  
 delete\_punctuation(sentence)  
 for line in sentence:  
 statement = filter\_text(line)  
 statement = pos\_tagging(statement)  
 # print(statement)  
 sents.append(statement)  
 return sents  
  
  
def sentence\_tag\_in\_array\_style(sentence):  
 sents = []  
 output = []  
 # delete\_punctuation(sentence)  
 sentence = nltk.sent\_tokenize(sentence)  
 for line in sentence:  
 statement = filter\_text(line)  
 statement = pos\_tagging\_array\_type(statement)  
 sents.append((statement))  
  
 for i in sents:  
 for j in i:  
 output.append(j)  
 return output  
  
  
def clean\_and\_tag(path):  
 with open(path, "r", encoding="utf-8") as sentences\_file:  
 # delete\_punctuation(sentences\_file)  
 sents = []  
 for line in sentences\_file:  
 # remove linebreak which is the last character of the string  
 statement = line[:-1]  
 statement = filter\_text(statement)  
 statement = pos\_tagging(statement)  
 # print(statement)  
 sents.append(statement)  
 print(sents)

1. HMM Tagger Implementation

Forward Algorithm

import codecs  
from decimal import \*  
  
tag\_list = set()  
tag\_count = {}  
word\_set = set()  
  
  
def get\_TrainSet():  
 print("In Training\n")  
 output\_file = "C:/Users/Chamindu/Desktop/New folder (2)/project/MathSolverRecre/Preprocessing/Resources/HMM.txt"  
 wordtag\_list = []  
  
 input\_file = codecs.open("C:/Users/ Chamindu /Desktop/New folder (2)/project/MathSolverRecre/Preprocessing/Resources/Tagger/Test\_Corpus.txt", mode='r', encoding="utf-16")  
 lines = input\_file.readlines()  
 for line in lines:  
 line = line.strip('\n')  
 data = line.split(" ")  
 wordtag\_list.append(data)  
  
 input\_file.close()  
 print("wordtag\_list", wordtag\_list)  
 return wordtag\_list  
  
  
def calculate\_Trans\_probs():  
 print("In Transition Model")  
 global tag\_list  
 global word\_set  
 train\_data = get\_TrainSet()  
 # print("train data> ",train\_data)  
 transition\_dict = {}  
 global tag\_count  
 for value in train\_data:  
 # print("Value> ",value)  
 previous = "start"  
 for data in value:  
 # print("value> ", value)  
 # print("data> ",data)  
 i = data[::-1]  
 # print("i> ",i)  
 word = data[:-i.find("\_") - 1]  
 # print("word> ",word)  
 fout = codecs.open("C:/Users/ Chamindu /Desktop/New folder (2)/project/MathSolverRecre/Preprocessing/Resources/training\_wordset.txt", mode='a', encoding="utf-8")  
 fout.write(word + "\n")  
  
 word\_set.add(word.lower())  
 data = data.split("/")  
 # print("data",data)  
 tag = data[-1]  
 # print("tag",tag)  
 if (tag.strip() != ''):  
 tag\_list.add(tag.strip())  
 # print("tag list: ",tag\_list)  
  
 # counting tags  
 if tag in tag\_count:  
 tag\_count[tag] += 1  
 else:  
 tag\_count[tag] = 1  
  
 # counting bigram tag sequences  
 if (previous + "-tag-" + tag) in transition\_dict:  
 transition\_dict[previous.strip() + "-tag-" + tag.strip()] += 1  
 previous = tag  
 else:  
 transition\_dict[previous.strip() + "-tag-" + tag.strip()] = 1  
 previous = tag  
 print("tag\_count> ", tag\_count)  
 print("tag\_list> ", tag\_list)  
 print("word\_set> ", word\_set)  
 print("transition\_dict", transition\_dict)  
 return transition\_dict  
  
  
def set\_trans\_probs():  
 count\_dict = calculate\_Trans\_probs()  
 prob\_dict = {}  
 for key in count\_dict:  
 den = 0  
 val = key.split("-tag-")[0]  
 # print("val> ",val)  
 for key\_2 in count\_dict:  
 # print("key\_2> ",key\_2)  
 if key\_2.split("-tag-")[0] == val:  
 den += count\_dict[key\_2]  
 prob\_dict[key] = Decimal(count\_dict[key]) / (den)  
 return prob\_dict  
  
  
def smoothing\_probs():  
 transition\_prob = set\_trans\_probs()  
 for tag in tag\_list:  
 if "start" + tag not in transition\_prob:  
 transition\_prob[("start" + "-tag-" + tag)] = Decimal(1) / Decimal(len(word\_set) + tag\_count[tag])  
 for tag1 in tag\_list:  
 for tag2 in tag\_list:  
 if (tag1 + "-tag-" + tag2) not in transition\_prob:  
 transition\_prob[(tag1 + "-tag-" + tag2)] = Decimal(1) / Decimal(len(word\_set) + tag\_count[tag1])  
 return transition\_prob  
  
  
def calculate\_emi\_probs():  
 # print "In Emission Model"  
 train\_data = get\_TrainSet()  
 count\_word = {}  
 for value in train\_data:  
 for data in value:  
 i = data[::-1]  
 # print("i> ",i)  
 word = data[:-i.find("/") - 1]  
 # print("word> ",word)  
 tag = data.split("/")[-1]  
 # print("tag> ",tag)  
 if word.lower() + "/" + tag in count\_word:  
 count\_word[word.lower() + "/" + tag] += 1  
 else:  
 count\_word[word.lower() + "/" + tag] = 1  
 return count\_word  
  
  
def set\_emi\_probs():  
 # print "In Emission Probability"  
 global tag\_count  
 word\_count = calculate\_emi\_probs()  
 emission\_prob\_dict = {}  
 for key in word\_count:  
 emission\_prob\_dict[key] = Decimal(word\_count[key]) / tag\_count[key.split("/")[-1]]  
 return emission\_prob\_dict  
  
  
def generate\_training\_file():  
 global tag\_count  
  
 transition\_model = smoothing\_probs()  
 emission\_model = set\_emi\_probs()  
  
 fout = codecs.open("C:/Users/ Chamindu /Desktop/New folder (2)/project/MathSolverRecre/Preprocessing/Resources/HMM.txt", mode='w', encoding="utf-8")  
 for key, value in transition\_model.items():  
 # print("key: ", key)  
 # print("value: ", value)  
 fout.write('%s: %s\n' % (key.strip(), value))  
  
 fout.write(u'Emission Model\n')  
 for key, value in emission\_model.items():  
 # print("key: ",key)  
 # print("value: ",value)  
 fout.write('%s:%s\n' % (key.strip(), value))  
  
generate\_training\_file()

Viterbi Algorithm

import codecs  
from decimal import \*  
  
from pip.\_vendor.msgpack.fallback import xrange  
  
tag\_set = set()  
word\_set = set()  
  
  
def get\_train\_data():  
 inputFile = "C:/Users/ Chamindu /Desktop/New folder (2)/project/MathSolverRecre/Preprocessing/Resources/HMM.txt"  
 output\_file = "C:/Users/ Chamindu /Desktop/New folder (2)/project/MathSolverRecre/Preprocessing/ViterbiOutput.txt"  
 transition\_prob = {}  
 emission\_prob = {}  
 tag\_list = []  
 tag\_count = {}  
 global tag\_set  
  
 input\_file = codecs.open(inputFile, mode='r', encoding="utf-8")  
 lines = input\_file.readlines()  
 flag = False  
 for line in lines:  
 line = line.strip('\n')  
 if line != "Emission Model":  
 i = line[::-1]  
 key\_insert = line[:-i.find(":") - 1]  
 # print("key\_insert",key\_insert)  
 value\_insert = line.split(":")[-1]  
 # print("value\_insert", value\_insert)  
 if flag == False:  
 transition\_prob[key\_insert] = value\_insert # getting transition probabilities  
 if (key\_insert.split("-tag-")[0] not in tag\_list) and (key\_insert.split("-tag-")[0] != "start"):  
 tag\_list.append(key\_insert.split("-tag-")[0])  
 else:  
 emission\_prob[key\_insert] = value\_insert # getting emission probabilities  
 key\_tag = line[:-i.find(":") - 1]  
 val = key\_tag.split("/")[-1] # getting the tag  
 j = key\_insert[::-1]  
 word = key\_insert[:-j.find("/") - 1] # getting the word  
 # print("word> ",word)  
 word\_set.add(word.lower())  
 if val in tag\_count:  
 tag\_count[val] += 1 # counting an existing tag  
 else:  
 tag\_count[val] = 1 # counting a new tag  
 tag\_set.add(val)  
  
 else:  
 flag = True  
 continue  
  
 input\_file.close()  
  
 return tag\_list, transition\_prob, emission\_prob, tag\_count, word\_set  
  
  
def viterbi\_process(sentence, tags, transition\_prob, emission\_prob, tag\_count, word\_set):  
 global tag\_set  
 sentence = sentence.strip("\n")  
 word\_list = sentence.split(" ")  
  
  
 current\_prob = {}  
 for tag in tags:  
 tp = Decimal(0)  
 em = Decimal(0)  
 if "start-tag-" + tag in transition\_prob: # calculating probabilities for start tag/word  
 tp = Decimal(transition\_prob["start-tag-" + tag])  
 if word\_list[0].lower() in word\_set:  
 if (word\_list[0].lower() + "/" + tag) in emission\_prob:  
 em = Decimal(emission\_prob[word\_list[0].lower() + "/" + tag])  
 current\_prob[tag] = tp \* em  
 else:  
 em = Decimal(1) / (tag\_count[tag] + len(word\_set)) # for an unknown word  
 current\_prob[tag] = tp  
  
 if len(word\_list) == 1:  
 max\_path = max(current\_prob, key=current\_prob.get)  
 return max\_path  
 else:  
  
 for i in xrange(1, len(word\_list)):  
 previous\_prob = current\_prob  
 # print("i> ",i)  
 # print("previous\_prob",previous\_prob)  
 current\_prob = {}  
 locals()['dict{}'.format(i)] = {}  
 previous\_tag = ""  
 for tag in tags:  
 # print(word\_list)  
 if word\_list[i].lower() in word\_set:  
 # print("word> ",word\_list[i].lower())  
 if word\_list[i].lower() + "/" + tag in emission\_prob:  
 # print("has emission")  
 em = Decimal(emission\_prob[word\_list[i].lower() + "/" + tag])  
 max\_prob, previous\_state = max((Decimal(previous\_prob[previous\_tag]) \* Decimal(transition\_prob[previous\_tag + "-tag-" + tag]) \* em, previous\_tag) for previous\_tag in previous\_prob)  
 current\_prob[tag] = max\_prob  
 locals()['dict{}'.format(i)][previous\_state + "-" + tag] = max\_prob  
 previous\_tag = previous\_state  
 # print("previous\_tag",previous\_tag)  
 else: # for unknown words  
 em = Decimal(1) / (tag\_count[tag] + len(word\_set))  
 # print("jdkdjkhk",tag)  
 max\_prob, previous\_state = max((Decimal(previous\_prob[previous\_tag]) \* Decimal(transition\_prob[previous\_tag + "-tag-" + tag]) \* em, previous\_tag) for previous\_tag in previous\_prob)  
 current\_prob[tag] = max\_prob  
 locals()['dict{}'.format(i)][previous\_state + "-" + tag] = max\_prob  
 previous\_tag = previous\_state  
 if i == len(word\_list) - 1:  
 max\_path = ""  
 last\_tag = max(current\_prob, key=current\_prob.get)  
 max\_path = max\_path + last\_tag + " " + previous\_tag  
 for j in range(len(word\_list) - 1, 0, -1):  
 for key in locals()['dict{}'.format(j)]:  
 data = key.split("-")  
 if data[-1] == previous\_tag:  
 max\_path = max\_path + " " + data[0]  
 previous\_tag = data[0]  
 break  
 result = max\_path.split()  
 result.reverse()  
 return " ".join(result)  
  
def HMM\_tagging():  
 tag\_list, transition\_model, emission\_model, tag\_count, word\_set = get\_train\_data()  
 input\_file = codecs.open("C:/Users/ Chamindu /Desktop/New folder (2)/project/MathSolverRecre/Preprocessing/Resources/Book.txt", mode='r', encoding="utf-8")  
 fout = codecs.open("C:/Users/ Chamindu /Desktop/New folder (2)/project/MathSolverRecre/Preprocessing/ViterbiOutput.txt", mode='w', encoding="utf-8")  
 for sentence in input\_file.readlines():  
 # print("New Sentence\n")  
 path = viterbi\_process(sentence.strip(), tag\_list, transition\_model, emission\_model, tag\_count, word\_set)  
 # print("path> ",path)  
 sentence = sentence.rstrip('\r\n')  
 # print("sentence> ",sentence)  
 word = sentence.strip().split(" ")  
 # print("word> ",word)  
 tag = path.split(" ")  
 sentence=[]  
 # print("tag> ",tag)  
 for j in range(0, len(word)):  
 if j == len(word) - 1:  
 # print("OO> ",word[j])  
 temp = word[j] + "/" + tag[j] + u'\n'  
 fout.write(temp)  
 sentence.append(temp)  
 else:  
 temp=word[j] + "/" + tag[j] + " "  
 fout.write(temp)  
 sentence.append(temp)  
 print(sentence)

Template Rules for type 01

import collections  
  
  
# Method to split a list by given values  
def split\_list\_by\_Val(givenList, value):  
 size = len(givenList)  
 idx\_list = [idx + 1 for idx, val in  
 enumerate(givenList) if val == value]  
  
 res = [givenList[i: j] for i, j in  
 zip([0] + idx\_list, idx\_list +  
 ([size] if idx\_list[-1] != size else []))]  
 return res  
  
  
def Type01\_solve(tagged, words):  
 duplicated = []  
 subs = []  
 required = []  
 eddited = []  
 NNPI = "ගණන"  
 print(tagged)  
 for i in range(0, len(words)):  
  
 # Common\_Rule 01 - Get all the potential variables from the problem  
 if (tagged[i][1] == "NNP" or tagged[i][1] == "ATT" or tagged[i][1] == "NNN"):  
 subs.append(words[i])  
 duplicated = ([item for item, count in collections.Counter(subs).items() if count > 1])  
  
 # Common\_Rule 02 - Remove NNPI values  
 if (tagged[i][0] == "මිල" or tagged[i][0] == "ගණන"):  
 temp = ("NULL", "NNN")  
 tagged[i] = temp  
 words[i] = "NULL"  
  
 # Common\_Rule 03 - Get noun and "මිල" as a single noun( Eg: මේසයේමිල,බංකුවේමිල)  
 if ((tagged[i][1] == "NNN" or tagged[i][1] == "PRP") and (tagged[i + 1][0] == "මිල")):  
 word = tagged[i][0] + tagged[i + 1][0]  
 tagged[i] = (word, "NNN")  
 tagged[i + 1] = tuple("NULL" + "NNN")  
 temp = str(words[i + 1])  
 # Remove the next NNN after recording it in Required array  
 if temp not in required:  
 required.append(temp)  
 words[i] = word  
 words[i + 1] = "NULL"  
  
 # Common\_Rule 04 - Get potential variables with attributes( Eg: කුඩාතැපැල්පත්,විශාලතැපැල්පත්)  
 if ((tagged[i][1] == "ATT" or tagged[i][1] == "JJ") and (tagged[i + 1][1] == "NNN" or tagged[i][1] == "NNP")):  
 word = tagged[i][0] + tagged[i + 1][0]  
 tagged[i] = (word, "NNN")  
 tagged[i + 1] = tuple("NULL" + "NNN")  
 temp = str(words[i + 1])  
 # Remove the next NNN after recording it in Required array  
 if temp not in required:  
 required.append(temp)  
 words[i] = word  
 words[i + 1] = "NULL"  
  
 # Common\_Rule 05 - Add "," after word "අතර"  
 if (tagged[i][0] == "අතර"):  
 words.insert(i + 1, ",")  
 temp = (",", ".")  
 tagged.insert(i + 1, temp)  
  
 # Common\_Rule 06 - Add "," and replace with "."  
 if (tagged[i][0] == ","):  
 words.insert(i + 1, ".")  
 temp = (".", ".")  
 tagged.insert(i + 1, temp)  
  
 # Common\_Rule 07 - Remove 'ගුණයක්' when dual multiplication is in the problem  
 if ((tagged[i][1] == "CD" or tagged[i][1] == "QFNUM" or tagged[i][1] == "VAR") and (  
 tagged[i + 1][0] == "ගුණයක්")):  
 tagged[i + 1] = tuple("NULL" + "NNN")  
 words[i + 1] = "NULL"  
  
 # Common\_Rule 08 - Remove 'ක්' when it occurs after a number or variable  
 if ((tagged[i][1] == "CD" or tagged[i][1] == "QFNUM" or tagged[i][1] == "VAR") and (  
 tagged[i + 1][0] == "ක්")):  
 tagged[i + 1] = tuple("NULL" + "NNN")  
 words[i + 1] = "NULL"  
  
 # Common\_Rule 09 - Remove NNPI values  
 if (tagged[i][1] == "NNPI"):  
 NNPI = tagged[i][0]  
 temp = ("NULL", "NNN")  
 tagged[i] = temp  
 words[i] = "NULL"  
  
 # Common\_Rule 10 - Include identified variables to Required Variables Array  
 for i in duplicated:  
 required.append(i)  
  
 # Create the final version of Extracted problem  
 for i in range(0, len(words)):  
 if words[i] in required:  
 eddited.append(tagged[i])  
  
 # Common\_Rule 11 - Remove NULLs from the sentences  
 if words[i] == "NULL":  
 continue  
  
 elif (words[i] not in subs):  
 eddited.append(tagged[i])  
  
 splittedEdit = split\_list\_by\_Val(eddited, ('.', '.'))  
 print(splittedEdit)  
  
 variCounter = 0  
 varPuts = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'i', 'j', 'k', 'l', 'm']  
 variablesReplace = []  
 variables = []  
 finalExp = []  
 for item in splittedEdit:  
 words\_inuse = []  
 for i in range(0, len(item)):  
 words\_inuse.append(item[i][0])  
  
 for i in range(0, len(item)):  
  
 # Template Rule 01 - Map variables with expressions  
 if (item[i][1] == "NNP" or item[i][1] == "ATT" or item[i][1] == "NNN"):  
 # Check for Duplication  
 if (words\_inuse[i] not in variables):  
 variables.append(words\_inuse[i])  
 temp = (words\_inuse[i], varPuts[variCounter])  
 variablesReplace.append(temp)  
 variCounter += 1  
  
 # Template Rule 02- Replace with mapped letters  
 if (words\_inuse[i] in variables):  
  
 # Template Rule 03 = Exclude 'ගණන' from being a variable  
 if (words\_inuse[i] != 'ගණන'):  
 reqVar = ''  
 for k in range(0, len(variablesReplace)):  
 if (words\_inuse[i] == variablesReplace[k][0]):  
 reqVar = str(variablesReplace[k][1])  
 words\_inuse[i] = reqVar  
 temp = (words\_inuse[i], "VAR")  
 item[i] = temp  
  
 # Template Rule 04 - Tackle "එකතුව" and replace with " Var1 + Var2"  
 if ((item[i][0] == "එකතුව" and (item[i + 1][1] == "CD" or item[i + 1][1] == "QFNUM") and (  
 not (variablesReplace)))):  
 # Template Rule 05 - Replace "එකතුව" with "+"  
 exp = str(variablesReplace[0][1] + " + " + variablesReplace[1][1])  
 words\_inuse[i] = exp  
 temp = (exp, "EXP")  
 item[i] = temp  
 # Put Equivalance to expression  
 words\_inuse.insert(i + 1, "=")  
 temp = ("=", "EQ")  
 item.insert(i + 1, temp)  
  
 # Template Rule 05 - Replace multiplication  
 if (item[i][1] == "MUL"):  
 words\_inuse[i] = "\*"  
 item[i] = ("\*", "MUL")  
  
 # Template Rule 06 - Replace Addition with "හා" and "සහ"  
 if (item[i][1] == "ADD" or item[i][0] == "හා" or item[i][0] == "සහ"):  
 words\_inuse[i] = "+"  
 temp = ("+", "ADD")  
 item[i] = temp  
  
 # Template Rule 07 - Replace Addition with "හා" and "සහ"  
 if (item[i][1] == "SUB"):  
 words\_inuse[i] = "-"  
 temp = ("-", "SUB")  
 item[i] = temp  
  
 # Template Rule 07- Apply multiplication and generate Variable  
 if ((item[i][1] == "VAR") and (item[i + 1][1] == "MUL") and (  
 item[i + 2][1] == "CD" or item[i + 2][1] == "QFNUM" or item[i + 2][1] == "VAR")):  
 multiplied = str(item[i + 2][0]) + str(item[i][0])  
 temp = (multiplied, "VAR")  
 words\_inuse[i] = multiplied  
  
 words\_inuse[i + 1] = words\_inuse[i + 2] = "NULL"  
 item[i] = temp  
 item[i + 1] = item[i + 2] = ("NULL", "NULL")  
  
 words\_inuse.insert(i + 1, "=")  
 temp = ("=", "EQ")  
 item.insert(i + 1, temp)  
  
 # Template Rule 08- Apply multiplication and generate Variable with "මෙන්"  
 if ((item[i][0] == "\*") and (item[i + 1][1] == "CD") and (  
 item[i + 2][1] == "NNN" and (item[i + 2][0] in variables) and (item[i + 2][0] != "ගණන"))):  
 for k in range(0, len(variablesReplace)):  
 if (words\_inuse[i + 2] == variablesReplace[k][0]):  
 result = str(variablesReplace[k][1])  
 multiplied = str(item[i + 1][0]) + "" + result  
 temp = (multiplied, "VAR")  
 words\_inuse[i] = multiplied  
  
 words\_inuse[i + 1] = words\_inuse[i + 2] = "NULL"  
 item[i] = temp  
 item[i + 1] = item[i + 2] = ("NULL", "NULL")  
  
 words\_inuse.insert(i, "=")  
 temp = ("=", "EQ")  
 item.insert(i, temp)  
  
 # Template Rule 09- Apply multiplication  
 if (((item[i][1] == "CD") or (item[i][1] == "VAR")) and (item[i + 1][1] == "EQ") and (  
 item[i + 2][0] == item[i + 3][0] == "NULL") and item[i + 4][0] == "."):  
 exp = "= " + item[i][0]  
 temp = (exp, "EXP")  
 words\_inuse[i] = exp  
 item[i] = temp  
 words\_inuse[i + 1] = "NULL"  
 item[i + 1] = ("NULL", "NULL")  
  
 # Template Rule 10 - Identifying Equivalance method.  
 if ((item[i][1] == "VAR") and (  
 item[i + 1][1] == "CD" or item[i + 1][1] == "QFNUM" or item[i + 1][1] == "VAR")):  
 words\_inuse.insert(i + 1, "=")  
 temp = ("=", "EQ")  
 item.insert(i + 1, temp)  
  
 # Template Rule 11 - Identifying Equivalance method with "නම්" word  
 if (item[i][0] == "නම්"):  
 temp = ("=", "EQ")  
 item[i] = temp  
 words\_inuse[i] = "="  
 print(item)  
  
 # Build Expressions  
 for item in splittedEdit:  
 print(item)  
 finalPart = []  
 for i in range(0, len(item)):  
 # Get only required things to build Expression  
 if ((item[i][1] == "ADD" or item[i][1] == "VAR" or item[i][1] == "SUB" or item[i][1] == "EQ") or (item[i][  
 1] == "MUL" or  
 item[i][  
 1] == "CD" or  
 item[i][  
 1] == "QFNUM" or  
 item[i][  
 1] == "EXP")):  
 finalPart.append(item[i][0])  
 if finalPart not in finalExp:  
 finalExp.append(finalPart)  
  
 print("----------------------------------------------------------------------------------------")  
 print(variablesReplace)  
 print(finalExp)  
 print(NNPI)  
 print("----------------------------------------------------------------------------------------")