**Digital Video Making and Smart Answering System**

**Zip2**

**Index Number Name**

164007P L.Araharan

164064K M.Jeevahasan

164024N A.Dhushanthini

Faculty of Information Technology

University of Moratuwa

2021

**Digital Video Making and Smart Answering System**

**Zip2**

**Index Number Name**

164007P L.Araharan

164064K M.Jeevahasan

164024N A.Dhushanthini

Dissertation submitted to the Faculty of Information Technology,

University of Moratuwa, Sri Lanka, for the partial fulfilment of the requirements of

The Honors Degree of Bachelor of Science in

Information Technology

2021

**Declaration**

We hereby declare that the project work entitled Integrated Platform for “Digital Video Making and Smart Answering System” submitted to University of Moratuwa, Faculty of Information Technology is a record of an original work conducted by Team Zip2, under the supervision of below mentioned supervisor. This project work has not been submitted in any form for another degree or diploma at any University or other Institute of tertiary education. Information derived from published and unpublished work of others have been acknowledged where necessary giving due credit to the original author.

| **Group Members** | | |
| --- | --- | --- |
| Name | Index No | Signature |
| L.Araharan | 164007P |  |
| M.Jeevahasan | 164064K |  |
| A.Dhushanthini | 164024N |  |
| **Supervisor** | | |
| **Name** | **Signature** |  |
| Dr. Sagar Sumathipala |  |  |

**Acknowledgement**

We wish to express our sincere gratitude to Dr. Sagara Sumathipala, for his endeavor, approach, motivation, patience and outstanding supervision which make it possible to apply theoretical knowledge in practical applications and development of this research project. His guidance leads us throughout this project and preparation of this report. We would also like to thank the academic staff of the Faculty of Information Technology for their support and further help in developing this project. Again, we express our sincere thanks to all who strength us with their support to accomplish this research project.

**Abstract**

Video learning is a highly-accepted learning method in the modern world. Human brain captures videos faster than texts. So kids can easily learn through videos rather than textbooks. Kids' learning ability increases when they learn through videos. Learning through videos increases kids' interests and attention in education. It provides an efficient and effective learning experience.

Manually making videos is time consuming and requires technical knowledge for editing. Digital video making for an audio content process takes several steps like summarizing the content, searching images on the internet and other edits to sync the searched images with the audio. Also it takes several weeks. People do not have much time to spend on video making. There are many monologues which explain the most important content without video streaming.

In order to solve this issue we have come up with a solution Digital Video Making and Question and Answering System. Anyone can produce a video by using audio content. In addition kids can engage with video and ask questions to get the answers. Through our research we are going to develop a system which can produce video by combining a set of images from google for an educational related content with a question and answering system.

This research involves the implementation of technologies related to Natural Language Processing, Deep Learning, especially the inclusion of Convolutional Neural Networks, Sentimental Analysis and Transfer Learning. There are three main modules in this research. They are Content Modification using Summarization with Lexical Simplification, Information extraction from sentences and Question answering and Information retrieval System.

**Table of Contents**

[**Chapter 01**](#_heading=h.26in1rg) **10**

[**Introduction**](#_heading=h.1ksv4uv) **10**

[1.1 Introduction](#_heading=h.2jxsxqh) 10

[1.2 Background and motivation](#_heading=h.z337ya) 11

[1.3 Aim and Objective](#_heading=h.1y810tw) 12

[*1.3.1 Aim*](#_heading=h.4i7ojhp) *12*

[*1.3.2 Objective*](#_heading=h.2xcytpi) *12*

[1.4 Proposed Solution](#_heading=h.4c7h44s3v45w) 13

[1.5 Summary](#_heading=h.3whwml4) 14

[**Chapter 02**](#_heading=h.2bn6wsx) **14**

[**Review of related Work**](#_heading=h.qsh70q) **14**

[2.1 Introduction](#_heading=h.3as4poj) 14

[2.2. Other’s work on individual modules](#_heading=h.1pxezwc) 14

[*2.2.1. Content Modification using Summarization with Lexical Simplification.*](#_heading=h.23ckvvd) *14*

[*2.2.2 Information extraction from sentences*](#_heading=h.vx1227) *17*

[*2.2.3 Predicting the similar sentence and finding the most appropriate answer*](#_heading=h.1v1yuxt) *19*

[2.3 Summary](#_heading=h.37m2jsg) 23

[**Chapter 03**](#_heading=h.1egqt2p) **24**

[**Technologies Adapted**](#_heading=h.3cqmetx) **24**

[3.1 Introduction](#_heading=h.1rvwp1q) 24

[3.2 Technology Required](#_heading=h.4bvk7pj) 24

[*3.2.1 Python*](#_heading=h.2r0uhxc) *24*

[*3.2.2 NLTK*](#_heading=h.1664s55) *24*

[*3.2.3 Pandas*](#_heading=h.3q5sasy) *25*

[*3.2.4 NumPy*](#_heading=h.25b2l0r) *25*

[*3.2.5 Bi-LSTM*](#_heading=h.kgcv8k) *25*

[*3.2.6 BERT*](#_heading=h.34g0dwd) *25*

[*3.2.7 Scikit-learn*](#_heading=h.1jlao46) *25*

[*3.2.8 Flask*](#_heading=h.43ky6rz) *26*

[*3.2.9 Torch*](#_heading=h.2iq8gzs) *26*

[*3.2.10 Transformers*](#_heading=h.xvir7l) *26*

[*3.2.11 Infersent*](#_heading=h.3hv69ve) *26*

[*3.2.12 Spacy*](#_heading=h.1x0gk37) *26*

[*3.2.13 PyCharm*](#_heading=h.4h042r0) *26*

[*3.2.14 Keras*](#_heading=h.2w5ecyt) *26*

[*3.2.15 Google Colab*](#_heading=h.1baon6m) *26*

[*3.2.16 Selenium*](#_heading=h.3vac5uf) *27*

[*3.2.18 MoviePy*](#_heading=h.pkwqa1) *27*

[3.3 Summary](#_heading=h.39kk8xu) 27

[**Chapter 04**](#_heading=h.1opuj5n) **28**

[**Our Approach**](#_heading=h.48pi1tg) **28**

[4.1 Introduction](#_heading=h.2nusc19) 28

[4.2 Our Approach](#_heading=h.1302m92) 28

[*4.2.1 Content modification using summarization with lexical simplification*](#_heading=h.3mzq4wv) *28*

[*4.2.2 Information extraction from sentences*](#_heading=h.2250f4o) *29*

[*4.2.3 Predicting the similar sentence and finding the most appropriate answer*](#_heading=h.haapch) *30*

[4.3 Summary](#_heading=h.319y80a) 31

[**Chapter 05**](#_heading=h.pi4usb6abrg) **31**

[**Analysis and Design**](#_heading=h.40ew0vw) **31**

[5.1 Introduction](#_heading=h.2fk6b3p) 31

[5.2 Content modification using summarization with lexical simplification](#_heading=h.upglbi) 32

[5.3 Information extraction from sentences](#_heading=h.4du1wux) 34

[5.4 Predicting the similar sentence and finding the most appropriate answer](#_heading=h.2szc72q) 36

[5.5 Summary](#_heading=h.279ka65) 41

[**Chapter 06**](#_heading=h.meukdy) **42**

[**Implementation**](#_heading=h.36ei31r) **42**

[6.1 Introduction](#_heading=h.1ljsd9k) 42

[6.2 Our Introduction](#_heading=h.45jfvxd) 42

[*6.2.1 Content Modification using summarization with lexical simplification*](#_heading=h.2koq656) *42*

[*6.2.2 Information extraction from sentences*](#_heading=h.3jtnz0s) *46*

[6.2.2.1 Collecting Dataset](#_heading=h.r4pqacj8b5ze) 47

[6.2.2.2 Features](#_heading=h.ue8dvsrwcdpj) 47

[6.2.2.3 Tokenizing](#_heading=h.7ew0oeeurinb) 49

[6.2.2.4 Training](#_heading=h.1uhpmtzdqbvm) 49

[6.2.2.5 Video making](#_heading=h.hz42i0dgzkhs) 50

[*6.2.3 Predicting the similar sentence and finding the most appropriate answer*](#_heading=h.2y3w247) *50*

[6.2.3.1 Collecting Dataset](#_heading=h.7x26jijc8jx3) 51

[6.2.3.2.3 Missing Value](#_heading=h.fa4m7cd8yb2) 54

[6.2.3.2.3 Training](#_heading=h.49q1bet6nffu) 55

[6.2 Summary](#_heading=h.4i4uc2n3fma0) 58

[**Chapter 07**](#_heading=h.3x8tuzt) **59**

[**Evaluation**](#_heading=h.2ce457m) **59**

[7.1 Introduction](#_heading=h.rjefff) 59

[7.2 Content modification using summarization with lexical simplification](#_heading=h.3bj1y38) 59

[7.3 Information extraction from sentences](#_heading=h.xc4o612gbrc7) 61

[7.4 Outcomes of Predicting the similar sentence and finding the most appropriate answer](#_heading=h.4anzqyu) 62

[Conclusion and Further Work](#_heading=h.2pta16n) 63

[**References**](#_heading=h.14ykbeg) **64**

[**Appendix A: Individual Contribution to the Project**](#_heading=h.3oy7u29) **68**

[Name of Student: Dhushanthini.A 164024N](#_heading=h.91k5ab5v1k4x) 69

[Name of Student: M.Jeevahasan 164064K](#_heading=h.8bayfqc3fyfq) 69

[Name of Student: L.Araharan 164007P](#_heading=h.4qx07pgrxgt0) 70

[**Appendix B - Code Implementations**](#_heading=h.7romvbeayhik) **71**

[Content modification using summarization with lexical simplification](#_heading=h.pl0u5rmg3urp) 71

[Information extraction from sentences](#_heading=h.1nrb7k44caz3) 108

[Predicting the similar sentence and finding the most appropriate answer](#_heading=h.6joryd28anjp) 121

List of Figures

[*Figure 1: 5.1 High level diagram for approach*](about:blank) *26*

[*Figure 2: 5.2 High level diagram for Summarization Process*](about:blank) *27*

[*Figure 3: 5.3 High Level Diagram for Lexical Simplification 28*](#_heading=h.3ep43zb)

[*Figure 4: 5.4 High level diagram of content modification using summarization with lexical simplification*](about:blank) *29*

[*Figure 5: 5.5 Labeling the children sentences datasets*](about:blank) *30*

[*Figure 6: 5.6 System design for finding sentence similarity 32*](#_heading=h.184mhaj)

[*Figure 7: 5.7 System design for finding appropriate answer 33*](#_heading=h.3s49zyc)

[*Figure 8: 6.1 CWI dataset 36*](#_heading=h.zu0gcz)

[*Figure 9: 6.2 Sample data of children sentences dataset 39*](#_heading=h.1yyy98l)

[*Figure 10: 6.3 Labeling the children sentences dataset 40*](#_heading=h.4iylrwe)

[*Figure 11: 6.4 Parse tree obtained for features extraction*](about:blank) *44*

[*Figure 12: 6.5 Sample extracted feature based on cosine similarity and Euclidean distance*](about:blank) *45*

# Chapter 01

# Introduction

## 1.1 Introduction

Visually telling a content has become an important factor in education, entertainment, and commerce. Many new products are developed and launched in this industry. The COVID-19 pandemic has affected nearly all social areas, including the education domain. Teaching has become more difficult because of the distance-based learning systems. In the challenging situation of digital learning caused by the COVID-19 restrictions, learning outcomes of the students vary greatly. Most of the Schools and institutions still use old techniques to educate students. Most of the students consider this way of educating students as not effective learning because of the way it was presented. Education institutes should focus on other educational systems which provide better learning outcomes for the students. In this case, Self-learning should be encouraged. To mitigate these challenges, we must investigate other opportunities which will give solutions to these problems.

Children can more easily understand content through a video than by listening to the audio or reading a book. Because it is efficient, entertaining to watch and easily conveys the meaning of the content. So, most people make videos to explain their content. This process takes several steps like summarizing the content, searching images on the internet and other edits to sync the searched images with the audio. Also, it takes several weeks. People do not have much time to spend on video making. There are many monologues which explain the most important content without video streaming.

Addition to that, there are many AI-based Text-To-Video or Audio-To-Video products in the market such as Luman5, Glia Cloud, Wichita, Wibbitz and Video which are used to create videos from text or URL content. But those systems are not focused on educational purposes and for children. Also, there is not any system to produce video which is made through images from audio with question and answering features.

The growth of interest in data analysis as information retrieval in the education domain becomes unavoidable. In this research we have made a system for kids which can produce a video by combining a set of images for a text content and a question and answering system. Also, kids can engage with the video and ask questions and get the answers. Anyone without proper training can easily make video content within minutes.

## 1.2 Background and motivation

The educational system has modernized globally with the emergence of cutting-edge technologies. Devices that use low-end to high-end technology, let the educators harness the digital world utilizing easy-to-use software. As an impact of technology, educators not only are able to develop their skills, knowledge but also play a significant role in magnifying the standard of education. Still, the education sector confronts many difficulties: one of them is increasing the interaction between students and technology to get better educational outcomes. It has become necessary to use innovative pedagogical methods to involve learners.

Children naturally attract books and podcasts as they will bring more creative ideas, and introduce them to places, and technologies they haven't heard before. Visually demonstrating content is an innovative pedagogical approach that can let the students in deep and meaningful learning, and it promotes children to be artistic and use their imagination to picture the whole scenario as it reveals [27]. These kinds of contents inspire your child to raise questions as the plot progresses, instead of just reading passively. Asking questions means children are thinking beyond the content and it would help them develop creativity and critical thinking skills.

It is a beneficial way for creating a constructivist learning atmosphere based on novel principles of educating and learning. Hence, this method has the potential to intensify students' involvement and provide better results for learners. But video making is a time-consuming and cost-effective process. The post-production part of the process takes the longest. There are many steps, including logging and backing up the footage, synchronizing the audio with the relevant image content, and running through the footage itself. To ease this process, automating the video editing processes as much as we can, could be more efficient, and raising questions and seeking answers improves our knowledge capability because it makes us active learners. When you analyze a piece of information by elaborating on it, thinking logically about its core, or relating some pieces of information to others, you increase the chance of learning actively.

## 1.3 Aim and Objective

### 1.3.1 Aim

Aim of this project is to develop a system for kids which can produce a video by combining a set of images for a text content with a smart question and answering system..

### 1.3.2 Objective

**Objectives**

* Extract the important information from the content.
* Identify the complex words from the extracted information and substitute it with simple words. (change the content to kids specific domain)
* Information extraction from sentences through sentiment analysis and semantic analysis.
* Age categorizing the output by comparing to the sentences used in the input.
* Design and develop a Question Answering System

## 

## 1.4 Proposed Solution

Our proposed solution will help the people to minimize the time and cost spent in digital content making in the educational domain. The system will provide an effective way to make a video for given audio or text, and answer questions effectively .We have divided the system into three modules. Figure 1 shows the interaction among these modules.

1. Content modification using Summarization with Lexical Simplification.
2. Information extraction from sentences through sentiment analysis and semantic analysis.
3. Question Answering System

Content modification is done to extract the important information using summarization techniques from original content and remove the complex words and substitute it with simple words using lexical simplification. Transfer learning and deep learning approaches have been used for this module. After this content modification, modified content can be more understandable for kids.

From the modified content, Information extraction will happen. This technique is used to extract meaningful information from the content. This module consists of text analysis and sentimental analysis tasks. After extracting meaningful information from the content, meaningful information along with the domain of the content will be used to identify the relevant image for each sentence.

The third module is used to create a question answering system. User can interact with the video by asking questions to retrieve information. This technique is used for effective learning outcomes to identify relevant answers for the questions. This module consists of getting the sentence having the right answer for a given question and finding the correct answer from the context.

## 1.5 Summary

This chapter gives an introduction to our project. Background and motivation as well as proposed solutions are also discussed in this chapter. Chapter 02 discusses the literature review. Chapter 03 discusses the technologies that we are going to adapt in making this project. Chapter 04 describes our approach in solving the problem. Chapter 05 discusses the analysis and design. In this section we are describing a model for every component. Chapter 06 describes the implementation. In the next chapter we wrote about the discussion which summarized our report and discussed the issues and challenges. Lastly we have written the Conclusion and further work for our project.

# Chapter 02

# Review of related Work

## 2.1 Introduction

In the previous chapter, we discussed what problems we are facing in the implementation and the solution of those problems in brief. In this chapter, we briefly discuss some literature work that is related to our solution. As far as we have studied some literature, there is no work similar to our project of Digital Video Making and Smart Answering System in Educational domain. But, there is some research that is similar to some parts. So, we are choosing some of those concepts from the literature and building up the solution for our problem.

## 2.2. Other’s work on individual modules

### 2.2.1. Content Modification using Summarization with Lexical Simplification.

For our research project in final year, as a group we selected a topic which is Digital Video making with a Question and Answering system. According to our group we have divided the main as three topics. Those are Content Modification using Summarization with Lexical Simplification, Information Extraction from sentences and question and answering system. As a group member I have selected Content Modification using Summarization with Lexical Simplification.

In our project, summarization is used to shorten the original content and lexical simplification used to identify the complexity of words and substitute with simple words. Text Summarization is the process of reducing the source text into shorter versions to preserve its information content and overall meaning [1]. The summary should be short and accurate. It is broadly classified into extractive and abstractive methods.

The extractive summarization requires statistical, linguistics and heuristics methods for ranking the sentences. Many techniques have been developed for summarization of text in various languages. Text abstraction understands the main concepts in a text and generates a new shorter text with the same content. The abstractive summarization requires natural language generation techniques. Abstractive summarization techniques are broadly classified into structure based approach and semantic based approach. Structure based methods abstract the most important information through cognitive theories [2]. In our project I chose the Abstractive Summarization method. Because, it is a more human-like way of generating summaries and these summaries are more effective as compared to the extractive approaches.

The sequence to sequence encoder-decoder architecture is the base for sequence transduction tasks. It essentially suggests encoding the complete sequence at once and then using this encoding as a context for the generation of decoded sequence or the target sequence.

The work on sequence to sequence models [4] and seq2seq with neural networks [3] opened up new possibilities for neural networks in natural language processing. From 2014 to 2015, LSTMs (a variety of RNN) became the dominant approach in the industry which achieved state of the art results. Such architectural changes became successful in tasks such as speech recognition, machine translation, parsing, and image captioning. The results of this paved the way for abstractive summarization, which began to score competitively against extractive summarization. Applying attention mechanisms with transformers became more dominant for tasks, such as translation and summarization.

In abstractive video summarization, models which incorporate variations of LSTM and deep layered neural networks have become state of the art performers. In addition to textual inputs, recent research in multi-modal summarization incorporates visual and audio modalities into language models to generate summaries of video content. However, generating compelling summaries from conversational texts using transcripts or a combination of modalities is still challenging.

The main issue with RNNs lies in their inability to provide parallelization while processing. The processing of RNN is sequential, i.e. we cannot compute the value of the next time step unless we have the output of the current. This makes RNN-based approaches slow. So, these problems are solved in Transformers [5]. The Transformer uses the self-attention mechanism where attention weights are calculated using all the words in the input sequence at once, hence it facilitates parallelization. Transformer uses self-attention as a means for effective computation.

Lexical Simplification (LS) is replacing complex words with simpler alternatives, which can help various groups of people, like children [6], non-native speakers [7], etc, to better understand a given text. LS is an effective way of simplifying a text because some work shows that those who are familiar with the vocabulary of a text can often understand its meaning even if the grammatical constructs used are confusing to them. The LS framework is commonly framed as a pipeline of three steps: complex word identification (CWI), substitute generation (SG) of complex words, and filtering and substitute ranking (SR). CWI is often treated as an independent task [8]. Existing LS systems mainly focused on the two steps (SG and SR) [9].

Early systems of LS often used standard statistical machine translation approaches to learn the simplification of a complex sentence into a simplified sentence [10]. Recently, LS methods adopted an encoder-decoder model to simplify the text based on parallel corpora [11]–[13]. Lexical simplification (LS) only focuses on simplifying complex words of one sentence. LS needs to identify complex words and find the best candidate substitution for these complex words [14], [15]. The best substitution needs to be more simplistic while preserving the sentence grammatically and keeping its meaning as much as possible, which is a very challenging task.

The popular lexical simplification approaches were rule-based, in which each rule contains a complex word and its simple synonyms [18], [16], [17]. Rule-based systems usually identified synonyms from WordNet or other linguistic databases for a predefined set of complex words and selected the ”simplest” from these synonyms based on the frequency of word or length of word [6], [19]. Previous research in LS done different target audiences like foreign and second language learners [20] and people with cognitive disability [21].

There is more existing research on summarization and lexical simplification. But our module is integration of summarization and lexical simplification for specifically kids' domains and we are building this model for spoken words. So kids can easily understand the content.

### 2.2.2 Information extraction from sentences

Artificial intelligent methods and machine learning algorithms are used in automatic information extraction and it achieves high performance. The most basic techniques are syntactic rules and basic Nature Language Processing techniques under all used techniques. Fine information from text is extracted by using some syntactic rules and patterns at the word level.

Artificial intelligent methods and machine learning algorithms are used in automatic information extraction and it achieves high performance. The most basic techniques are syntactic rules and basic Nature Language Processing techniques under all used techniques. Fine information from text is extracted by using some syntactic rules and patterns at the word level.

In our project, information extraction is used to find the relevant searching queries which are suitable for the sentence within the domain. To extract the information sentimental analysis, text analysis, semantic analysis is mainly used. A basic task in sentiment analysis is classifying the polarity, motion Detection, Intent Detection; Text Analysis is about parsing texts in order to extract machine-readable facts from them; Semantic analysis is the process of relating syntactic structures, from the levels of phrases, clauses, sentences and paragraphs to the level of the writing as a whole, to their language-independent meanings.

Sentiment classification is a form of text classification in which classify the text into predefined sentiment classes (very positive, positive, neutral, negative, very negative). These five classes in fine-grained sentiment classification. A lot of research and progress is going on solving Sentiment classification because it is one of the most popular tasks in NLP. Binary sentiment classification is most focused (Positive and negative are the possible classes in binary sentiment classification) [23]. Text is converted into a fixed-size vector is the first step in sentiment classification of a text. It is called Embedding. Researchers first tackled the problem of learning word embedding since the number of words in the vocabulary after tokenization and stemming is limited [24]. Combining a variable number of word vectors into a single fixed-size document vector is the next step. Recent language model research has been trying to train contextual embedding. Peters et al. [25] extracted context-sensitive features from left to right and right to left LSTM-based language model. To train deep bidirectional representations from unlabeled texts, Devlin et al. [26] proposed BERT (Bidirectional Encoder Representations from Transformers), an attention-based Transformer architecture. Their architecture obtains state of the art results on many NLP tasks and allows a high degree of parallelism since it is not based on sequential or recurrent connections. These sentiment classifications are not mainly focused on children's education purposes.

To have subjects read textual material and then have them produce some form of summary, such as answering questions or writing an essay is the typical approach in text-comprehension research. Information the subject has gained from the text is determined by the experimenter. To relate what was in the summary and what the subject has read is to analyze what a subject has learned from a text, the task of the experimenter. This permits the subject's representation of the text to be compared with the representation expressed in the original text. For such an analysis, the reader must examine each sentence in the subject's summary and match the information contained in the sentence to the information contained in the texts that were read. Information in the summary that is highly related to information from the texts would indicate that it was likely learned from the text. Matching this information is not easy. Scanning through the original texts to locate the information. It is not possible to look for exact matches if the subjects do not write exactly the same words as those that they have read. Instead, the reader must make the match on the basis of the semantic content of the text.

There is more research on information extraction from sentences. But our module is integration of information extraction for kids' domains. So we can easily find the relevant searching queries which are suitable for the sentence within the domain in order to retrieve the image.

### 

### 2.2.3 Predicting the similar sentence and finding the most appropriate answer

There is much research that has worked on sentence similarity prediction and transfer learning method in question answering domain. Almost every attempt to predict the sentence similarity most of them used state of the art techniques to identify the sentence. But in this work, we have tried to enhance the supervised learning techniques to predict the sentence similarity. We used state of the art embedding techniques such as Infer Sent to provide semantic sentence representation rather than using word2vec, doc2vec embeddings, we created features such as cosine similarity, Euclidean distance, and root match features combinedly to enhance the performance of supervised model.

Question answering domain has evolved over the time most of the state-of-the-art models are obtained from transformers they adopt the mechanism of attention, weighing the influence of different parts of the input data. Transfer learning, used in machine learning, is the reuse of a pre-trained model on a new problem. State of the art models have significantly benefited from the transfer learning models. We proposed a method that finer tuning of the already available model gives better guidance for learning lexical and syntactical information.

**2.2.3.1 Long range semantic representation sentence similarity prediction**

In this paper, predicting sentence similarity through word embeddings and convolutional neural networks to produce sentence vectors and use that sentence vector pair to estimate the sentence similarity. Yao et al. (2018) [27] Has carried out a research on finding sentence similarity related to Wikipedia corpus fort that he has analyzed long range semantic representation to predict similarity. The result of their experiment shows that we can use the vector pair similarity to estimate the similar sentence.

They proposed an effective model for the similarity metrics of English sentences. In the model, they first make use of word embedding and convolutional neural network (CNN) to produce a sentence vector and then leverage the information of the sentence vector pair to calculate the score of sentence similarity. Considering the case of long-range semantic dependencies between words, they proposed a novel method transforming word embeddings to construct the three-dimensional sentence feature tensor. In addition, they incorporate the k-max pooling into the convolutional neural network to adapt to variable lengths of input sentences. The proposed model requires no external resources such as WordNet and parse tree. Meanwhile, it consumes very lit-tle time for training.

**2.2.3.2 Structural ,Syntactic and Semantic based approach**

Hassan Zadeh et al (2015) [28] has used sentence similarity in the medical domain to find out the disease causes through scholarly publications and systematic reviews on diseases. He used a regression model on a varied set of features to evaluate in general English corpus (SICK corpus). This paper revealed that we can achieve more precision than human judgement in the medical domain.

.

This paper presents an approach for estimat-ing the Semantic Textual Similarity of full English sentences as specified in Shared Task 2 of SemEval-2015. The semantic similarity of sentence pairs is quantified from three per-spectives - structural, syntactical, and seman-tic. The numerical representations of the derived similarity measures are then applied to train a regression ensemble. Although none of these three sets of measures is able to rep-resent the semantic similarity of two sentences individually, their experimental results show that the combination of these features can precisely assess the semantic similarity of the sentences. In the English subtask our system’s best result ranked 35 among 73 system runs with 0.7189 average Pearson correlation over five test sets. This was 0.08 correlation points less than the best submitted run.

**2.2.3.3 Sentence similarity prediction through Lexical Decomposition**

Wang et al (2016) [29] has discussed the sentence similarity finding based on considering similar parts and the dissimilar parts of the sentence by decomposing and composing lexical semantic over sentence. He created similar components and dissimilar components based on the semantic matching vector and trained using two-channel CNN to get better accuracy.

Most classic sentence similarity approaches only consider the sections of two input sentences that are similar, ignoring the parts that are dissimilar, which typically offer us with information and semantic implications about the phrases. In this paper, they provided a model that decomposes and composes lexical semantics over sentences to account for both similarities and differences. In the model, each word is represented as a vector, and the semantic matching vector for each word is calculated based on all words in the other phrase. Each word vector is then deconstructed into a similar component and a dissimilar component using the semantic matching vector. A two-channel CNN model is then used to collect features by combining the similar and dissim-ilar components. Finally, a similarity score is produced for the composite feature vectors. According to testing results, our model achieves state-of-the-art performance on the answer sentence selection test and a comparable result on the paraphrase identification challenge.

**2.2.3.4 Lexical database and corpus statistics based approach**

Pawar et al (2018) [30] used semantic similarity and corpus statistics to find out sentence similarity in different domains. He proposed an edge-based approach using lexical databases to find out semantic similarity between sentences. He obtained Pearson correlation coefficient of 0.8753 for sentence similarity.

In the field of natural language processing, calculating the semantic similarity of sentences has long been an issue. In the field of text analytics research, the field of semantic analysis plays a critical role. As the domain of operation changes, the semantic similarity changes as well. They described a methodology in this study that addresses this issue by integrating semantic similarity and other factors and corpus statistics. The suggested method uses an edge-based approach with a lexical database to calculate semantic similarity between words and phrases.The methodology can be used in a wide range of fields. Both benchmark criteria and a mean human similarity dataset were used to test the methodology. They tested it on two datasets and found that it has the strongest correlation for both word and sentence similarity models. They obtained a Pearson correlation coefficient of 0.8753 for word similarity and a correlation of 0.8794 for sentence similarity.

**2.2.3.5 Multi stage Pre-training for Low-Resource Question Answering systems**

Zhang et al (2020) [33] used transfer learning techniques, which are particularly beneficial in NLP applications that need a large amount of high-quality annotated data. Current methods apply a pre-trained language model (LM) to in-domain text without fine-tuning it for downstream applications. They showed that adding domain-specific terms to the LM's vocabulary leads to increased ains. They used the structure in the unlabeled data to build supplementary synthetic tasks, which aided the LM's transfer to downstream tasks. They sequentially applied these methodologies to a pre-trained Roberta-large LM and demonstrated significant performance gains on three tasks in the IT domain: Duplicate Question Detection, Extractive Reading Comprehension, and Document Ranking.

**2.2.3.6 Evaluation of models for transfer learning based Question Answering systems**

Lee et al (2019) [32] studied cross lingual transfer learning-based question answering on a QA model on a target language task. He concluded that they show significant improvements over several already available baselines on a Chinese QA task.

They created the Biomedical Language Understanding Evaluation (BLUE) standard to help researchers in the biomedicine domain establish pre-training language representations. The benchmark comprises of five tasks with ten datasets that include biological and clinical literature of various volumes and difficulty levels.They also looked at numerous BERT and ELMo baselines and discovered that the BERT model, which was pre-trained on PubMed abstracts and MIMIC-III clinical notes, produced the best results.

## 2.3 Summary

In this chapter, we discussed the literature work that is related to our solution. Next chapter explains about the technologies that we plan to adapt to our solution and how they are related to our solution.

# Chapter 03

# Technologies Adapted

## 3.1 Introduction

Previous chapter is about the related works that have been conducted in specific areas which have been come across and how the proposed approach differs from the existing systems. This chapter describes the technologies that had been adopted in order to implement the proposed solution.

## 3.2 Technology Required

### 3.2.1 Python

Python is a sophisticated, interpreted, high-level general-purpose programming language with substantial libraries for string operations, web service tools, and Natural Language Processing (NLP) packages for document categorization, POS tagging, and sentiment analysis.

### 3.2.2 NLTK

Natural language toolkit is a rich Natural Language Processing platform which facilitates python programs to work with native languages. It is an interface for more than 50 corpora such as WordNet. NLTK supports research areas closely related to natural language processing areas such as linguistics, cognitive science, artificial intelligence, machine learning and feature extraction. Furthermore, classification, tagging, stemming, tagging, tokenization, parsing, semantic, reasoning functionalities. In our approach nltk library is used for keyword extraction, n-gram method, TF-IDF vectorization.

### 3.2.3 Pandas

Pandas is a Python package that makes it easier for programs to manipulate and analyze data. Timely access Data Frames and data manipulation tools are also included. This technique use the Panda library to alter data in keyword extraction modules.

### 3.2.4 NumPy

NumPy is another library for the Python programming language that supports handling large, multidimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on them. NumPy is used for easy manipulation of multi-dimensional vectors.

### 3.2.5 Bi-LSTM

bidirectional Long Short-Term Memory (LSTM) is working fine, and they are doing sequential processing of text. This technology will be used in most of the sequential text-based models.

### 3.2.6 BERT

It is a language model based on transformer architecture. It helps to train better accurate NLP models. It is trained on huge amounts of text data with easily fine-tuned structure. It is a universal computation engine for multiple deep learning tasks including NLP and computer vision.

### 3.2.7 Scikit-learn

Scikit-learn is a Python-based machine learning library which is freely available. It covers support vector machines as well as many other classification, regression, and clustering algorithms.

### 3.2.8 Flask

Flask is an accessible web framework. This means flask offers you all tools, modules, and technologies you have to develop a web app.

### 3.2.9 Torch

PyTorch is an open source machine learning library based on the Torch library that can be used for tasks including object recognition interpretation.

### 3.2.10 Transformers

In over 100 languages, Transformers provides lots of pretrained models for tasks such as classification, information retrieval, language understanding, summarization, interpretation, text generation, and more.

### 3.2.11 Infersent

InferSent is a semantic sentence representation method based on sentence embedding. It's derived from natural language inference data and also can perform a wide range of tasks.

### 3.2.12 Spacy

spaCy is a Python and Cython library for advanced Natural Language Processing. It's built on cutting-edge research and was developed from the bottom up to be applied in real goods. spaCy includes systems statistical models and word vectors, and it already supports tokenization for further over 60 languages.

### 3.2.13 PyCharm

PyCharm by JetBrains is an integrated development environment used in computer programming, specifically for the Python language.

### 3.2.14 Keras

Keras is an open-source high-level API for machine learning frameworks like tensor-flow, Theano etc. In this project keras will be used with a tensor-flow backend.

### 3.2.15 Google Colab

As the processing power of the computers of team members are not enough to do our tasks. We use Google Colab to train our model.

### 3.2.16 Selenium

Selenium is an explanatory potential for application online testing. Without having to learn a test programming language, this became a playback tool for developing functional tests.

### 3.2.17 PIL

Python Imaging Library is a free and open-source extended library for Python that adds support for importing, editing, and saving a broad range of file formats.

### 3.2.18 MoviePy

To do the basic operations like cuts, concatenations, title insertions we can use the Python module MoviePy. It can be used for video composing, creating advanced effects, video processing, image analysis, and complex effects. It supports the most popular video codecs, like GIF.

## 3.3 Summary

This chapter described the technologies that have been adopted and applied in the development of the solution. Also, it was mentioned why those technologies were used. Moving to the next chapter, it will discuss how those adopted technologies are going to be applied in the proposed solution.

# Chapter 04

# Our Approach

## 4.1 Introduction

Previous chapter was about what technologies were used in developing the system and their usage. Hereafter, in this chapter it is to be discussed how those suggested technologies are going to be applied in the proposed solution development.

## 4.2 Our Approach

This approach is divided into 3 major modules, as described under the following subheadings. It has been decided in the initial planning stages of the system to carry out the research in this manner to capture the full scope of the research problem and to provide a better accurate result on the problem in hand.

### 4.2.1 Content modification using summarization with lexical simplification

Aim of this module is to modify the content which is converted from a given text content for video implementation. Content modification includes Summarization of original text with Lexical Simplification. Our main domain is kids. Kids find it difficult to understand complex words. So we are going to convert the complex words into the simplest form with summarization. Because extracting the important information from original content is essential for video making. Here lexical simplification is done by predicting the complex words (hard vocabulary) and substituting it with simple words. So the output of this module is a simplified summary.

There are two methods for summarization. They are abstraction and extraction methods. I had chosen the abstract summarization method because it produces a summary in a human thinking way. To train the model, I have got news summary datasets for text summarization and complex word identification datasets for lexical simplification. News articles contain good vocabulary and grammar which gives greater insights. The Complex Word Identification dataset contains additional information about the language level of the annotators for non-native English speakers. The data format is the same as the CWI shared task dataset plus 4 additional columns. The four columns show the number of advanced, intermediate, beginner, and (not-provided) number of annotators for the non-native speakers. The not-provided column shows where the worker does not provide their English language during the experiments.

### 4.2.2 Information extraction from sentences

Aim of this module is to generate a video which can be understood by the children for the given audio. We can make a video for the given audio with the suitable images which we get for the particular sentences or word from google. Additionally we are going to categorize the video age wise according to the children’s capability. This will help to enhance their knowledge, skills, and therefore enhance the standard of education. We can get the video age limit using the summary sentences age limit. According to the majority age limit we can get the applicable age. Above 14 years video will be mentioned and adult contents, violence contents will be ignored from the video.

A basic task in sentiment analysis is classifying the polarity, motion Detection, Intent Detection; additionally we are going to categorize the sentences age wise according to the children’s capability. Text Analysis is about parsing texts in order to extract machine-readable facts from them; Semantic analysis is the process of relating syntactic structures, from the levels of phrases, clauses, sentences and paragraphs to the level of the writing as a whole, to their language-independent meanings. Through these techniques we are going to find the relevant searching queries which are suitable for the sentence within the domain. In order to retrieve the image, we are going to use google image searcher.

### 4.2.3 Predicting the similar sentence and finding the most appropriate answer

Aim of this module is to identify the sentence which has the answer for the question and identify the most appropriate answer. We collected our data from SQUAD dataset, and we manually annotated data using the tool ‘QA data annotator’. For predicting the sentence, first we break the paragraph into multiple sentences (text blob library used here). Then, we are using sentence embedding techniques to represent every sentence and question in semantic form. We created vocabulary from the training data and then we trained an infersent model using those vocabulary. Then, we got the vector representation of each sentence and question using the infersent model. Based on this vector representation of the sentence, we derived the features based on Euclidean distance, cosine similarity by comparing sentence and question vector values. To improve model performance, we derived another feature based on root match index which is based on comparing the root of the question and answer (to derive sentence parse tree and compare roots we used spacy library). Then using supervised machine learning techniques to predict the sentence which has the answer. Here we defined our target variable as a sentence index.

The second part of my module is to find the appropriate answer for the question. To do this task, we used the manually annotated question answering dataset. Here we used the approach called taking a custom dataset and using it to fine tune a pretrained model to improve the performance. We used a distilbert-base-uncased pretrained model for this task. We tokenized our dataset using hugging face Tokenizer to encode the question and context which used to create each corresponding paragraph and question to train Bert. Once we trained the model, we compared the performance between the base model and the model which pre pre-trained by custom dataset.

## 4.3 Summary

In this chapter we explained our approach in brief and ways of solving the problems that we are addressing and use of technology that we plan to use for our project. The next chapter will explain in detail designs of our system.

# 

# 

# Chapter 05

# Analysis and Design

## 5.1 Introduction

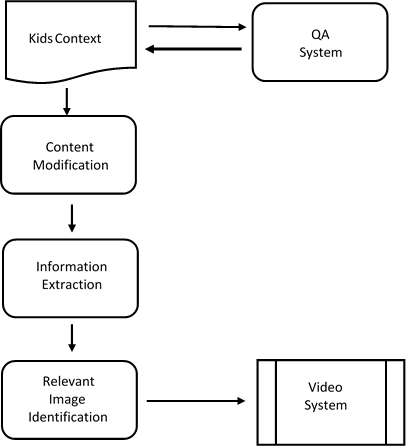
In this chapter, it is expected to mention about the System design and the analysis of our solution. Top level architecture of the system, how modules are interconnected, and the system design are included and briefly described.

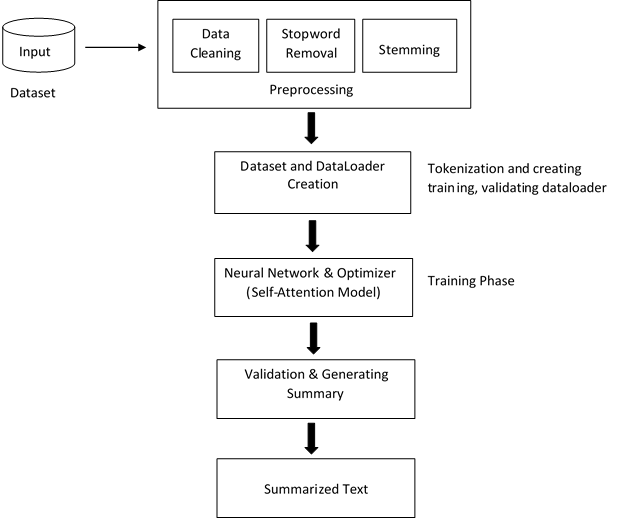


Figure 1:5.1 High level diagram for approach

As the initial design of the system, it is agreed that the final outcome of the system should take both content/ languages driven and network driven approaches. First module involves Content Modification using summarization with lexical simplification, the second module involves Information extraction from sentences. Last module involves Question answering and Information retrieval System.

## 5.2 Content modification using summarization with lexical simplification

This module is responsible for modifying the text content by summarizing and identifying the complex words from summarized content and substituting it with simple understandable words. 

Figure 2: 5.2 High level diagram for Summarization Process

As the above shown figure summarization is done by transfer learning using neural network and optimizer with the self-attention model. These are the main steps:

1. Preparing Environment and Importing Libraries
2. Preparing the Dataset for data processing
3. Defining function for training the model: Function
4. Validating the Model Performance: Function
5. Main Function
   1. Importing and Pre-Processing the domain data
   2. Creation of Dataset and Dataloade[r](https://www.kaggle.com/eggwhites2705/transformers-summarization-t5#section503)
   3. Neural Network and Optimizer
   4. Training Model
   5. Validation and generation of Summary

Then Lexical Simplification is done to substitute the complex word with simple words. The diagram of the work is as follows:

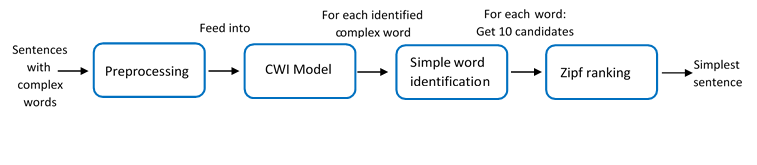


Figure 3: 5.3 High Level Diagram for Lexical Simplification

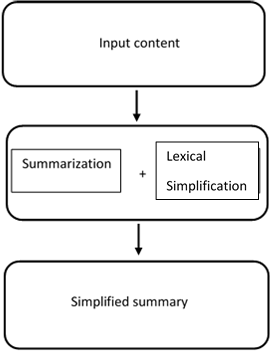
This task will be address in 3 steps:

**1. Identify complex words in a given sentence:** Create a model that can detect or identify possible complex words, it is called complex word identification (CWI).

**2. Generate candidates:** Used BERT´s masked language model to get possible words candidates.

**3. Select the best candidates based on Zipf values:** Compute the zipf values of each candidate to select the simplest one.

Main objective of this module is to modify the transcript content. So this module builds a model by integrating summarization and lexical simplification. Here lexical simplification is done for kids’ domain. So complex summarized content will be formed into the simplest format which can be easily understood by kids.



### 

### 

### 

### 

### 

### 

### 

### 

### 

## 

## Figure 4: 5.4 High level diagram of content modification using

## summarization with lexical simplification

## 5.3 Information extraction from sentences

This part is to extract the information from the relevant sentence or group of Sentences and make a video using Google images. To do this task we are going to use,

1. Sentimental analysis, Age categorizes
2. Text analysis, Semantic analysis

On the summarized sentences. We get the summarized sentences from the Text Summarization module and analyze them. So, we can make a video for the given audio with the suitable images which we get for the particular sentences or word from Google. A basic task in sentiment analysis is classifying the polarity, motion Detection, Intent Detection; additionally we are going to categorize the sentences age wise according to the children’s capability. Here we are going to limit the age to five main contents,

1. Grade 1
2. Grade 2
3. Grade 3
4. Grade 4
5. Grade 5

Above 14 years video will be mentioned and adult contents, violence contents will be

ignored from the video. So, we can categorize the video according to these age limits. This will help to enhance their knowledge, skills, and therefore enhance the standard of

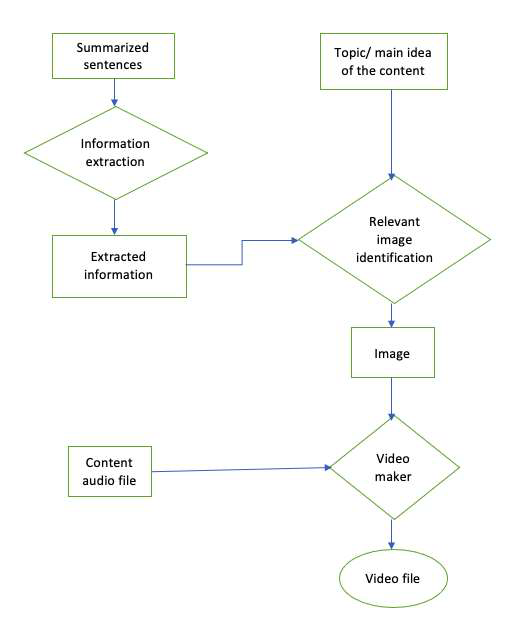
education.

Figure 5: 5.5 High level diagram of Information extraction.

## 5.4 Predicting the similar sentence and finding the most appropriate answer

**5.4.1 Predicting the similar sentence**

In Predicting the sentence and finding the most appropriate answer through text analysis there are two components. First component is used to predict the sentence which has the answer, and the next component is predicting the most appropriate answer for the question. For sentence similarity finding there are many novel approaches existing in the domain using transformers and neural network-based language pretraining. But there are not many more approaches or research studies based on supervised learning techniques to identify sentence similarity. To carry out my research We try to model relationships and dependencies between sentences to predict the similar sentence using supervised learning techniques. For that we have used Squad dataset and manually annotated dataset to carry out the research. When language is converted into machine readable form the standard approach, we use is to represent in vector form. To carry out language into vector form we used sentence embedding method which is based on Infersent. First, we break the paragraph into sentences and clean and remove punctuation in the sentence. Then we used the Infersent model to retrieve the vector representation of each sentence likewise we represented the question in vector form. Using the vector values of the sentence question pair, we created features based on cosine similarity and Euclidean distance for each sentence question pair in the dataset. We created another feature based on the dependency parse tree. In here, we have used the spacy library to derive dependency parsing trees. If we find the sentence root and question roots are the same there is a much more likely chance that sentence will answer for that question. Considering that feature we created another variable which indicates both roots are equal or not, whose value is considered as either ‘1’ or ‘0’. Here we have assigned our target variables as sentence index. Based on the features we obtained from the dataset we have used to train with multinomial logistic regression, random forest, and gradient boosting technique to compare performance.

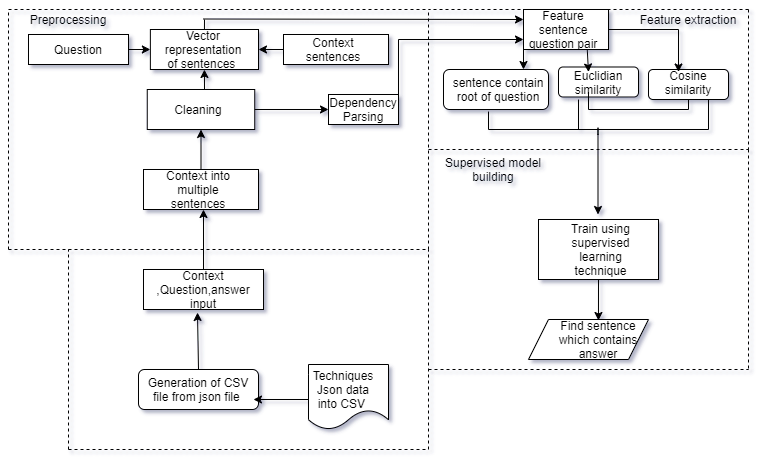
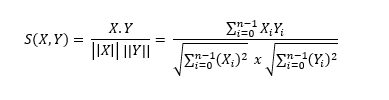


Figure 6: 5.6 System design for finding sentence similarity

**5.4.1.1 Vector Representation of Sentences**

**5.4.1.2 Cosine similarity**

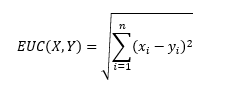
The cosine similarity calculates the cosine of the angle between two vectors. In order to calculate the cosine similarity we use the following formula:

Figure 7: 5.7 Equation of cosine similarity

Accordingly, the cosine similarity can take on values between -1 and +1. If the vectors point in the exact same direction, the cosine similarity is +1. If the vectors point in opposite directions, the cosine similarity is -1.The cosine similarity is very popular in text analysis. It is used to determine how similar documents are to one another irrespective of their size. The TF-IDF text analysis technique helps convert the documents into vectors where each value in the vector corresponds to the TF-IDF score of a word in the document. Each word has its own axis, the cosine similarity then determines how similar the documents are.

**5.4.1.3 Euclidean distance**

The Euclidean distance is a straight-line distance between two vectors.For the two vectors x and y, this can be computed as follows:

Figure 8: 5.8 Equation of euclidean distance

Euclidean distance multiplies the effect of redundant information in the dataset. If I had five variables which are heavily correlated and we take all five variables as input, then we would weight this redundancy effect by five.Euclidean similarity can take on values between 0 to 60. 0 means both vector values are more similar and 60 means vector values are far away from each other.

**5.4.1.3 Sentence structure parsing**

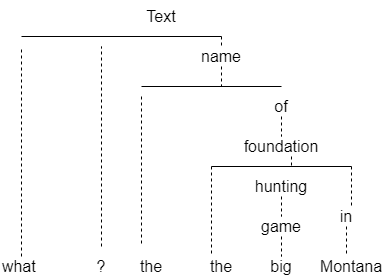
Dependency derives relations among the words in a sentence with directed , labeled arcs from heads to dependents. Labels are drawn from a fixed inventory of grammatical relations. It also consists of a root node that explicitly marks the root of the tree, the head of the entire structure.

**Obtaining root using spacy parse tree**

import spacy

en\_nlp = spacy.load('en\_core\_web\_sm')

question\_root = st.stem(str([sent.root for sent in en\_nlp(question).sents][0]))

Figure 9: 5.9 Dependency parse tree obtained from spacy library

**5.4.2 finding the most appropriate answer**

The next component consists of finding the appropriate answer for the question. Suggesting an appropriate answer is an important factor in deciding children's learning process. There are many systems in this domain to predict the answer for the question. But there is no system which was created for kids’ domains. to do this task. This module is done by transfer learning using a neural network and optimizer self-attention model. I created a custom dataset using the ‘QA annotator’ tool. Which consist of kid’s understandable questions and answers. The QA annotator tool will output the data in json format. Then we preprocess the dataset which used to simplify our process to change into Bert accepted format. Which includes converting sentences into tokens and finding out the answer start and answer end character positions. After preprocessing the data, we train our model and evaluate against the existing base model.

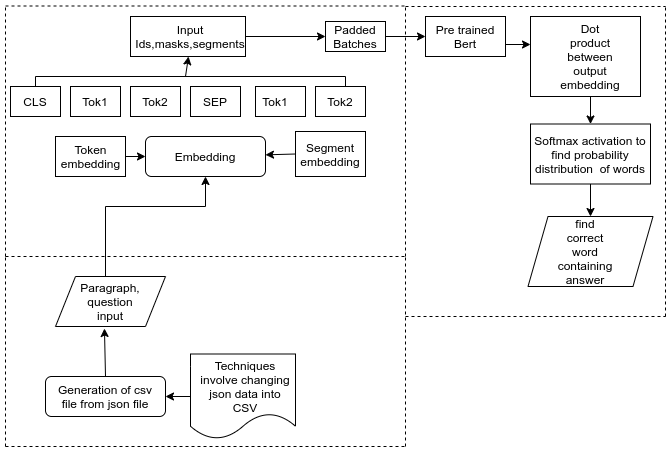


Figure 10: 5.10 System design for finding appropriate answer

**5.4.2.1 Collecting dataset**

Dataset related to children domain obtained from manually annotating data from children story, general knowledge story books. Data was annotated using the QA annotator tool.

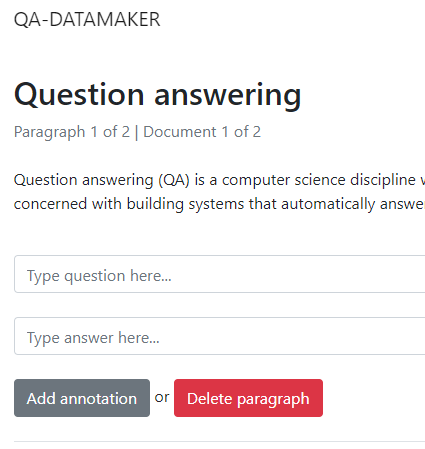


Figure 11: 5.11 QA Annotator Tool UI

**5.4.2.1 Bert Model Structure**

In the Question Answering task, BERT takes the input question and passage as a single packed sequence. The input embeddings are the sum of the token embeddings and the segment embeddings.To fine-tune BERT for a Question-Answering system, it introduces a start vector and an end vector. The probability of each word being the start-word is calculated by taking a dot product between the final embedding of the word and the start vector, followed by a softmax over all the words. The word with the highest probability value is considered.

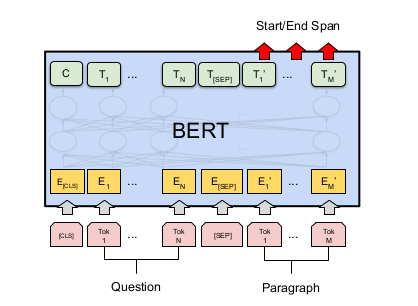
****

Figure 12: 5.12 Bert Model Structure for Question Answering

## 5.5 Summary

In this chapter we explained in detail the design of our system. We have divided our system to have three parts such as Content modification using summarization with lexical simplification, Information extraction from sentences, Predicting the similar sentence and finding the most appropriate answer. Certain constraints and requirements needed at some stages of the system were mentioned in this chapter. Next chapter we are going to discuss how we implement the model.

# Chapter 06

# Implementation

## 6.1 Introduction

In the previous chapter we discussed our system's complete design. In this chapter we discuss implementation details of each module that we have designed.

## 6.2 Our Introduction

We have divided our part into three modules such as, Content Modification using summarization with lexical simplification, Information extraction from sentences and Question answering and Information retrieval System. We plan to implement these modules individually and integrate them as one system. First, we individually collect our dataset as usage of each module. Also, we collected dataset which can be used for each module as well.

### 6.2.1 Content Modification using summarization with lexical simplification

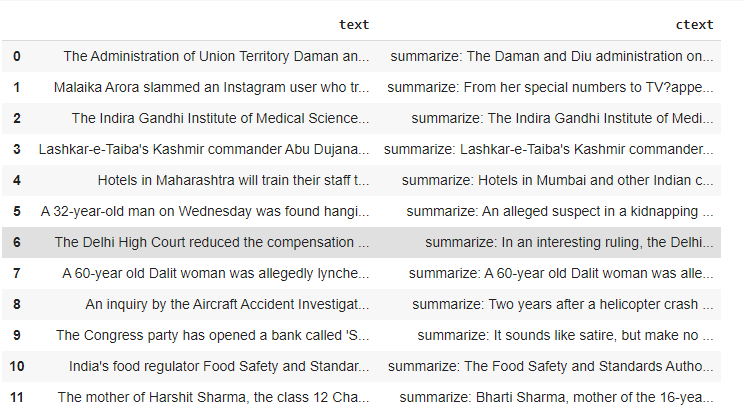
This module is responsible for modifying the text content by summarizing and identifying the complex words from summarized content and substituting it with simple understandable words.

Unique approach of this module is that we have modified the original content by summarization and lexical simplification. It can be more understandable by kids. Therefore, original content will be transformed to a more understandable simplest form for kid’s specific domains.

There are mainly two summarization approaches used for summarization. Here we have used Abstractive Summarization. Because it includes heuristic approaches to train the system in making an attempt to understand the whole context and generate a summary based on that understanding. This is a more human-like way of generating summaries and these summaries are more effective as compared to the extractive approaches.

**6.2.1.1 Collecting dataset for summarization and lexical simplification**

For the summarization task I have used the News Summary dataset. News articles contain good vocabulary and grammar which gives greater insights.

Figure - News Summary Dataset

For the lexical simplification task I have used complex word identification datasets. Complex word Identification (CWI) is a sub-task of lexical simplification (LS), which identifies difficult words or phrases in a text.

The English datasets consists of three genres:

* Professionally written news
* News written by amateurs (WikiNews)
* Wikipedia articles

These datasets contain information about complex phrases annotated with some statistics. Each line represents a sentence with one complex phrase (CP) annotation and relevant information, each separated by a TAB character.

* First column shows the HIT ID of the sentence. All sentences with the same ID belong to the same HIT.
* Second column shows the actual sentence where there exists a complex phrase annotation.
* The third and fourth columns display the start and end offsets of the complex phrase annotation in this sentence.
* The fifth column represents the actual complex phrase annotation.
* The sixth, seventh, and eighth columns show the number of native annotators, the number of non-native annotators and the total number of annotators who have marked this complex phrase.

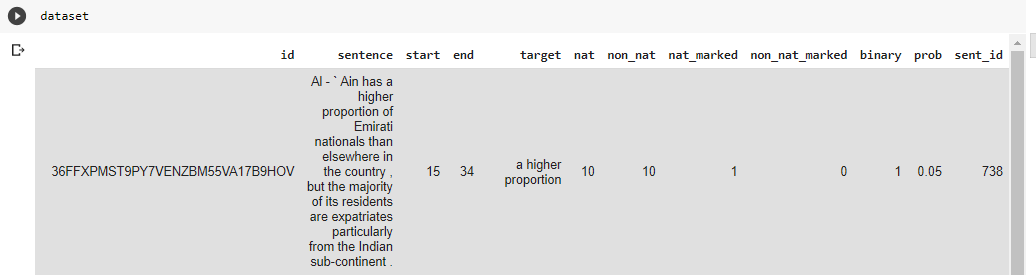


Figure 13: 6.1 CWI dataset

Here, we can see that the phrase "flexed" is marked as a complex phrase by 2 native and 7 non-native English speakers whereas the phrase "flexed their muscles" is marked by 4 native and 2 nonnative English speakers.

**6.2.1.2 Text Preprocessing for summarization**

The methodology I have chosen to implement for this module is Transfer Learning. It can be approached by using a model which is trained for a particular task to train our dataset and fine tune the base model for our dataset problem. In transfer learning, we first train a base network on a base dataset and task, and then we repurpose the learned features, or transfer them, to a second target network to be trained on a target dataset and task. Text preprocessing following steps.

**6.2.1.2.1 Tokenization**

We will start with creation of a Dataset class - This defines how the text is pre-processed. This class is defined to accept the Data frame as input and generate tokenized output that is used by the transfer learning model for training. We have used T5 tokenizer to tokenize the data into original text and summarized text columns of the data frame.

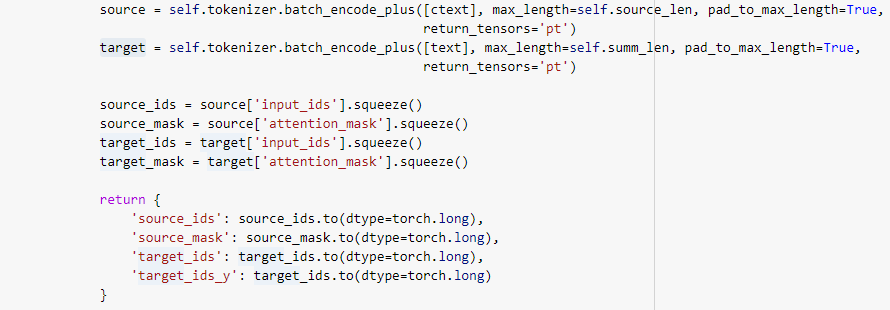


Figure: Tokenization

**6.2.1.2.2 Dataset creation**

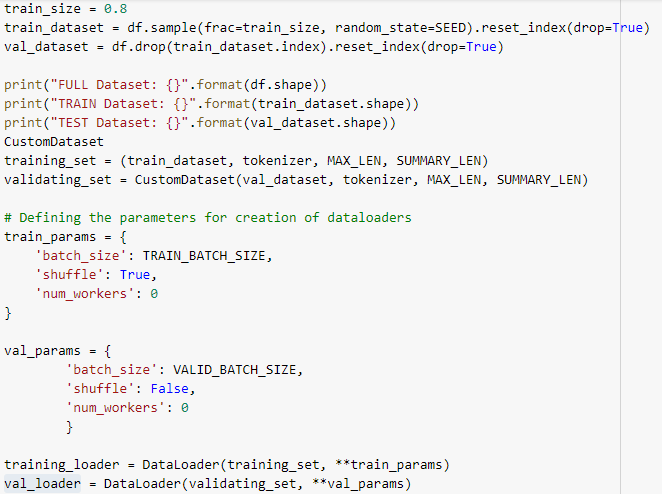
The Custom Dataset class is used to create 2 datasets, for training and for validation. Training Dataset is used to fine tune the model: 80% of the original data**.** Validation Dataset is used to evaluate the performance of the model. The model has not seen this data during training. Train and validation parameters are defined and passed to the pytorch dataloader construct to create train and validation dataloader. These dataloader will be passed to train() and validation function respectively for training and validation.

Figure: Training and validation dataloader creation

**6.2.1.3 Training**

In the training phase we have used a self-attention model in transfer learning. After the training is completed, the validation step is initiated. We use the fine-tuned model to generate new summaries based on the article text. The original summary and generated summary are converted into a list and returned to the main function.

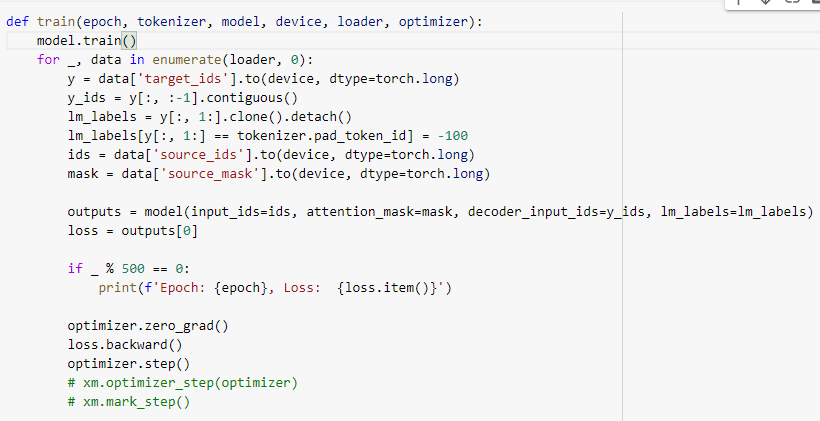


Figure: Training

**6.2.1.4 Text preprocessing for lexical simplification**

Our research is implemented to enhance the learning ability for kids. So, kids should easily understand the content. So, we have simplified the content by Lexical Simplification. Lexical Simplification (LS) is identifying complex words and replacing them with simple words, which can help children to better understand a given text.

data preprocessing has to be done to the dataset before it feeds to the deep learning model. It includes:

* Clean the data
* Create Vocabulary
* Get the embedding vectors



Figure: Load the embedding model

**6.2.1.4 Complex word identification model**

The first goal is to identify complex words, for that we need to train a model able to detect complex words in sentences. To train this model we are going to use the labeled dataset and we are going to use a sequential architecture based in Bi-LSTM, which provides contextual information from both the left and right context of a target word.

Dataset has sentences with binary labeled words (if the word is considered complex), we will use this to train the CWI model. As usual in NLP, we need to apply some preprocessing to text in order to feed the data into a deep learning model:

1. clean the text: delete not alphanumeric characters, lower case all words, etc.
2. create the vocabulary.
3. Get the embedding vectors (in this project we are using glove).

After preprocessing we will be creating a model for complex word identification (CWI) using keras. This model is going to be used only for identifying possible complex words in a given sentence. After training the CWI model, we will use BERT to generate candidates over the words which the CWI model identified as complex, the candidates are based not only in the synonym of the word but in context of them.

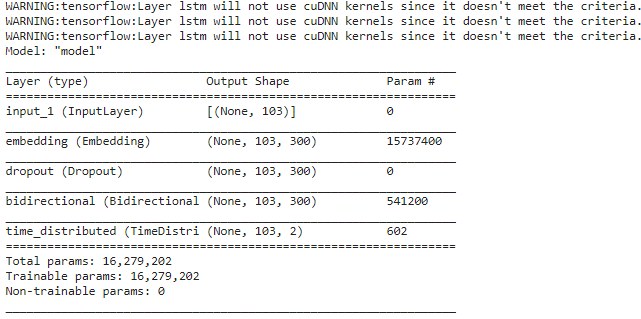


Figure - CWI model

**6.2.1.5 Generate candidates using BERT**

We have used one of BERT's task masked language modeling (MLM), which identifies missing tokens in a sequence given its left and right context. One of the tasks BERT was trained for, is to be able to predict the [MASK] word, so if we input into a BERT model, it will output the most probable word given the context. We are masking the complex words of each sentence and get the probability distribution of the vocabulary corresponding to the masked word.

We’ll concatenate the original sequence and the sequence where we replace the complex word with mask token as a sentence pair, and feed the sentence pair into the Bert to obtain the probability distribution of the vocabulary corresponding to the mask word. By using the sentence pair approach, we not only consider the complex word itself, but also fit the context of the complex word.

**6.2.1.6 Get the Zipf value to select the suitable word from generated candidates**

Final step of this task is to select the best candidates based on Zipf values. We adopt the Zipf frequency of a word which is the base-10 logarithm of the number of times it appears per billion words to rank the replacement word candidates. The greater the value the more common or familiar the word is for a person. We compute the zipf score using the python’s package wordfreq.

Above sections we have seen how these summarization and lexical simplification were built. Our aim is to combine both summarization and lexical simplification in order to modify the original content for children's understandable content. So in the first phase original content has been summarized. Then the summarized content will feed to the lexical simplifier. Lexical simplifier will output the simplified summary. In this module our main input is a converted transcript text and output will be a summarized simplified content.

### 6.2.2 Information extraction from sentences

My part of the project is to extract the information from the relevant sentence or group of sentences and make a video using Google images. To do this task I am going to use,

* Sentimental analysis, Age categorizes
* Text analysis, Semantic analysis

I get the summarized sentences from the Content modification module and analyse them. So, I can make a video for the given audio with the suitable images which I get for the particular sentences or word from Google. A basic task in sentiment analysis is classifying the polarity, motion Detection, Intent Detection; For sentiment analysis I am going to analysis a sentence according to,

1. Positive
2. Neutral
3. Negative

Sentiments and additionally I am going to categorize the sentences age wise according to the children’s capability.

##### 6.2.2.1 Collecting Dataset

The first step is the collecting of the dataset. The dataset that we used was collected from Srilanka teachers handbook and students book for the selected age categories.We have divided the training datasets and testing datasets into 80:20 ratios.

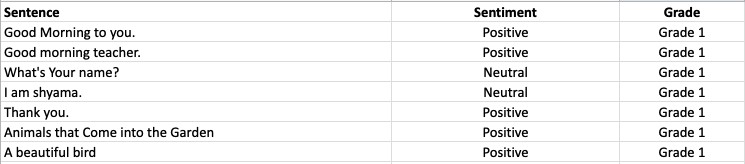


Figure 14: 6.2 Sample data of children sentences dataset

##### 6.2.2.2 Features

Using this age categorizing we can tell that the system made video is suitable for which age group. Most of the schools divide student’s knowledge capability according to,

1. Grade 1, 2
2. Grade 3, 4
3. Grade 5
4. Grade 6, 7, 8
5. Grade 9, 10 ,11
6. Above grade 11

So, we can categorize the video according to these age limits. This will help to enhance their knowledge, skills, and therefore enhance the standard of education. We can get the video age limit using the summary sentences age limit. According to the majority age limit we can get the applicable age. Here I am going to limit the age to five main contents,

1. Grade 1
2. Grade 2
3. Grade 3
4. Grade 4
5. Grade 5

Above 14 years video will be mentioned, and adult contents and violence contents will be ignored from the video.



Figure 15: 6.3 Labeling the children sentences dataset

After that I will do semantic analysis. Semantic analysis is the process of relating syntactic structures, from the levels of phrases, clauses, sentences and paragraphs to the level of the writing as a whole, to their language independent meanings. Through these techniques I am going to find the relevant searching queries which are suitable for the sentence within the domain. In order to retrieve the image, I am going to use google image searcher.

##### 6.2.2.3 Tokenizing

We need to tokenize our texts. BERT was trained using the Word Piece tokenization. It means that a word can be divided into more than one sub-words. When dealing with out of vocabulary words, this kind of tokenization is beneficial. It helps better represent complicated words. During the training time the sub-words are constructed and depend on the corpus the model was trained on. We’ll get the best results if we tokenize with the same tokenizer the BERT model was trained on, even though we could use any other tokenization technique. The PyTorch-Pretrained-BERT library provides us with a tokenizer for each of the BERTS models. Here I use the basic bert-baseuncased model.

##### 6.2.2.4 Training

I used several layers to increase the model accuracy. An embedding is a mapping of a discrete categorical variable to a vector of continuous numbers. Embeddings are low dimensional, learned continuous vector representations of discrete variables in the context of neural networks. Neural network embeddings are useful because they can reduce the dimensionality of categorical variables and meaningfully represent categories in the transformed space.

Bidirectional LSTMs are an extension of general LSTMs that will improve performance on sequence classification problems. Bidirectional LSTMs train two LSTMs on the input sequence instead of one. First it trind the input sequence as it is and the second on a reversed copy of the input sequence. This will provide additional context to the network and even fuller learning on the problem. The result is faster. Dense layers learn a weight matrix, where the first dimension of the matrix is the dimension of the input data, and the second dimension is the dimension of the output data. Recall that your Input layer has a shape of 1. In this case, your output layer will also have a shape of 1. This means that the dense layer will learn a 1x1 weight matrix.

##### 6.2.2.5 Video making

After this training I will get the sentences and the sentiment, semantic of the sentence and age group for the sentence when I give a summarized text as an input. So I can search for google images using those tags and make a video.



Figure 16: 6.4 Sample output of sentiment, semantic of the sentence and age group

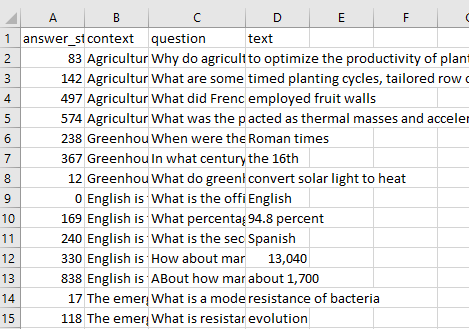
Using selenium I can scrap for google images. Selenium is a portable framework for testing web applications. Selenium provides a playback tool for authoring functional tests without the need to learn a test scripting language. Python Imaging Library is a free and open-source additional library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats. To do the basic operations like cuts, concatenations, title insertions we can use the Python module MoviePy. It can be used for video composing, creating advanced effects, and video processing. It can read and write the most common video formats. I will add the images according to the audio file time frame. So the children can easily learn information about the particular sentence with the images while watching the video.

### 6.2.3 Predicting the similar sentence and finding the most appropriate answer

This module is responsible for predicting the sentence which has the answer for the question and finding the most appropriate answer for the question. In the first component we are using supervised learning techniques to enhance the performance rather than using state of the art techniques which are involved in neural networks. The second component is based on transfer learning technique, and it was discussing how we train the existing model in the custom data to enhance the performance.

##### 6.2.3.1 Collecting Dataset

The initial step is the preparation of the dataset. The dataset that we used for our project was collected from SQUAD question answering dataset from Stanford university and we manually annotated data through ‘QA Data annotator’ which is available as an open-source project to annotate data for different domains. In this research we collected around 1500 question answer pairs from SQUAD dataset and 500 manually annotated data for kid’s education domain which mostly consist of Wikipedia articles about places, persons etc. In the data, it consists of some features like paragraph (context), question, starting position of the answer index (answer\_start) and answer for that question. So, in that data we are using 80% of data for training and 20% for testing purposes. All data is saved in a csv file after preprocessing json data into a csv file.

Figure 17:6.5 Dataset finding similar sentence

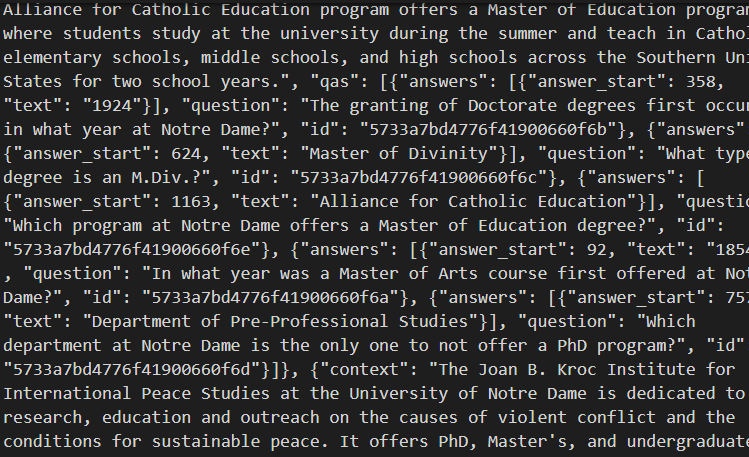


Figure 18: 6.6 Annotated dataset finding most appropriate answer

**6.2.3.2 Predicting the Similar Sentence**

**6.2.3.2.1 Pre-processing**

In the first component, our goal is to find the sentence which has the correct answer. First, we break the paragraph into sentences using the text blob library. After that we clean the data by removing punctuations in the sentences.

To compare sentences, we must transform sentences into numerical form. So, we are using sentence embedding technique which was based on Infersent. We create vocabulary from the training data, and we use that vocabulary to train the infersent model. When we input a sentence into the infersent model it will output the vector representation of the sentence without considering the number of words. This model performed well on many natural language tasks like summarization, sentence similarity finding etc.

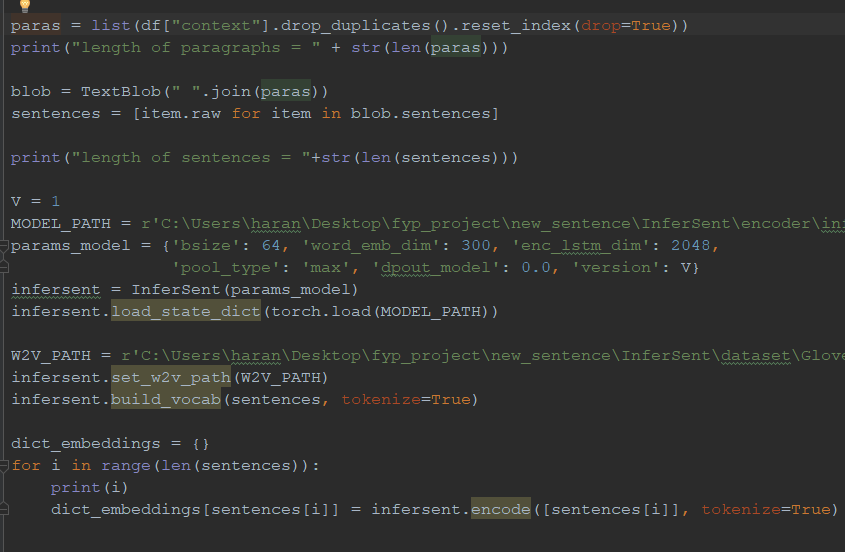


Figure 19: 6.7 Preprocessing and obtain vector representation of sentences

**6.2.3.2.2 Create Features**

After creating vector representation of each sentence and question, we created features like cosine similarity and Euclidean distance from the obtained vector values, which involved comparing two vector values of sentence and question pair. We further created another feature based on the dependency parse tree. In here we used the spacy library to create a dependency parse tree and used the parse tree to compare roots of the sentence question pair. We stemmed down sentence and question roots using the spacy library. If the root of the question and sentence are equal after stemming to its original form, there is a good chance that the sentence contains the answer for that question. Considering this feature, I have created another feature which contains values ‘0’ or ‘1’.’1’ indicates that the roots of the sentence and question are equal and ‘0’ represent not equal. If the sentence and question have multiple roots, we compare every root and set the value either ‘1’ or ‘0’. Here we defined our target variable as a sentence index.

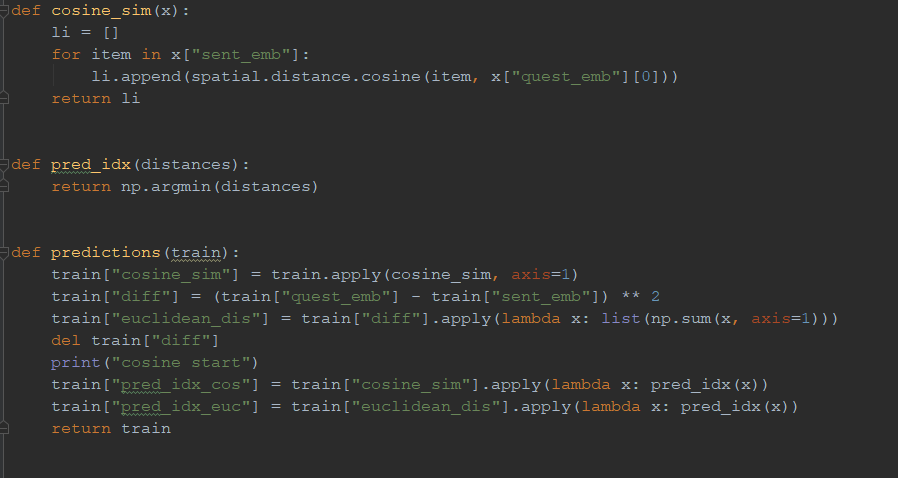


Figure 20: 6.8 Create features cosine and euclidean

##### Figure 21: 6.9 Dataset after creating feature based on euclidean and cosine

##### 6.2.3.2.3 Missing Value

It is important to fill missing values and standardize all the columns. We consider all the paragraphs to have at least 10 sentences. So, if a paragraph does not have 10 sentences, we replace cosine similarity value as ‘1’ and Euclidean distance value as ‘60’, which indicate that it has maximum deviation from the question vector value.

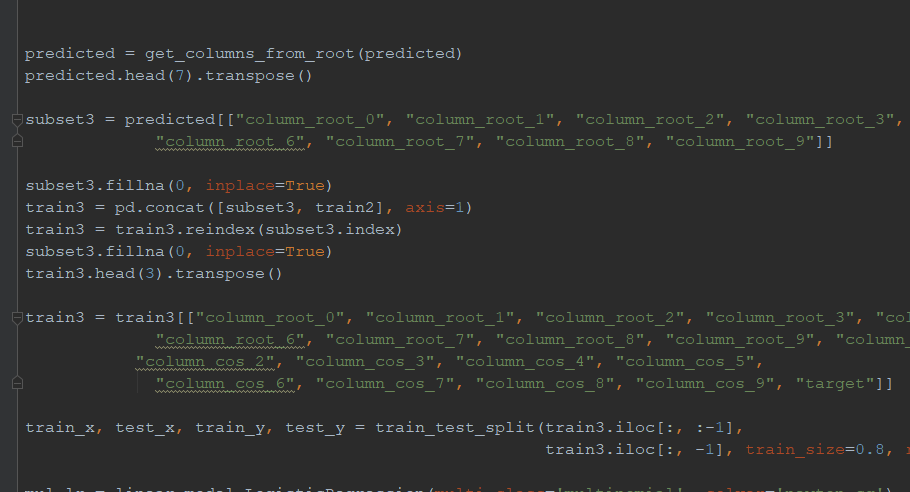
**

Figure 22: 6.10 Handling missing values in dataset

##### 6.2.3.2.3 Training

There are many ways to do classification problems. In this research we involved a supervised learning approach to get best performance rather than considering state of the art approaches. We created features based on cosine similarity, Euclidean distance, root match and we defined our target variable as sentence index. We have used different supervised machine learning techniques like Logistic Regression, Gradient Boosting and Random Forest.

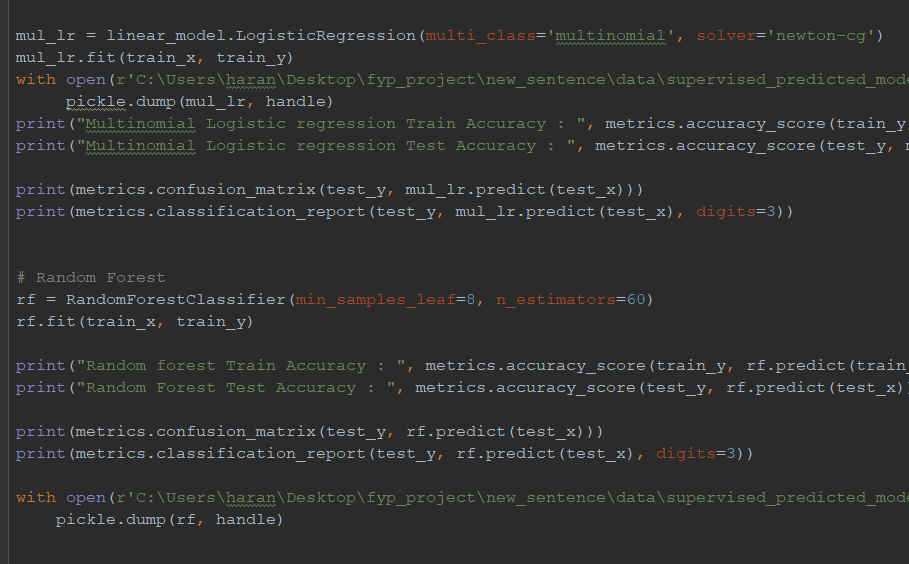


Figure 23: 6.11 Training dataset using supervised algorithms

**6.2.3.3 Finding the most Appropriate Answer**

**6.2.3.3.1 Pre-processing**

We prepared the dataset in previous steps. It consists of paragraph (Context), question and answer start index regard to character position. We are using one of BERT’s task masked language modeling (MLM), which predicts missing tokens in a sequence given its left and right context. One of the tasks BERT was trained for, is to be able to predict the [MASK] word, so if we input into a BERT model, it will output the most probable word given the context. We are masking the answer to each question. To mask the answer in the paragraph we must add the end index of the answer too. To identify the end index of the answer we created a function add end index to identify the position. After finding the end index of our answer in the paragraph we must convert the paragraph and question into tokens. We are using the hugging face tokenizer library for this purpose. In this process our paragraph and context change into encoding objects. We add another two encoding objects: start position and end position with respect to question and context.

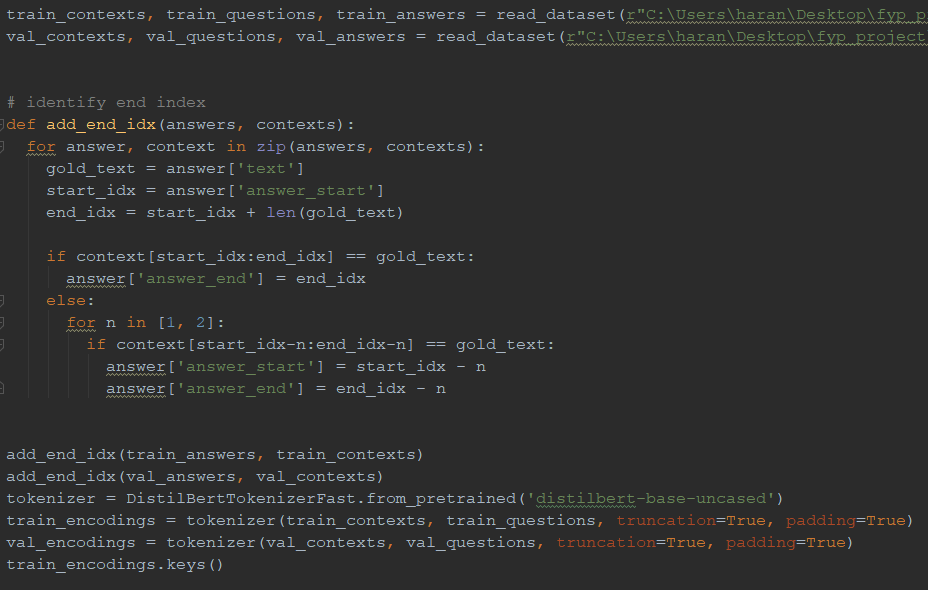
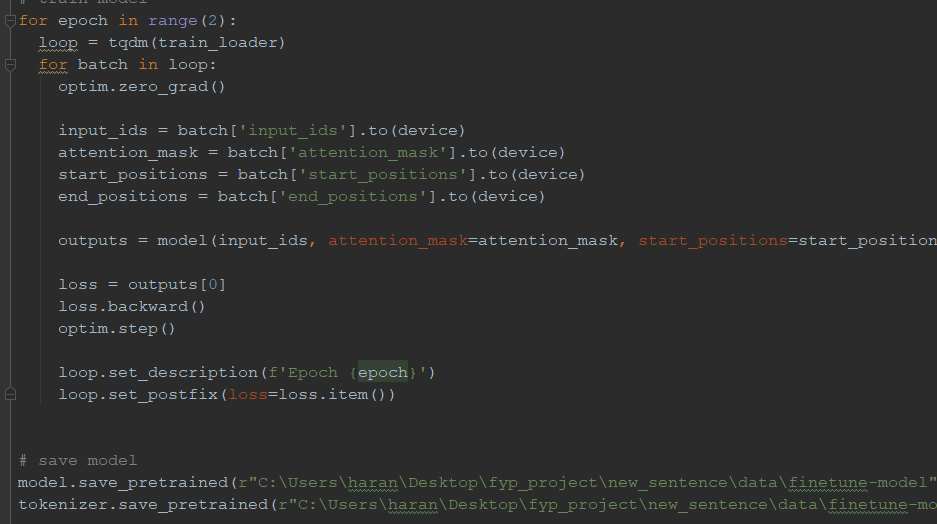


Figure 24: 6.12 Preprocessing data for finding appropriate answer

**6.2.3.3.2 FineTuning**

We used the PyTorch library to implement our finetuned model. We used a distilbert base model to fine tune into our custom dataset. After training our model. Our model predicts the start and end index of our answer positions. To estimate our model performance, we use an exact match matrix which is based on how many start and end tokens are predicted correctly.

Figure 25: 6.13 Pretrain the model using custom dataset

## 

## 

## 6.2 Summary

This chapter discusses what are the things that we have implemented and what are the future work that we are going to do to build the four modules. Following chapter discusses the discussion.

# Chapter 07

# Evaluation

## 7.1 Introduction

In the previous chapter we discussed descriptively about the implementation of our system. In this chapter we discuss the results we obtained from our analysis.

## 7.2 Content modification using summarization with lexical simplification

There are two main phases in this module. First phase outcome is summarized text and the second phase outcome is a simplified summary. Initially evaluation was done separately in two phases. For the summarization, the dataset has been divided into 80:20 ratios which mean 80% data is used for training and 20% data is used for testing purposes. Same as CWI dataset has been divided into training and testing dataset. Here I evaluated accuracy, ROUGE Score and f1 score. Human evaluation is done for summarization to evaluate the grammar by a professional. Overall evaluation is done manually.

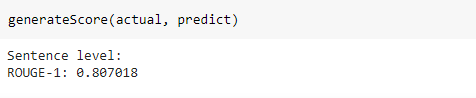


Figure 26: 7.1 Rouge score: evaluation results for summarization

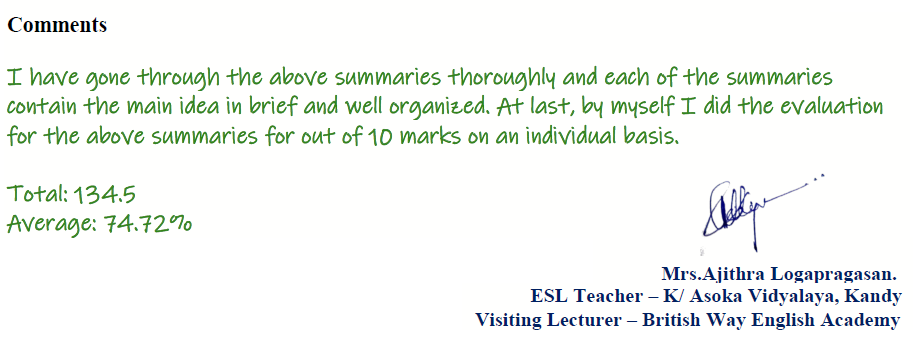


Figure 27: 7.2 Human Evaluation results for summarization

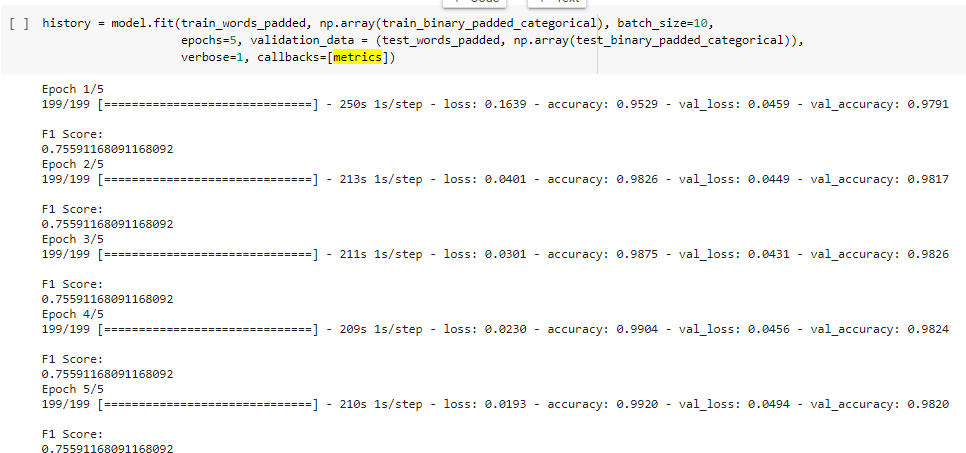
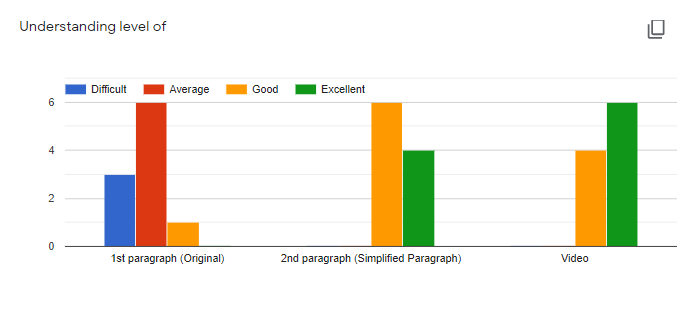


Figure 28: 7.3 F1 Score: Evaluation Results for lexical simplification

Figure 29: 7.4 Manual evaluation results for content modification

## 7.3 Information extraction from sentences

In this module there are two outcomes. First outcome is the Sentimental analysis, Age categorizes, and the second outcome is Text analysis, Semantic analysis. In this project Sentiment is categorized into three groups as positive, neutral, negative and Age categorized into five groups as Grade 1, Grade 2, Grade 3, Grade 4, Grade 5. We have evaluated the Sentiment analysis by dividing the training and testing into 80:20 ratios. I have used 80% data for training and 20% for testing. The training accuracy is 0.9646 and the testing accuracy is 0.6499. And I evaluate the final output video manually with the kids who are suitable for that video age group.



Figure 30: 7.5 Training Results for Sentiment analysis

Figure 31: 7.6 Testing Results for Sentiment analysis

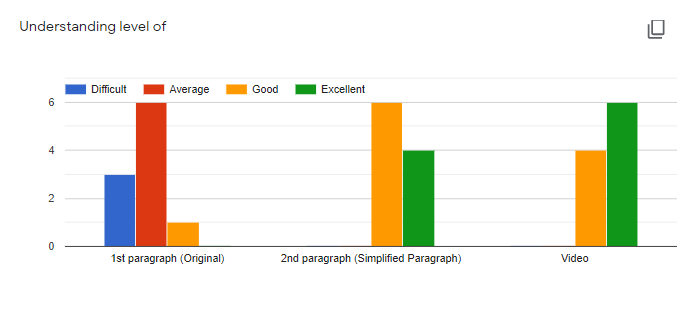


Figure 32: 7.7 Manual evaluation results for generated video

## 7.4 Outcomes of Predicting the similar sentence and finding the most appropriate answer

In this module there are two outcomes. First outcome was based on the sentence similarity prediction and the second outcome was based on finding the most appropriate answer. For the first outcome, we evaluated the supervised learning model without considering the root match index feature and divided the training and testing data into 80:20 ratio. The sentence similarity model based on logistic regression gives accuracy about 57.23. After evaluating this model, we have evaluated the model including the root match feature. This time classifier is evaluated using Multinomial Logistic regression, Random Forest classifier and XGboost classifier and it gives the accuracy as 64.44, 67.57 and 67.49, respectively. Consider this outcome when we added a feature based on root match the accuracy improved around 5%. Out of these supervised learning techniques Random Forest Regressor technique performed well. When we consider the next module based on transfer learning. The model manages to predict the start and end index of the answer around 62.4883% in exact match metric.

##### 

| **ML Algorithm** | **Number of Features** | **Precision** | **Recall** | **Accuracy**  **(in %)** |
| --- | --- | --- | --- | --- |
| Multiclass Logistic Regression | 2 | 0.567 | 0.583 | 57.23 |
| Multiclass Logistic Regression | 3 | 0.638 | 0.651 | 64.44 |
| Random Forest Regressor  min samples leaf=8  no of estimators=60 | 3 | 0.663 | 0.689 | 67.57 |
| XGBoost classifier  max depth = 3  min child weight = 5  learning rate = 0.2 | 3 | 0.667 | 0.683 | 67.49 |

##### 

##### Figure 33: 7.8 Performance evaluation of Classification Algorithms

##### 

## Conclusion and Further Work

In this era, Video-based learning is a widely accepted e-learning trend, and it’s gaining momentum. Visually telling a content has become an important factor in education, entertainment, and commerce. Children can more easily understand content through a video than by listening to the audio or reading a book. Because it is efficient, entertaining to watch and easily conveys the meaning of the content. So, most people make videos to explain their content. This process takes several steps like summarizing the content, searching images on the internet and other edits to sync the searched images with the audio. Also, it takes several weeks. People do not have much time to spend on video making. So, this project proposes a system for kids which can produce a video by combining a set of images for a /text content and a question and answering system. Also, kids can engage with the video and ask questions and get the answers. To develop our system, we gather a dataset from educational sites and teachers who are teaching to children. After that we start our individual part separately, for that we want to get a proper idea about the project and our parts. So, we study the previous research and research papers. Gather good knowledge about the children's education. After that we study about the proper technology to do this research then we start our initial implementation.in this project. For every component we researched the existing literature review and came up with new solutions which enhance the performance of the already available system. Every part in our system used a machine learning approach and was evaluated using different algorithms. The models we created are still not good enough to use in the daily learning activities of students, but with advanced machine learning approaches and domain specific data set We can increase the performance of each module. We planned to add more features to enhance the learning ability of kids. We planned to use some new techniques to achieve state of the art performance. This research is going to be a promising area in the future.

# References

[1] E. Hovy and C-Y Lin, (1997), “Automated Text Summarization in SUMMARIST”, Proceedings of the Workshop of Intelligent Scalable Text Summarization.

[2] P.E. Genest and G. Lapalme, (2011) "Framework for abstractive summarization using text -to-text generation," in Proceedings of the Workshop on Monolingual Text- To-Text Generation, pp. 64-73.

[3] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gül.ehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. CoRR abs/1406.1078 (2014). arXiv:1406.1078 <http://arxiv.org/abs/1406.1078>

[4] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. CoRR abs/1409.3215 (2014). arXiv:1409.3215 <http://arxiv.org/abs/1409.3215>

[5] Alamri, F., Kalkan, S., and Pugeault, N., “Transformer-Encoder Detector Module: Using Context to Improve Robustness to Adversarial Attacks on Object Detection”, <i>arXiv e-prints</i>, 2020.

[6] J. De Belder, M.-F. Moens, Text simplification for children, In Proceedings of the 2010 SIGIR Workshop on Accessible Search Systems (2010) 19–26.

[7] G. H. Paetzold, L. Specia, Unsupervised lexical simplification for nonnative speakers., in: AAAI, 2016, pp. 3761–3767.

[8] G. Paetzold, L. Specia, Lexical simplification with neural ranking, in: ACL: Volume 2, Short Papers, 2017, pp. 34–40.

[9] S. Gooding, E. Kochmar, Recursive context-aware lexical simplification, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing

(EMNLP-IJCNLP), 2019, pp. 4855–4865.

[10]W. Coster, D. Kauchak, Simple english wikipedia: a new text simplification task, in: ACL, 2011, pp. 665–669.

[11] T. Wang, P. Chen, K. Amaral, J. Qiang, An experimental study of lstm encoderdecoder model for text simplification, arXiv preprint arXiv:1609.03663.

[12] S. Nisioi, S. Stajner, S. P. Ponzetto, L. P. Dinu, Exploring neural text ˇ simplification models, in: ACL, Vol. 2, 2017, pp. 85–91.

[13] Y. Dong, Z. Li, M. Rezagholizadeh, J. C. K. Cheung, Editnts: An neural programmer-interpreter model for sentence simplification through explicit editing, in: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, 2019, pp. 3393–3402.

[14] M. Shardlow, A survey of automated text simplification, International Journal of Advanced Computer Science and Applications 4 (1) (2014) 58–70.

[15] G. H. Paetzold, L. Specia, A survey on lexical simplification, in: Journal of Artificial Intelligence Research, Vol. 60, 2017, pp. 549–593.

[16] M. Lesk, Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone, in: Proceedings of the 5th Annual International Conference on Systems Documentation, 1986, pp. 24–26.

[17] M. Maddela, W. Xu, A word-complexity lexicon and a neural readability ranking model for lexical simplification, in: EMNLP, 2018, pp. 3749– 3760.

[18] E. Pavlick, C. Callison-Burch, Simple ppdb: A paraphrase database for simplification, in: ACL: Volume 2, Short Papers, 2016, pp. 143–148.

[19] S. Devlin, J. Tait, The use of a psycholinguistic database in the simplification of text for aphasic readers, Linguistic Databases 1 (1998) 161173.

[20] S. Petersen. Natural language processing tools for reading level assessment and text simplification for bilingual education. PhD thesis, University of Washington, 2007.

[21]L. Feng, N. Elhadad, and M. Huenerfauth. Cognitively motivated features for readability assessment. In Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (EACL), pages 229{237. Association for Computational Linguistics, 2009.

[22] Martin Ponweiser (2012) Latent Dirichlet Allocation in R, Vienna University of Business and Economics.

[23]A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, “Learning word vectors for sentiment analysis,” in Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies. Association for Computational Linguistics, June 2011, pp. 142–150.

[24]T. Mikolov, K. Chen, G. S. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” CoRR, vol. abs/1301.3781, 2013

[25]M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, “Deep contextualized word representations,” arXiv preprint arXiv:1802.05365, 2018.

[26]J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” in NAACLHLT, 2018.

[27] Yao, H., Liu, H., & Zhang, P. (2018). A novel sentence similarity model with word embedding based on convolutional neural network. Concurrency and Computation: Practice and Experience, 30(23), e4415.

[28] Hassanzadeh, H., Groza, T., Nguyen, A., & Hunter, J. (2015). A Supervised Approach to Quantifying Sentence Similarity: With Application to Evidence Based Medicine. PLOS ONE, 10(6), e0129392

[29] Wang, Zhiguo & Mi, Haitao & Ittycheriah, Abraham. (2016). Sentence Similarity Learning by Lexical Decomposition and Composition.

[30] Pawar, Atish & Mago, Vijay. (2018). Calculating the similarity between words and sentences using a lexical database and corpus statistics.

[31] Min, S., Seo, M., & Hajishirzi, H. (2017). Question Answering through Transfer Learning from Large Fine-grained Supervision Data. Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers).

[32] Lee, Chia-Hsuan & Lee, Hung-yi. (2019). Cross-Lingual Transfer Learning for Question Answering.

[33] Zhang, R., Gangi Reddy, R., Sultan, M. A., Castelli, V., Ferritto, A., Florian, R., Sarioglu Kayi, E., Roukos, S., Sil, A., & Ward, T. (2020). Multi-Stage Pre-training for Low-Resource Domain Adaptation. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Published. https://doi.org/10.18653/v1/2020.emnlp-main.440

# Appendix A: Individual Contribution to the Project

There are three members in the group and the project has been divided into three main modules. These modules will be integrated together and a complete system will be built. The contributions of each member in the group are described below.

### Name of Student: Dhushanthini.A 164024N

My module in our research part is Content modification using summarization with lexical simplification. Our main aim is enhancing kids' learning through video content. Video will be produced for a given education related content. Therefore I have to modify the text content for the kid's specific domain. My main input is text content.

Initially I have collected the News article dataset for summarization. Transfer learning self-attention model has been used for summarization. Then summarized content will have been passed to lexical simplifier to simplify the content. I have used complex word identification (CWI) dataset for lexical simplification. In the lexical simplification process the main aim is to replace difficult words with simpler ones. There are mainly three steps in lexical simplification. They are, identify the complex words in a given sentence, generate candidates using BERT´s masked language model to get possible word candidates, select the best candidates based on Zipf values. Here complex word identification model is used to identify the complex word for a given sentence.

Finally I have integrated summarization and lexical simplification. The input is large content and output will be kids understandable simplified summary.

### 

### 

### 

### Name of Student: M.Jeevahasan 164064K

As a member of this group I have been working on the module Information extraction from sentences. To do this task I am working on,

* Sentimental analysis, Age categorize
* Text analysis, Semantic analysis

I get the summarized sentences from the Text Summarization module and analyze them. So, I can make a video for the given audio with the suitable images which I get for the particular sentences or word from google. A basic task in sentiment analysis is classifying the polarity, motion Detection, Intent Detection; and additionally I am going to categorize the sentences age wise according to the children’s capability. Above 14 years video will be mentioned and adult contents and violence contents will be ignored from the video.

After that I will do semantic analysis. Semantic analysis is the process of relating syntactic structures, from the levels of phrases, clauses, sentences and paragraphs to the level of the writing as a whole, to their language independent meanings. Through these techniques I am going to find the relevant searching queries which are suitable for the sentence within the domain. In order to retrieve the image, I am going to use google image searcher. Using selenium I can scrap for google images. Selenium is a portable framework for testing web applications. Using MoviePy I will edit videos and do basic operations. I will add the images according to the audio file time frame. So the children can easily learn information about the particular sentence with the images while watching the video. In the end the system will output a video with age limit using the summary sentences age limit. According to the majority age limit we can get the applicable age.

### 

### Name of Student: L.Araharan 164007P

In this project my part is to find the sentence which has the right answer and find the appropriate answer for the question. I have gone through many research papers and identified the methods to implement the above two components. In the data collection part, I collected data from the Stanford question answering dataset. In addition to that I manually annotated data using QA annotator tool. I preprocessed the question answering dataset and created a vector representation of each sentence and question using the infersent sentence embedding technique. After representing each sentence and question into vector form. I created features based on Euclidean distance and cosine similarity. I further created another feature based on the root match index. It is based on a dependency parse tree to compare sentence and question pair root strings. There were many research papers which used neural networks to predict the sentence similarity, but I proposed a method using supervised learning techniques. For this task I have used many supervised learning techniques. I have evaluated the model using various supervised learning algorithms. Among all the classifiers I chose the classifier which gave the highest accuracy.

In the second component of my part, I preprocessed data in different forms. Which consist of the start index and end index of the answer. I have created a function to identify the end index of the answer which is not specified in the dataset. After creating the end index, I tokenized the context question pair into encoding objects. For this component I have used transfer learning techniques to train the existing model. I evaluated the model using the exact match metric. Finally, I built a system which is capable of identifying the similar sentence and able to predict the appropriate answer for the question.

# Appendix B - Code Implementations

## 

## Content modification using summarization with lexical simplification

**Summarization**

* Import Modules

!pip install transformers==2.9.0

import numpy as np

import pandas as pd

import torch

from torch.utils.data import Dataset, DataLoader

from transformers import T5Tokenizer, T5ForConditionalGeneration

from torch import cuda

* Get Dataset

from google.colab import drive

drive.mount('/content/drive')

* Read Dataset

%cd sample\_data

df = pd.read\_csv('content\_extraction\_dataset.csv', encoding='latin-1')

df = df[['text', 'ctext']]

df.ctext = 'summarize: ' + df.ctext

print(df.head())

df.head(20)

* Dataset creation to read the dataframe

class CustomDataset(Dataset):

def \_\_init\_\_(self, dataframe, tokenizer, source\_len, summ\_len):

self.tokenizer = tokenizer

self.data = dataframe

self.source\_len = source\_len

self.summ\_len = summ\_len

self.text = self.data.text

self.ctext = self.data.ctext

def \_\_len\_\_(self):

return len(self.text)

def \_\_getitem\_\_(self, index):

ctext = str(self.ctext[index])

ctext = ' '.join(ctext.split())

text = str(self.text[index])

text = ' '.join(text.split())

source = self.tokenizer.batch\_encode\_plus([ctext], max\_length=self.source\_len, pad\_to\_max\_length=True, return\_tensors='pt')

target = self.tokenizer.batch\_encode\_plus([text], max\_length=self.summ\_len, pad\_to\_max\_length=True, return\_tensors='pt')

source\_ids = source['input\_ids'].squeeze() source\_mask = source['attention\_mask'].squeeze() target\_ids = target['input\_ids'].squeeze()

target\_mask = target['attention\_mask'].squeeze()

return {

'source\_ids': source\_ids.to(dtype=torch.long),

'source\_mask': source\_mask.to(dtype=torch.long),

'target\_ids': target\_ids.to(dtype=torch.long),

'target\_ids\_y': target\_ids.to(dtype=torch.long)

}

* Hyper Parameters

TRAIN\_BATCH\_SIZE = 5

VALID\_BATCH\_SIZE = 2

TRAIN\_EPOCHS = 7

VAL\_EPOCHS = 2

LEARNING\_RATE = 1e-4

SEED = 42

MAX\_LEN = 512

SUMMARY\_LEN = 150

# Set random seeds and deterministic pytorch for reproducibility

torch.manual\_seed(SEED) # pytorch random seed

np.random.seed(SEED) # numpy random seed

torch.backends.cudnn.deterministic = True

* Add Tokenizer, train and test data loader creation

tokenizer = T5Tokenizer.from\_pretrained("t5-base")

train\_size = 0.8

train\_dataset = df.sample(frac=train\_size, random\_state=SEED).reset\_index(drop=True)

val\_dataset =df.drop(train\_dataset.index).reset\_index(drop=True)

print("FULL Dataset: {}".format(df.shape))

print("TRAIN Dataset: {}".format(train\_dataset.shape))

print("TEST Dataset: {}".format(val\_dataset.shape))

CustomDataset

training\_set = (train\_dataset, tokenizer, MAX\_LEN, SUMMARY\_LEN)

validating\_set = CustomDataset(val\_dataset, tokenizer, MAX\_LEN, SUMMARY\_LEN)

# Defining the parameters for creation of dataloaders

train\_params = {

'batch\_size': TRAIN\_BATCH\_SIZE,

'shuffle': True,

'num\_workers': 0

}

val\_params = {

'batch\_size': VALID\_BATCH\_SIZE,

'shuffle': False,

'num\_workers': 0

}

training\_loader = DataLoader(training\_set, \*\*train\_params)

val\_loader = DataLoader(validating\_set, \*\*val\_params)

* Model

class SummaryGenerator(nn.Module):

def \_\_init\_\_(self):

super(SummaryGenerator, self).\_\_init\_\_()

self.summarizer = T5ForConditionalGeneration.from\_pretrained("t5-base")

self.summarizer.dropout\_rate = 0.1

self.summarizer.num\_layers = 6

def forward(self, input\_ids, attention\_mask,decoder\_input\_ids,lm\_labels):

pooled\_output = self.summarizer(

input\_ids=input\_ids,

attention\_mask=attention\_mask,

decoder\_input\_ids=y\_ids,

lm\_labels=lm\_labels

)

return pooled\_output

model = SummaryGenerator()

model = model.to(device)

* Optimizer

optimizer = torch.optim.Adam(params=model.parameters(), lr=LEARNING\_RATE)

* Train

def train(epoch, tokenizer, model, device, loader, optimizer):

model.train()

for \_, data in enumerate(loader, 0):

y = data['target\_ids'].to(device, dtype=torch.long)

y\_ids = y[:, :-1].contiguous()

lm\_labels = y[:, 1:].clone().detach()

lm\_labels[y[:, 1:] == tokenizer.pad\_token\_id] = -100

ids = data['source\_ids'].to(device, dtype=torch.long)

mask = data['source\_mask'].to(device, dtype=torch.long)

outputs = model(input\_ids=ids, attention\_mask=mask, decoder\_input\_ids=y\_ids, lm\_labels=lm\_labels)

loss = outputs[0]

if \_ % 500 == 0:

print(f'Epoch: {epoch}, Loss: {loss.item()}')

optimizer.zero\_grad()

loss.backward()

optimizer.step()

# xm.optimizer\_step(optimizer)

# xm.mark\_step()

for epoch in range(TRAIN\_EPOCHS):

train(epoch, tokenizer, model, device, training\_loader, optimizer)

* Save the weight

from google.colab import drive

drive.mount('/content/drive')

model.save\_pretrained("./drive/MyDrive/Research\_summary/kids\_summary\_dataset")

tokenizer.save\_pretrained("./drive/MyDrive/Research\_summary/kids\_summary\_dataset")

* Validation

def validate(epoch, tokenizer, model, device, loader):

model.eval()

predictions = []

actuals = []

with torch.no\_grad():

for \_, data in enumerate(loader, 0):

y = data['target\_ids'].to(device, dtype = torch.long)

ids = data['source\_ids'].to(device, dtype = torch.long)

mask = data['source\_mask'].to(device, dtype =torch.long)

generated\_ids = model.generate(

input\_ids = ids,

attention\_mask = mask,

max\_length=150,

num\_beams=2,

repetition\_penalty=2.5,

length\_penalty=1.0,

early\_stopping=True

)

preds = [tokenizer.decode(g, skip\_special\_tokens=True, clean\_up\_tokenization\_spaces=True) for g in generated\_ids]

target = [tokenizer.decode(t, skip\_special\_tokens=True, clean\_up\_tokenization\_spaces=True)for t in y]

if \_%100==0:

print(f'Completed {\_}')

predictions.extend(preds)

actuals.extend(target)

return predictions, actuals

for epoch in range(VAL\_EPOCHS):

predictions, actuals = validate(epoch, tokenizer, model, device, val\_loader)

final\_df = pd.DataFrame({'Generated Text':predictions,'Actual Text':actuals})

final\_df.to\_csv('./drive/MyDrive/Research\_summary/predictions.csv')

print('Output Files generated for review')

* Test

text ="""

Any time an astronaut gets out of a vehicle while in space, it is called a spacewalk. A spacewalk is also called an EVA. EVA stands for extravehicular activity. The first person to go on a spacewalk was Alexei Leonov. He was from Russia. The first spacewalk was on March 18, 1965. It was 10 minutes long. The first American to go on a spacewalk was Ed Whi te. His spacewalk was on June 3, 1965, during the Gemini 4 mission. Whi te's spacewalk lasted 23 minutes. Today, astronauts go on spacewalks outside the International Space Station. Astronauts go on spacewalks for many reasons. Spacewalks let astronauts work outside their spacecraft while still in space. Astronauts can do science experiments on a spacewalk. Experiments can be placed on the outside of a spacecraft. This lets scientists learn how being in space affects different things. When astronauts go on spacewalks, they wear space suits to keep themselves safe. Inside spacesuits, astronauts have the oxygen they need to breathe. They have the water they need to drink. Astronauts put on their spacesuits several hours before a spacewalk. The suits are pressurized. This means that the suits are filled with oxygen. Once in their suits, astronauts breathe pure oxygen for a few hours.Astronauts practice spacewalks underwater in a large swimming pool. Astronauts practice spacewalks underwater in a large swimming pool . The pool is called the Neutral Buoyancy Laboratory, or NBL. It is near NASA's Johnson Space Center in Houston, Texas. The pool holds 6.2 million gallons of water. Astronauts train seven hours in the pool for every one hour they will spend on a spacewalk.

"""

preprocess\_text = text.strip().replace("\n","")

t5\_prepared\_Text = "summarize: "+preprocess\_text

print ("original text preprocessed: \n", preprocess\_text)

tokenized\_text = tokenizer.encode(t5\_prepared\_Text, return\_tensors="pt").to(device)

# summmarize

summary\_ids = model.generate(tokenized\_text,

num\_beams=4,

no\_repeat\_ngram\_size=2,

min\_length=50,

max\_length=150,

early\_stopping=True)

output = self.tokenizer.decode(summary\_ids[0], skip\_special\_tokens=True)

print ("\n\nSummarized text: \n",output)

**Lexical Simplification**

* Load the libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from collections import namedtuple

from nltk import word\_tokenize

from functools import lru\_cache

import re

import unicodedata

import sys

from collections import Counter

import nltk

# nltk.download('brown')

from nltk.corpus import brown

from nltk import word\_tokenize

# nltk.download('punkt')

# nltk.download('stopwords')

# nltk.download('wordnet')

from nltk.corpus import stopwords

from nltk import pos\_tag

# nltk.download('averaged\_perceptron\_tagger')

import pickle

import json

* **Part 1: Complex word identification model**

Dataset = namedtuple('Dataset', 'name, train, test')

Model = namedtuple('Model', 'type, name, dimension, corpus, model')

* Get the dataset to train the CWI model

!wget <https://www.inf.uni-hamburg.de/en/inst/ab/lt/resources/data/complex-word-identification-dataset/cwishareddataset.zip>

!unzip cwishareddataset.zip

from google.colab import drive

drive.mount('/content/drive')

* Load the dataset

pd.set\_option('display.max\_columns', 500)

pd.set\_option('display.max\_colwidth', 200)

MAIN\_PATH\_DATASET = "traindevset/english/"

genres = ['Wikipedia', 'WikiNews', 'News']

datasets = ['Train', 'Dev']

columns = ['id', 'sentence', "start", "end", "target",

"nat", "non\_nat", "nat\_marked", "non\_nat\_marked", "binary", "prob"]

datasets = [Dataset('Wikipedia', 'Train', 'Dev'),

Dataset('WikiNews', 'Train', 'Dev'),

Dataset('News', 'Train', 'Dev')]

feature\_categories = []

def load\_df(path):

df = pd.read\_csv(path, header=None, sep = "\t")

df.columns = columns

return df

datasets = [Dataset(d.name, load\_df(MAIN\_PATH\_DATASET + d.name + '\_' + d.train + '.tsv'),

load\_df(MAIN\_PATH\_DATASET + d.name + '\_' + d.test + '.tsv'))

for d in datasets]

* Get the glove embedding

!wget http://nlp.stanford.edu/data/glove.6B.zip

!unzip glove.6B.zip -d embeddings

* Load the embedding model

from gensim.test.utils import datapath, get\_tmpfile

from gensim.models import KeyedVectors

from gensim.scripts.glove2word2vec import glove2word2vec

MAIN\_PATH = 'embeddings/'

glove\_models = []

glove\_defs = [ Model('glove', 'glove.6B.300d.txt', 300, 'wikipedia+gigaword5', None)]

for model in glove\_defs:

glove\_file = MAIN\_PATH + model.name

tmp\_file = get\_tmpfile(model.name + '-temp')

glove2word2vec(glove\_file, tmp\_file)

vecs = KeyedVectors.load\_word2vec\_format(tmp\_file)

glove\_models.append(Model(model.type, model.name, model.dimension, model.corpus, vecs))

print('load model : {}'.format(model.name))

print(glove\_models)

* Process the dataset to format

dataframe = datasets[0].train[0:30]

def overlaps(start1, end1, start2, end2):

return bool(range(max(start1, start2), min(end1, end2)+1))

def extract\_ngrams\_group(group):

targets = zip(group['target'].values.tolist(), group['start'].values.tolist(),

group['end'].values.tolist(), group['binary'].values.tolist())

for word, start, end, binary in targets:

tokens = word.split()

if len(tokens)>1:

olap\_words = [(w, b) for w, s, e, b in targets if overlaps(start, end, s, e)]

grouped = dataframe.groupby('sentence').apply(lambda group : extract\_ngrams\_group(group))

import nltk

nltk.download('brown')

wordlist\_lowercased = set(i.lower() for i in brown.words())

print (len(wordlist\_lowercased))

tbl = dict.fromkeys(i for i in range(sys.maxunicode)

if unicodedata.category(chr(i)).startswith('P'))

def remove\_punctuation(text):

return text.translate(tbl)

@lru\_cache(maxsize=None)

def all\_tokens\_with\_index(context):

curr\_pos = 0

targets = []

j = 0

w = 0

curr\_split = ''

ctx\_split = context.split()

whitespaces = re.findall('\s+', context)

num\_whitespaces = [len(token) for token in whitespaces]

num\_whitespaces.append(1)

tokens = word\_tokenize(context)

tokens = ['"' if token not in context else token for token in tokens]

for index, token in enumerate(tokens, 1):

targets.append((token, index, curr\_pos, (curr\_pos + len(token))))

curr\_pos += len(token)

curr\_split += token

if ctx\_split[j] == curr\_split:

curr\_pos += num\_whitespaces[w]

j += 1

w += 1

curr\_split = ''

return [val for val in targets if val[0] != '"']

def build\_vocabulary(sentences, embedding\_model, dimension):

all\_words = [tpl[0] for sentence in sentences for tpl in sentence['seq']] + list(wordlist\_lowercased)

print('# Words : {}'.format(len(all\_words)))

counter = Counter(all\_words)

vocab\_size = len(counter) + 1

print('# Vocab : {}'.format(vocab\_size))

print('# embeding model : {}'.format(len(embedding\_model.vocab)))

word2index = {word : index for index, (word, count) in enumerate(counter.most\_common(), 1)}

index2word = {index : word for word, index in word2index.items()}

# +1 required for pad token

embedding\_matrix = np.zeros(((vocab\_size), dimension))

missing\_embed\_words = []

i\_ = 0

for word, index in word2index.items():

if word in embedding\_model.vocab:

embedding = embedding\_model[word]

else:

i\_ +=1

continue

embedding\_matrix[index] = embedding

missing\_embed\_count = len(missing\_embed\_words)

print('# Words missing embedding : {}'.format(missing\_embed\_count))

print('Embedding shape : {}'.format(embedding\_matrix.shape))

print("i: ", i\_ )

return word2index, index2word, embedding\_matrix

def forward\_transformation(dataframe, lowercase = True, filter\_punc = True, filtering = "a132"):

grouped = dataframe.groupby('sentence').apply(lambda row :

{'sent\_id' : list(set(row['sent\_id']))[0],

'sentence' : list(set(row['sentence']))[0],

'tags': [tag for tag in zip(row['target'],

row['start'], row['end'], row['binary'], row['prob'])]})

sentences = []

for vals in grouped:

sent\_id = vals['sent\_id']

sentence = vals['sentence']

tags = vals['tags']

tags\_without\_labels = [(word, start, end) for word, start, end, binary, prob in tags]

all\_tokens = all\_tokens\_with\_index(sentence)

sent\_repr = [(word, start, end, tags[tags\_without\_labels.index((word, start, end))][3],

tags[tags\_without\_labels.index((word, start, end))][4])

if (word, start, end) in tags\_without\_labels

else (word, start, end, 0, 0.0) for word, index, start, end in all\_tokens]

if lowercase:

sent\_repr = [(word.lower(), start, end, binary, prob)

for word, start, end, binary, prob in sent\_repr]

if filter\_punc:

sent\_repr = list(filter(lambda vals : remove\_punctuation(vals[0]), sent\_repr))

if filtering:

sent\_repr = list(filter(lambda vals : vals[0] != "'s", sent\_repr))

sent\_repr = list(filter(lambda vals : vals[0] != "``", sent\_repr))

sentences.append({'sent\_id' : sent\_id, 'sentence' : sentence, 'seq' : sent\_repr})

return sentences

def split\_sentence\_seqs(sentences):

words, start\_end, binary, prob = [], [], [] ,[]

for sent in sentences:

sequence = sent['seq']

curr\_w, curr\_se, curr\_b, curr\_p = map(list, zip(\*[(vals[0], (vals[1], vals[2]), vals[3], vals[4]) for vals in sequence]))

words.append(curr\_w)

start\_end.append(curr\_se)

binary.append(curr\_b)

prob.append(curr\_p)

return words, start\_end, binary, prob

datasets.append(Dataset('train\_all\_test\_wiki',

datasets[0].train.append(datasets[1].train).append(datasets[2].train), datasets[0].test))

# Append train and test set

dataset\_sel = datasets[3]

train\_num\_rows = dataset\_sel.train.shape[0]

train\_num\_sents = len(list(set(dataset\_sel.train.sentence.values.tolist())))

test\_num\_rows = dataset\_sel.test.shape[0]

test\_num\_sents = len(list(set(dataset\_sel.test.sentence.values.tolist())))

dataset = dataset\_sel.train.append(dataset\_sel.test)

dataset['sent\_id'] = dataset.groupby('sentence').ngroup()

dataset\_num\_rows = dataset.shape[0]

dataset\_num\_sents = len(list(set(dataset.sentence.values.tolist())))

print('# Rows train : {}'.format(train\_num\_rows))

print('# Rows test : {}'.format(test\_num\_rows))

print('# Rows dataset : {}'.format(dataset\_num\_rows))

print('# Sents train : {}'.format(train\_num\_sents))

print('# Sents test : {}'.format(test\_num\_sents))

print('# Sents dataset : {}'.format(dataset\_num\_sents))

import nltk

nltk.download('punkt')

sentences = forward\_transformation(dataset)

train\_sentences = sentences[:train\_num\_sents]

test\_sentences = sentences[train\_num\_sents:]

words, start\_end, binary, prob = split\_sentence\_seqs(sentences)

sentence\_lens = [len(sent) for sent in words]

* Dimensions of the embedding and vectors for the model

embedding\_model = glove\_models[0].model

dimension = embedding\_model.vector\_size

word2index, index2word, embedding = build\_vocabulary(sentences, embedding\_model, dimension)

* Padding the input sentences and get the binary labels

words\_with\_indices = [[word2index[word] for word in sent] for sent in words]

sent\_lens = [len(sentence['seq']) for sentence in sentences]

sent\_max\_length = np.max(sent\_lens)

print('Max length sentence : {}'.format(sent\_max\_length))

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.utils import to\_categorical

words\_padded = pad\_sequences(maxlen=sent\_max\_length, sequences=words\_with\_indices, padding="post", value=0)

binary\_padded = pad\_sequences(maxlen=sent\_max\_length, sequences=binary, padding="post", value=0)

prob\_padded = pad\_sequences(maxlen=sent\_max\_length, sequences=prob, padding="post", value=0, dtype="float")

binary\_padded\_categorical = [to\_categorical(clazz, num\_classes=2) for clazz in binary\_padded]

* Split the train and test dataset

# (1) Training set

train\_words\_padded = words\_padded[:train\_num\_sents]

train\_binary\_padded = binary\_padded[:train\_num\_sents]

train\_binary\_padded\_categorical = binary\_padded\_categorical[:train\_num\_sents]

train\_prob\_padded = prob\_padded[:train\_num\_sents]

train\_start\_end = start\_end[:train\_num\_sents]

# (2) Test set

test\_words\_padded = words\_padded[train\_num\_sents:]

test\_binary\_padded = binary\_padded[train\_num\_sents:]

test\_binary\_padded\_categorical = binary\_padded\_categorical[train\_num\_sents:]

test\_prob\_padded = prob\_padded[train\_num\_sents:]

test\_start\_end = start\_end[train\_num\_sents:]

print('Training set length : {}'.format(len(train\_words\_padded)))

print('Test set length : {}'.format(len(test\_words\_padded)))

* Create the Keras’s to callback to validate the model on train

from sklearn.metrics import f1\_score

import tensorflow.keras.callbacks

from tensorflow.keras import backend as K

class Metrics(tensorflow.keras.callbacks.Callback):

def \_\_init\_\_(self, validation\_data):

self.f1\_scores = []

self.validation\_data = validation\_data

def on\_epoch\_end(self, batch, logs={}):

predict = np.asarray(self.model.predict(self.validation\_data[0]))

targ = self.validation\_data[1]

targ = np.array(targ)

shape = targ.shape

targ = targ.reshape((shape[0]\*shape[1], shape[2]))

targ = np.argmax(targ, axis = 1)

predict = predict.reshape((shape[0]\*shape[1]), shape[2])

predict = np.argmax(predict, axis = 1)

self.f1s=f1\_score(targ, predict)

print("\nF1 Score:")

print(f1\_score(targ, np.ones(shape[0]\*shape[1])))

self.f1\_scores.append(self.f1s)

return

* Create the Keras model for the Complex Word Identification Task

from keras.models import Model, Input

from keras import backend as K

from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirectional

vocab\_size = embedding.shape[0]

dimension = embedding.shape[1]

np.set\_printoptions(threshold=np.inf)

in\_seq = Input(shape=(sent\_max\_length,))

embed = Embedding(input\_dim=vocab\_size, output\_dim=dimension, \

weights=[embedding], input\_length=sent\_max\_length)(in\_seq)

drop = Dropout(0.1)(embed)

lstm = Bidirectional(LSTM(units=150, return\_sequences=True, recurrent\_dropout=0.1))(drop)

out = TimeDistributed(Dense(2, activation="softmax"))(lstm)

model = Model(in\_seq, out)

model.compile(optimizer="adam", loss='categorical\_crossentropy', metrics=["accuracy"])

model.summary()

metrics = Metrics((test\_words\_padded, np.array(test\_binary\_padded\_categorical)))

* Train the Model

history = model.fit(train\_words\_padded, np.array(train\_binary\_padded\_categorical), batch\_size=10, epochs=5, validation\_data = (test\_words\_padded, np.array(test\_binary\_padded\_categorical)), verbose=1, callbacks=[metrics])

* Check the performance by plot the chart

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'validation'], loc='upper left')

plt.show()

* Save the model

from google.colab import drive

drive.mount('/content/gdrive')

model\_save\_name = 'lexical\_trainedmodel\_cleaneddata1.h5'

path\_dir = F"/content/gdrive/My Drive/{model\_save\_name}"

with open(F"/content/gdrive/My Drive/train.pickle", 'wb') as f:

pickle.dump({'word2index': word2index, 'sent\_max\_length': sent\_max\_length}, f)

from keras.models import load\_model

model.save(path\_dir) # creates a HDF5 file 'model\_CWI\_full.h5'

* Retrieve the model from google drive

from keras.models import load\_model

model\_cwi = load\_model(path\_dir)

* Clean the data and remove the non-characters symbols

import nltk

nltk.download('stopwords')

stop\_words\_ = set(stopwords.words('english'))

def cleaner(word):

#Remove links

word = re.sub(r'((http|https)\:\/\/)?[a-zA-Z0-9\.\/\?\:@\-\_=#]+\.([a-zA-Z]){2,6}([a-zA-Z0-9\.\&\/\?\:@\-\_=#])\*','', word, flags=re.MULTILINE)

word = re.sub('[\W]', ' ', word)

word = re.sub('[^a-zA-Z]', ' ', word)

return word.lower().strip()

* Create the padded sequence

def process\_input(input\_text):

input\_text = cleaner(input\_text)

clean\_text = []

index\_list =[]

input\_token = []

index\_list\_zipf = []

for i, word in enumerate(input\_text.split()):

if word in word2index:

clean\_text.append(word)

input\_token.append(word2index[word])

else:

index\_list.append(i)

input\_padded = pad\_sequences(maxlen=sent\_max\_length, sequences=[input\_token], padding="post", value=0)

return input\_padded, index\_list, len(clean\_text)

def complete\_missing\_word(pred\_binary, index\_list, len\_list):

list\_cwi\_predictions = list(pred\_binary[0][:len\_list])

for i in index\_list:

list\_cwi\_predictions.insert(i, 0)

return list\_cwi\_predictions

* **Part 2: Generate candidates using BERT**

!pip install transformers

!pip install torch

import torch

from transformers import BertTokenizer, BertModel, BertForMaskedLM

bert\_model = 'bert-large-uncased'

tokenizer = BertTokenizer.from\_pretrained(bert\_model)

model = BertForMaskedLM.from\_pretrained(bert\_model)

model.eval(**)**

* **Part 3: Compute the Zipf values**

**!pip install wordfreq**

* **Get the candidates from Bert**

def get\_bert\_candidates(input\_text, list\_cwi\_predictions, numb\_predictions\_displayed = 10):

list\_candidates\_bert = []

for word,pred in zip(input\_text.split(), list\_cwi\_predictions):

if (pred and (pos\_tag([word])[0][1] in ['NNS', 'NN', 'VBP', 'RB', 'VBG','VBD' ])) or (zipf\_frequency(word, 'en')) <3.1:

replace\_word\_mask = input\_text.replace(word, '[MASK]')

text = f'[CLS]{replace\_word\_mask} [SEP] {input\_text} [SEP] '

tokenized\_text = tokenizer.tokenize(text)

masked\_index = [i for i, x in enumerate(tokenized\_text) if x == '[MASK]'][0]

indexed\_tokens = tokenizer.convert\_tokens\_to\_ids(tokenized\_text)

segments\_ids = [0]\*len(tokenized\_text)

tokens\_tensor = torch.tensor([indexed\_tokens])

segments\_tensors = torch.tensor([segments\_ids])

# Predict all tokens

with torch.no\_grad():

outputs = model(tokens\_tensor, token\_type\_ids=segments\_tensors)

predictions = outputs[0][0][masked\_index]

predicted\_ids = torch.argsort(predictions, descending=True)[:numb\_predictions\_displayed]

predicted\_tokens = tokenizer.convert\_ids\_to\_tokens(list(predicted\_ids))

list\_candidates\_bert.append((word, predicted\_tokens))

return list\_candidates\_bert

* Testing: Simplify the sentences

list\_texts = [

'wind energy is used to generate electricity. Wind is caused due to the sun shining and by heating of earth\'s surface by the Sun. The wind turbine helps to pump water, grind grain and saw wood for next 300 years. By the usage of aerodynamics, wind power is produced in different sizes and give great power output.']

import nltk

nltk.download('averaged\_perceptron\_tagger')

for input\_text in list\_texts:

new\_text = input\_text

input\_padded, index\_list, len\_list = process\_input(input\_text)

pred\_cwi = model\_cwi.predict(input\_padded)

pred\_cwi\_binary = np.argmax(pred\_cwi, axis = 2)

complete\_cwi\_predictions = complete\_missing\_word(pred\_cwi\_binary, index\_list, len\_list)

bert\_candidates = get\_bert\_candidates(input\_text, complete\_cwi\_predictions)

for word\_to\_replace, l\_candidates in bert\_candidates:

tuples\_word\_zipf = []

for w in l\_candidates:

if w.isalpha():

tuples\_word\_zipf.append((w, zipf\_frequency(w, 'en')))

tuples\_word\_zipf = sorted(tuples\_word\_zipf, key = lambda x: x[1], reverse=True)

new\_text = re.sub(word\_to\_replace, tuples\_word\_zipf[0][0], new\_text)

print("Original text: ", input\_text )

print("Simplified text:", new\_text, "\n")

**Evaluation script for summarization**

**!pip install easy-rouge**

**!pip install sentence\_transformers**

**from rouge import rouge\_n\_sentence\_level**

**from rouge import rouge\_l\_sentence\_level**

**from rouge import rouge\_n\_summary\_level**

**from rouge import rouge\_l\_summary\_level**

**from rouge import rouge\_w\_sentence\_level**

**from rouge import rouge\_w\_summary\_level**

**import pandas as pd**

**from sentence\_transformers import SentenceTransformer, util**

**import csv**

**from google.colab import drive**

**drive.mount('/content/drive')**

**evaluation\_df = pd.read\_csv("drive/MyDrive/Research\_summary/predictions\_final2.csv")**

**evaluation\_df.head()**

**def generateScore(actual\_summary, generated\_summary):**

**reference\_sentence = actual\_summary.split()**

**summary\_sentence = generated\_summary.split()**

**print('Sentence level:')**

**score = rouge\_n\_sentence\_level(summary\_sentence, reference\_sentence, 1)**

**print('ROUGE-1: %f' % score.f1\_measure)**

**actual = evaluation\_df["Actual Text"][0]**

**predict = evaluation\_df["Generated Text"][0]**

**print(actual)**

**generateScore(actual, predict)**

**Integration of summarization and lexical simplification**

* **Summarization testing**

**from transformers import T5Tokenizer, T5ForConditionalGeneration**

**class Summary:**

**def \_\_init\_\_(self):**

**self.device = 'cpu'**

**self.tokenizer = T5Tokenizer.from\_pretrained("./Summary\_Test/kids\_summary\_dataset\_trainedmodel")**

**self.model = T5ForConditionalGeneration.from\_pretrained('./Summary\_Test/kids\_summary\_dataset\_trainedmodel', return\_dict=True)**

**self.model = self.model.to(self.device)**

**def getSummary(self, ctext):**

**preprocess\_text = ctext.strip().replace("\n", "")**

**t5\_prepared\_Text = "summarize: " + preprocess\_text**

**print("original text preprocessed: \n", preprocess\_text)**

**tokenized\_text = self.tokenizer.encode(t5\_prepared\_Text, return\_tensors="pt").to(self.device)**

**# summmarize**

**summary\_ids = self.model.generate(tokenized\_text,**

**num\_beams=4,**

**no\_repeat\_ngram\_size=2,**

**min\_length=50,**

**max\_length=150,**

**early\_stopping=True)**

**output = self.tokenizer.decode(summary\_ids[0], skip\_special\_tokens=True)**

**print("\n\nSummarized text: \n", output)**

**return output**

* **Integration of summarization and lexical simplification**

**from flask import Flask, request, Response, jsonify**

**from Summary\_Test.test import Summary**

**from Lexical\_Simplification.test import predict**

**app = Flask(\_\_name\_\_)**

**summary\_obj = Summary()**

**summary\_text = ""**

**@app.route('/summary', methods=['GET', 'POST'])**

**def summary():**

**global summary\_text**

**data = request.form['ctext']**

**summary\_text = summary\_obj.getSummary(str(data))**

**print(" get Summry ", summary\_text)**

**stext = ''.join(summary\_text)**

**lexical\_text = predict(stext)**

**final\_result = {"Original Text": str(data), "Summary Text": str(stext),**

**"Lexical Simplification Text": str(lexical\_text)}**

**return jsonify(final\_resul**

**@app.route('/hello', methods=['GET', 'POST'])**

**def hello():**

**global summary\_text**

**data = request.form['ctext']**

**count = str(data).split(".")**

**print("Count is ",count)**

**print("Length of count is ", len(count))**

**first\_para = ""**

**for i in count[:int(len(count) / 2)]:**

**first\_para = first\_para + str(i) + ". "**

**second\_para = ""**

**for i in count[int(len(count) / 2):]:**

**second\_para = second\_para + str(i) + ". "**

**print("First para is ",first\_para)**

**print("Second para is ",second\_para)**

**first\_para\_Summary = summary\_obj.getSummary(str(first\_para))**

**second\_para\_Summary = summary\_obj.getSummary(second\_para)**

**print("First paragraph summary is ", first\_para\_Summary)**

**print("Second paragraph summary is ", second\_para\_Summary)**

**summary\_final = ''.join(first\_para\_Summary) + ''.join(second\_para\_Summary)**

**print("Final Summary is ", summary\_final)**

**lexical\_text = predict(str(summary\_final))**

**final\_result = {"Original Text": str(data), "Summary Text": str(summary\_final),"Lexical Simplification Text": str(lexical\_text)} return jsonify(final\_result)**

**if \_\_name\_\_ == '\_\_main\_\_':**

**app.run(host='0.0.0.0', debug=True)**

* **Lexical simplification testing**

from nltk.corpus import stopwords

tensorflow.keras.models import load\_model

import torch

from transformers import BertTokenizer, BertModel, BertForMaskedLM

from wordfreq import zipf\_frequency

import pickle

import re

from keras.preprocessing.sequence import pad\_sequences

import numpy as np

from nltk import pos\_tag

model\_cwi = load\_model('./Lexical\_Simplification/lexical\_trainedmodel\_cleaneddata.h5')

with open('./Lexical\_Simplification/train.pickle', 'rb') as f:

pickle1 = pickle.load(f)

word2index = pickle1['word2index']

sent\_max\_length = pickle1['sent\_max\_length']

print("max ",pickle1['sent\_max\_length'])

# Now, let´s define some useful functions in order to use the CWI with some out of samples sentences

# Function for clean the data and remove non characters symbols

stop\_words\_ = set(stopwords.words('english'))

def cleaner(word):

# Remove links

word = re.sub(r'((http|https)\:\/\/)?[a-zA-Z0-9\.\/\?\:@\-\_=#]+\.([a-zA-Z]){2,6}([a-zA-Z0-9\.\&\/\?\:@\-\_=#])\*',

'', word, flags=re.MULTILINE)

word = re.sub('[\W]', ' ', word)

word = re.sub('[^a-zA-Z]', ' ', word)

return word.lower().strip()

# Function for to create the padded sequence

def process\_input(input\_text):

input\_text = cleaner(input\_text)

clean\_text = []

index\_list = []

input\_token = []

index\_list\_zipf = []

for i, word in enumerate(input\_text.split()):

if word in word2index:

clean\_text.append(word)

input\_token.append(word2index[word])

else:

index\_list.append(i)

input\_padded = pad\_sequences(maxlen=sent\_max\_length, sequences=[input\_token], padding="post", value=0)

return input\_padded, index\_list, len(clean\_text)

def complete\_missing\_word(pred\_binary, index\_list, len\_list):

list\_cwi\_predictions = list(pred\_binary[0][:len\_list])

for i in index\_list:

list\_cwi\_predictions.insert(i, 0)

return list\_cwi\_predictions

bert\_model = 'bert-large-uncased'

tokenizer = BertTokenizer.from\_pretrained(bert\_model)

model = BertForMaskedLM.from\_pretrained(bert\_model)

model.eval()

zipf\_frequency('stop', 'en')

zipf\_frequency('thwart', 'en')

# Now the function to get the candidates out of BERT (MLM):

def get\_bert\_candidates(input\_text, list\_cwi\_predictions, numb\_predictions\_displayed=10):

list\_candidates\_bert = []

for word, pred in zip(input\_text.split(), list\_cwi\_predictions):

if (pred and (pos\_tag([word])[0][1] in ['NNS', 'NN', 'VBP', 'RB', 'VBG', 'VBD'])) or (

zipf\_frequency(word, 'en')) < 3.1:

replace\_word\_mask = input\_text.replace(word, '[MASK]')

text = f'[CLS]{replace\_word\_mask} [SEP] {input\_text} [SEP] '

tokenized\_text = tokenizer.tokenize(text)

masked\_index = [i for i, x in enumerate(tokenized\_text) if x == '[MASK]'][0]

indexed\_tokens = tokenizer.convert\_tokens\_to\_ids(tokenized\_text)

segments\_ids = [0] \* len(tokenized\_text)

tokens\_tensor = torch.tensor([indexed\_tokens])

segments\_tensors = torch.tensor([segments\_ids])

# Predict all tokens

with torch.no\_grad():

outputs = model(tokens\_tensor, token\_type\_ids=segments\_tensors)

predictions = outputs[0][0][masked\_index]

predicted\_ids = torch.argsort(predictions, descending=True)[:numb\_predictions\_displayed]

predicted\_tokens = tokenizer.convert\_ids\_to\_tokens(list(predicted\_ids))

list\_candidates\_bert.append((word, predicted\_tokens))

return list\_candidates\_bert

# Simplifying new sentences:

# Given a list of new sentences with complex words:

list\_texts = [

'The Risk That Students Could Arrive at School With the Coronavirus As schools grapple with how to reopen, new estimates show that large parts of the country would probably see infected students if classrooms opened now.',

'How a photograph of a young man cradling his dying friend sent me on a journey across India.',

'Pro-democracy parties, which had hoped to ride widespread discontent to big gains, saw the yearlong delay as an attempt to thwart them.',

'Night after night, calm gave way to chaos. See what happened between the protesters and the federal agents.',

'Contact Tracing Is Failing in Many States. Here is Why. Inadequate testing and protracted delays in producing results have crippled tracking and hampered efforts to contain major outbreaks.',

'After an initial decrease in the youth detention population, the rate of release has slowed, and the gap between white youth and Black youth has grown.'

'A laboratory experiment hints at some of the ways the virus might elude antibody treatments. Combining therapies could help, experts said.',

'Though I may not be here with you, I urge you to answer the highest calling of your heart and stand up for what you truly believe.',

'The research does not prove that infected children are contagious, but it should influence the debate about reopening schools, some experts said.',

'Dropping antibody counts are not a sign that our immune system is failing against the coronavirus, nor an omen that we can not develop a viable vaccine.',

'The Senate majority leader has said he will not approve a stimulus package without a “liability shield,” but top White House officials say they do not see it as essential.',

'Campaign efforts to refocus come as the president continues to push divisive messages that have frustrated his own party.'

]

def predict(sent):

new\_text = sent

input\_padded, index\_list, len\_list = process\_input(sent)

pred\_cwi = model\_cwi.predict(input\_padded)

pred\_cwi\_binary = np.argmax(pred\_cwi, axis=2)

complete\_cwi\_predictions = complete\_missing\_word(pred\_cwi\_binary, index\_list, len\_list)

bert\_candidates = get\_bert\_candidates(sent, complete\_cwi\_predictions)

for word\_to\_replace, l\_candidates in bert\_candidates:

tuples\_word\_zipf = []

for w in l\_candidates:

if w.isalpha():

tuples\_word\_zipf.append((w, zipf\_frequency(w, 'en')))

tuples\_word\_zipf = sorted(tuples\_word\_zipf, key=lambda x: x[1], reverse=True)

new\_text = re.sub(word\_to\_replace, tuples\_word\_zipf[0][0], new\_text)

# print("Original text: ", input\_text)

# print("Simplified text:", new\_text, "\n")

return new\_text

## Information extraction from sentences

**Sentiment analysis**

* Load the libraries

import pandas as pd

import numpy as np

from tensorflow import keras

from tensorflow.keras.preprocessing.text import text\_to\_word\_sequence, one\_hot

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.models import Sequential, Model

from tensorflow.keras.layers import Dense, LSTM, Softmax, Embedding, SpatialDropout1D, Bidirectional, Input

from tensorflow.keras.utils import to\_categorical

from transformers import TFBertModel, BertTokenizer

import tensorflow as tf

* Load data and encoding

df\_train = pd.read\_excel('datasets.xlsx', sheet\_name='train')

df\_test = pd.read\_excel('datasets.xlsx', sheet\_name='test')

def labeling(grade, sent):

if grade == 'grade1' and sent == 'positive':

a = 0

elif grade == 'grade1' and sent == 'negative':

a = 1

elif grade == 'grade1' and sent == 'neutral':

a = 2

elif grade == 'grade2' and sent == 'positive':

a = 3

elif grade == 'grade2' and sent == 'negative':

a = 4

elif grade == 'grade2' and sent == 'neutral':

a = 5

elif grade == 'grade3' and sent == 'positive':

a = 6

elif grade == 'grade3' and sent == 'negative':

a = 7

elif grade == 'grade3' and sent == 'neutral':

a = 8

elif grade == 'grade4' and sent == 'positive':

a = 9

elif grade == 'grade4' and sent == 'negative':

a = 10

elif grade == 'grade4' and sent == 'neutral':

a = 11

elif grade == 'grade5' and sent == 'positive':

a = 12

elif grade == 'grade5' and sent == 'negative':

a = 13

elif grade == 'grade5' and sent == 'neutral':

a = 14

else:

a = 15

return a

df\_train['label'] = df\_train[['Grade', 'Sentiment']].apply(lambda a: labeling(a.Grade, a.Sentiment), axis=1).astype('i')

df\_test['label'] = df\_test[['Grade', 'Sentiment']].apply(lambda a: labeling(a.Grade, a.Sentiment), axis=1).astype('i')

X\_train = df\_train['Sentence'].to\_list()

y\_train = df\_train['label'].to\_list()

X\_test = df\_test['Sentence'].to\_list()

y\_test = df\_test['label'].to\_list()

* Tokenizing

tokenizer = BertTokenizer.from\_pretrained("bert-base-uncased")

train\_data = tokenizer.batch\_encode\_plus(

X\_train,

add\_special\_tokens=True,

max\_length=50,

return\_attention\_mask=True,

return\_token\_type\_ids=True,

pad\_to\_max\_length=True,

return\_tensors="tf",

)

test\_data = tokenizer.batch\_encode\_plus(

X\_test,

add\_special\_tokens=True,

max\_length=50,

return\_attention\_mask=True,

return\_token\_type\_ids=True,

pad\_to\_max\_length=True,

return\_tensors="tf",

)

train\_input\_id = train\_data['input\_ids']

train\_token\_type\_id = train\_data['token\_type\_ids']

train\_attention\_mask = train\_data['attention\_mask']

test\_input\_id = test\_data['input\_ids']

test\_token\_type\_id = test\_data['token\_type\_ids']

test\_attention\_mask = test\_data['attention\_mask']

y\_train = to\_categorical(y\_train, num\_classes=16)

y\_test = to\_categorical(y\_test, num\_classes=16)

* Training

input\_ids = tf.keras.layers.Input(shape=(50,), dtype=tf.int32, name="input\_ids")

attention\_masks = tf.keras.layers.Input(shape=(50,), dtype=tf.int32, name="attention\_masks")

token\_type\_ids = tf.keras.layers.Input(shape=(50,), dtype=tf.int32, name="token\_type\_ids")

bert\_model = TFBertModel.from\_pretrained('bert-base-uncased',output\_hidden\_states=True)

bert\_model.trainable = True

outputs = bert\_model(input\_ids, attention\_mask=attention\_masks, token\_type\_ids=token\_type\_ids)

hidden\_states = outputs[2]

print(len(hidden\_states))

embedding\_output = hidden\_states[0]

bi\_lstm1 = Bidirectional(LSTM(256, dropout=0.2, recurrent\_dropout=0.2, return\_sequences=True))(embedding\_output)

bi\_lstm2 = Bidirectional(LSTM(256,dropout=0.2, recurrent\_dropout=0.2))(bi\_lstm1)

output = Dense(16, activation="softmax")(bi\_lstm2)

model = Model(inputs=[input\_ids, attention\_masks, token\_type\_ids], outputs=output)

keras.utils.plot\_model(model, "multi\_input\_and\_output\_model.png", show\_shapes=True)

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

history = model.fit([train\_input\_id, train\_attention\_mask, train\_token\_type\_id], y\_train, epochs=10, batch\_size=32, shuffle=True)

* Saving model

from google.colab import drive

drive.mount('/content/drive')

!ls /content/gdrive/MyDrive

save\_directory = "/content/drive/MyDrive/FYP/Final"

tf.saved\_model.save(model, export\_dir=save\_directory)

tokenizer.save\_pretrained(save\_directory)

* Testing model

results = model.evaluate([test\_input\_id, test\_attention\_mask, test\_token\_type\_id], y\_test, batch\_size=32)

* Getting sentiment and grade output with the model

for sentence in sentences:

speech = gTTS(text = sentence, lang = language, slow = True)

speech.save("../data/inputs/sentence.mp3")

duration = eyed3.load('../data/inputs/sentence.mp3').info.time\_secs

time.append(duration)

os.remove("../data/inputs/sentence.mp3")

data = tokenizer.batch\_encode\_plus(

sentence,

add\_special\_tokens=True,

max\_length=50,

return\_attention\_mask=True,

return\_token\_type\_ids=True,

pad\_to\_max\_length=True,

return\_tensors="tf",

)

input\_id = data['input\_ids']

token\_type\_id = data['token\_type\_ids']

attention\_mask = data['attention\_mask']

out = reloaded([input\_id, token\_type\_id, attention\_mask], training=False)[0]

out = np.argmax(out)

def decoding(out):

grade = ""

sentiment = ""

if out == 0:

grade = 'grade1'

sentiment = 'positive'

elif out == 1:

grade = 'grade1'

sentiment = 'negative'

elif out == 2:

grade = 'grade1'

sentiment = 'neutral'

elif out == 3:

grade = 'grade2'

sentiment = 'positive'

elif out == 4:

grade = 'grade2'

sentiment = 'negative'

elif out == 5:

grade = 'grade2'

sentiment = 'neutral'

elif out == 6:

grade = 'grade3'

sentiment = 'positive'

elif out == 7:

grade = 'grade3'

sentiment = 'negative'

elif out == 8:

grade = 'grade3'

sentiment = 'neutral'

elif out == 9:

grade = 'grade4'

sentiment = 'positive'

elif out == 10:

grade = 'grade4'

sentiment = 'negative'

elif out == 11:

grade = 'grade4'

sentiment = 'neutral'

elif out == 12:

grade = 'grade5'

sentiment = 'positive'

elif out == 13:

grade = 'grade5'

sentiment = 'negative'

elif out == 14:

grade = 'grade5'

sentiment = 'neutral'

elif out == 15:

grade = 'Adult'

sentiment = None

return grade, sentiment

grade, sent = decoding(out)

print('Sentiment is {sent} | Grade Group is {grade}'.format(sent=sent, grade=grade))

**Semantic analysis**

* Load the libraries

import pandas as pd

import nltk; nltk.download('stopwords')

import nltk

nltk.download('stopwords')

nltk.download('punkt')

from nltk.corpus import stopwords

stop\_words = stopwords.words('english')

stop\_words.extend(['from', 'subject', 're', 'edu', 'use'])

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

* Getting semantic output

if(catogory.semantic\_process\_data(sentence) == None):

key\_word = '{sentence},{sent}'.format(sentence=sentence, sent=sent)

else:

key\_word = '{sentence},{sent},{semantic}'.format(sentence=sentence, sent=sent, semantic=catogory.semantic\_process\_data(sentence))

print(key\_word)

new\_sentences.append(key\_word)

grade\_array.append(grade)

total\_sentiment.append(sent)

def semantic\_process\_data(inp\_qstn):

stop\_words = set(stopwords.words('english'))

word\_tokens = word\_tokenize(inp\_qstn)

filtered\_sentence = [w for w in word\_tokens if not w.lower() in stop\_words]

filtered\_sentence = []

for w in word\_tokens:

if w not in stop\_words:

filtered\_sentence.append(w)

for count in range(0, len\_category):

for inp\_type in (data[category\_list[count]]):

if str(w).lower() == str(inp\_type).lower():

print(w, 'belongs to the category : ', category\_list[count])

return w, category\_list[count]

**Image scraping**

* Load the libraries

from selenium import webdriver

import time

import requests

import io

from PIL import Image

import os

from video\_maker import video\_maker

import base64

DRIVER\_PATH = '../chromedriver'

* Image scraping and saving

def search(input, time):

i = 0

global inputs

global timeArray

timeArray = time

inputs = input

for data in inputs:

i = i+1

img\_name = str(i)

search\_term = data

search\_and\_download(

search\_term = search\_term,

driver\_path = DRIVER\_PATH,

img\_name = img\_name

)

def search\_and\_download(search\_term:str,driver\_path:str,img\_name:str, target\_path='../data/inputs/images',number\_images=1):

target\_folder = os.path.join(target\_path)

if not os.path.exists(target\_folder):

os.makedirs(target\_folder)

with webdriver.Chrome(executable\_path=driver\_path) as wd:

res = fetch\_image\_urls(search\_term, number\_images, wd=wd, sleep\_between\_interactions=0.5)

for elem in res:

persist\_image(target\_folder, elem, img\_name)

def fetch\_image\_urls(query:str, max\_links\_to\_fetch:int, wd:webdriver, sleep\_between\_interactions:int):

def scroll\_to\_end(wd):

wd.execute\_script("window.scrollTo(0, document.body.scrollHeight);")

time.sleep(sleep\_between\_interactions)

search\_url = "https://www.google.com/search?safe=off&site=&tbm=isch&source=hp&q={q}&oq={q}&gs\_l=img"

wd.get(search\_url.format(q=query))

image\_urls = set()

image\_count = 0

results\_start = 0

while image\_count < max\_links\_to\_fetch:

scroll\_to\_end(wd)

thumbnail\_results = wd.find\_elements\_by\_css\_selector("img.Q4LuWd")

number\_results = len(thumbnail\_results)

print(f"Found: {number\_results} search results. Extracting links from {results\_start}:{number\_results}")

for img in thumbnail\_results[results\_start:number\_results]:

try:

img.click()

time.sleep(sleep\_between\_interactions)

except Exception:

continue

actual\_images = wd.find\_elements\_by\_css\_selector('img.n3VNCb')

for actual\_image in actual\_images:

if actual\_image.get\_attribute('src') and 'http' in actual\_image.get\_attribute('src'):

image\_urls.add(actual\_image.get\_attribute('src'))

else:

data = actual\_image.get\_attribute('src').replace(' ', '+')

data = data[data.find(",") + 1:]

image\_urls.add(data)

image\_count = len(image\_urls)

if len(image\_urls) >= max\_links\_to\_fetch:

break

else:

time.sleep(30)

return

load\_more\_button = wd.find\_element\_by\_css\_selector(".mye4qd")

if load\_more\_button:

wd.execute\_script("document.querySelector('.mye4qd').click();")

results\_start = len(thumbnail\_results)

return image\_urls

def persist\_image(folder\_path:str,url:str, img\_name:str):

try:

image\_content = requests.get(url).content

except Exception as e:

print(f"ERROR - Could not download {url} - {e}")

try:

if 'http' in url:

image\_file = io.BytesIO(image\_content)

image = Image.open(image\_file).convert('RGB')

file\_path = os.path.join(folder\_path, img\_name + '.jpg')

with open(file\_path, 'wb') as f:

f.save(image, "JPEG", quality=85)

elif 'https' in url:

image\_file = io.BytesIO(image\_content)

image = Image.open(image\_file).convert('RGB')

file\_path = os.path.join(folder\_path, img\_name + '.jpg')

with open(file\_path, 'wb') as f:

f.save(image, "JPEG", quality=85)

else:

image = io.BytesIO(base64.b64decode(url))

image = Image.open(image).convert('RGB')

file\_path = os.path.join(folder\_path, img\_name + '.jpg')

image.save(file\_path, "JPEG", quality=85)

except Exception as e:

print(f"ERROR - Could not save {url} - {e}")

image = Image.open("../data/inputs/test.jpg")

file\_path = os.path.join(folder\_path, img\_name + '.jpg')

image.save(file\_path, "JPEG", quality=85)

if int(img\_name) == len(inputs):

video\_maker(timeArray)

**Video making**

* Load the libraries

from conf import SAMPLE\_INPUTS, SAMPLE\_OUTPUTS

from moviepy.editor import \*

from PIL import Image

* Making video with images and adding audio

from resizeimage import resizeimage

images = os.path.join(SAMPLE\_INPUTS, 'images')

output\_video = os.path.join(SAMPLE\_OUTPUTS, 'video.mp4')

final\_video = os.path.join(SAMPLE\_OUTPUTS, 'final\_video.mp4')

source\_audio\_path = os.path.join(SAMPLE\_INPUTS, 'test.mp3')

def video\_maker(time):

directory = {}

for root, dirs, files in os.walk(images):

for fname in files:

filepath = os.path.join(root, fname)

try:

key = float(fname.replace(".jpg", ""))

except:

key = None

if key != None:

directory[key] = filepath

img = Image.open(filepath)

newsize = (700, 500)

img = img.resize(newsize)

with open(filepath, 'wb') as f:

img.save(f, "JPEG", quality=85)

new\_paths = []

for k in sorted(directory.keys()):

filepath = directory[k]

new\_paths.append(filepath)

my\_clips = []

for index,path in enumerate(list(new\_paths)):

frame = ImageClip(path)

my\_clips.append({'img':frame.img,'duration':time[index]})

clips = [ImageClip(m['img']).set\_duration(m['duration'])

for m in my\_clips]

concat\_clip = concatenate\_videoclips(clips, method="compose")

concat\_clip.write\_videofile(output\_video, fps=24)

background\_audio\_clip = AudioFileClip(source\_audio\_path)

video\_clip = VideoFileClip(output\_video).cutout(background\_audio\_clip.duration, VideoFileClip(output\_video).duration)

bg\_music = background\_audio\_clip.subclip(0, video\_clip.duration)

final\_clip = video\_clip.set\_audio(bg\_music)

final\_clip.write\_videofile(final\_video, codec='libx264', audio\_codec='aac')

## 

## Predicting the similar sentence and finding the most appropriate answer

**Predicting the similar sentence**

**Create Features**

* Load the libraries

import warnings

warnings.filterwarnings('ignore')

import pickle

import numpy as np

import pandas as pd

from textblob import TextBlob

from scipy import spatial

import torch

from models import InferSent

* Load data and create vector representation using Infersent

train = pd.read\_csv(r'C:\Users\haran\Desktop\fyp\_project\new\_sentence\data\train\_custom2.csv')

# valid = pd.read\_json("drive/MyDrive/interim/data/dev-v1.1.json")

print(train.shape)

df = train

print(df.columns)

print("shape of initial dataset = " + str(df.shape))

# Create dictionary of sentence embeddings for faster computation

paras = list(df["context"].drop\_duplicates().reset\_index(drop=True))

print("length of paragraphs = " + str(len(paras)))

blob = TextBlob(" ".join(paras))

sentences = [item.raw for item in blob.sentences]

print("length of sentences = "+str(len(sentences)))

V = 1

MODEL\_PATH = r'C:\Users\haran\Desktop\fyp\_project\new\_sentence\InferSent\encoder\infersent%s.pkl' %V

params\_model = {'bsize': 64, 'word\_emb\_dim': 300, 'enc\_lstm\_dim': 2048,

'pool\_type': 'max', 'dpout\_model': 0.0, 'version': V}

infersent = InferSent(params\_model)

infersent.load\_state\_dict(torch.load(MODEL\_PATH))

W2V\_PATH = r'C:\Users\haran\Desktop\fyp\_project\new\_sentence\InferSent\dataset\Glove\glove.840B.300d.txt'

infersent.set\_w2v\_path(W2V\_PATH)

infersent.build\_vocab(sentences, tokenize=True)

dict\_embeddings = {}

for i in range(len(sentences)):

print(i)

dict\_embeddings[sentences[i]] = infersent.encode([sentences[i]], tokenize=True)

questions = list(df["question"])

len(questions)

for i in range(len(questions)):

print(i)

dict\_embeddings[questions[i]] = infersent.encode([questions[i]], tokenize=True)

d3 = {key:dict\_embeddings[key] for i, key in enumerate(dict\_embeddings) if i % 2 == 0}

d4 = {key:dict\_embeddings[key] for i, key in enumerate(dict\_embeddings) if i % 2 == 1}

with open(r'C:\Users\haran\Desktop\fyp\_project\new\_sentence\data\dict\_embeddings\dict\_embeddings\_custom1.pickle', 'wb') as handle:

pickle.dump(d3, handle)

with open(r'C:\Users\haran\Desktop\fyp\_project\new\_sentence\data\dict\_embeddings\dict\_embeddings\_custom2.pickle', 'wb') as handle:

pickle.dump(d4, handle)

with open(r"C:\Users\haran\Desktop\fyp\_project\new\_sentence\data\dict\_embeddings\dict\_embeddings1.pickle", "rb") as f:

d1 = pickle.load(f)

with open(r"C:\Users\haran\Desktop\fyp\_project\new\_sentence\data\dict\_embeddings\dict\_embeddings2.pickle", "rb") as f:

d2 = pickle.load(f)

with open(r"C:\Users\haran\Desktop\fyp\_project\new\_sentence\data\dict\_embeddings\dict\_embeddings\_custom1.pickle", "rb") as f:

d3 = pickle.load(f)

with open(r"C:\Users\haran\Desktop\fyp\_project\new\_sentence\data\dict\_embeddings\dict\_embeddings\_custom2.pickle", "rb") as f:

d4 = pickle.load(f)

dict\_emb = dict(d1)

dict\_emb.update(d2)

dict\_emb.update(d3)

dict\_emb.update(d4)

print(len(dict\_emb))

* Save vector representation of sentences and questions

train = pd.read\_csv(r'C:\Users\haran\Desktop\fyp\_project\new\_sentence\data\train2.csv')

* Create cosine similarity between sentence question pair

def cosine\_sim(x):

li = []

for item in x["sent\_emb"]:

li.append(spatial.distance.cosine(item, x["quest\_emb"][0]))

return li

* Create euclidean distance between sentences

def pred\_idx(distances):

return np.argmin(distances)

* Predict cosine and euclidean similarity between sentence question pair

def predictions(train):

train["cosine\_sim"] = train.apply(cosine\_sim, axis=1)

train["diff"] = (train["quest\_emb"] - train["sent\_emb"]) \*\* 2

train["euclidean\_dis"] = train["diff"].apply(lambda x: list(np.sum(x, axis=1)))

del train["diff"]

print("cosine start")

train["pred\_idx\_cos"] = train["cosine\_sim"].apply(lambda x: pred\_idx(x))

train["pred\_idx\_euc"] = train["euclidean\_dis"].apply(lambda x: pred\_idx(x))

return train

* Get target sentence which contains answer

def get\_target(x):

idx = -1

for i in range(len(x["sentences"])):

if x["text"] in x["sentences"][i]:

idx = i

return idx

* Find accuracy using unsupervised techniques

def accuracy(target, predicted):

acc = (target == predicted).sum() / len(target)

return acc

**Create additional features**

* load libraries

import warnings

warnings.filterwarnings('ignore')

import pickle

import pandas as pd

import ast

from sklearn.preprocessing import MinMaxScaler

from sklearn.model\_selection import train\_test\_split

from sklearn import linear\_model

from sklearn import metrics

import spacy

en\_nlp = spacy.load('en\_core\_web\_sm')

from nltk import Tree

from nltk.stem.lancaster import LancasterStemmer

st = LancasterStemmer()

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.ensemble import RandomForestClassifier

import xgboost as xgb

from sklearn.model\_selection import GridSearchCV

* load dataset and create features cosine difference between every sentence question pair

def create\_features(data):

train = pd.DataFrame()

for k in range(len(data["euclidean\_dis"])):

dis = ast.literal\_eval(data["euclidean\_dis"][k])

for i in range(len(dis)):

train.loc[k, "column\_euc\_" + "%s" % i] = dis[i]

print("Finished")

for k in range(len(data["cosine\_sim"])):

dis = ast.literal\_eval(data["cosine\_sim"][k].replace("nan", "1"))

for i in range(len(dis)):

train.loc[k, "column\_cos\_" + "%s" % i] = dis[i]

train["target"] = data["target"]

return train

* Train data considering cosine and euclidean features using multiclass logistic regression

train = create\_features(data)

print(train.head(3))

print(train.head(3).transpose())

#multiclass logistic regression

train.apply(max, axis=0)

subset1 = train.iloc[:, :10].fillna(60)

subset2 = train.iloc[:, 10:].fillna(1)

print(subset 1.head(3))

print(subset2.head(3))

train2 = pd.concat([subset1, subset2], axis=1)

train2=train 2.reindex(subset 1.index)

print(train 2.head(3))

train 2.apply(max, axis=0)

scaler = MinMaxScaler()

X = scaler.fit\_transform(train2.iloc[:,:-1])

print(X)

train\_x, test\_x, train\_y, test\_y = train\_test\_split(X, train.iloc[:, -1], train\_size=0.8, random\_state=5)

mul\_lr = linear\_model.LogisticRegression(multi\_class='multinomial', solver='newton-cg')

mul\_lr.fit(train\_x, train\_y)

print("Multinomial Logistic regression Train Accuracy : ", metrics.accuracy\_score(train\_y, mul\_lr.predict(train\_x)))

print("Multinomial Logistic regression Test Accuracy : ", metrics.accuracy\_score(test\_y, mul\_lr.predict(test\_x)))

* Create parse tree for sentences and question

predicted = pd.read\_csv(r"C:\Users\haran\Desktop\fyp\_project\new\_sentence\data\train\_detect\_sent.csv").reset\_index(drop=True)

doc = en\_nlp(predicted.iloc[0, 1])

print(predicted.iloc[0, 1])

print(predicted.iloc[0, 2])

def to\_nltk\_tree(node):

if node.n\_lefts + node.n\_rights > 0:

return Tree(node.orth\_, [to\_nltk\_tree(child) for child in node.children])

else:

return node.orth\_

[to\_nltk\_tree(sent.root).pretty\_print() for sent in en\_nlp(predicted.iloc[0, 2]).sents]

print([to\_nltk\_tree(sent.root) .pretty\_print() for sent in doc.sents][5])

for sent in doc.sents:

roots = [st.stem(chunk.root.head.text.lower()) for chunk in sent.noun\_chunks]

print(roots)

* Create feature based on root match feature

def match\_roots(x):

question = x["question"].lower()

sentences = en\_nlp(x["context"].lower()).sents

question\_root = st.stem(str([sent.root for sent in en\_nlp(question).sents][0]))

li = []

for i, sent in enumerate(sentences):

roots = [st.stem(chunk.root.head.text.lower()) for chunk in sent.noun\_chunks]

if question\_root in roots:

for k, item in enumerate(ast.literal\_eval(x["sentences"])):

if str(sent) in item.lower():

li.append(k)

return li

* Create features root\_match\_idx , root\_match\_idx\_first

question = predicted["question"][0].lower()

sentences = en\_nlp(predicted["context"][0].lower()).sents

question\_root = st.stem(str([sent.root for sent in en\_nlp(question).sents][0]))

li = []

for i, sent in enumerate(sentences):

roots = [st.stem(chunk.root.head.text.lower()) for chunk in sent.noun\_chunks]

print(roots)

if question\_root in roots: li.append(i)

ast.literal\_eval(predicted["sentences"][0])

print(predicted["context"][0])

sentences = en\_nlp(predicted["context"][0].lower()).sents

for item in sentences:

print(item)

TfidfVectorizer(predicted["sentences"][0], ngram\_range=(1, 2))

# multiclass logistic regression with root train

predicted = pd.read\_csv(r"C:\Users\haran\Desktop\fyp\_project\new\_sentence\data\train\_detect\_sent.csv").reset\_index(drop=True)

predicted = predicted[predicted["sentences"].apply(lambda x: len(ast.literal\_eval(x))) < 11].reset\_index(drop=True)

print(predicted.shape)

* Apply root\_match feature for whole dataset

def get\_columns\_from\_root(train):

for i in range(train.shape[0]):

if len(ast.literal\_eval(train["root\_match\_idx"][i])) == 0:

pass

else:

for item in ast.literal\_eval(train["root\_match\_idx"][i]):

train.loc[i, "column\_root\_" + "%s" % item] = 1

return train

* Create features root\_match\_idx , root\_match\_idx\_first

predicted = get\_columns\_from\_root(predicted)

predicted.head(7).transpose()

subset3 = predicted[["column\_root\_0", "column\_root\_1", "column\_root\_2", "column\_root\_3", "column\_root\_4", "column\_root\_5",\"column\_root\_6", "column\_root\_7", "column\_root\_8", "column\_root\_9"]]

subset 3.fillna(0, inplace=True)

train3 = pd.concat([subset3, train2], axis=1)

train3 = train 3.reindex(subset 3.index)

subset 3.fillna(0, inplace=True)

train 3.head(3).transpose()

train3 = train3[["column\_root\_0", "column\_root\_1", "column\_root\_2", "column\_root\_3", "column\_root\_4", "column\_root\_5",

"column\_root\_6", "column\_root\_7", "column\_root\_8", "column\_root\_9", "column\_cos\_0", "column\_cos\_1",

"column\_cos\_2", "column\_cos\_3", "column\_cos\_4", "column\_cos\_5",

"column\_cos\_6", "column\_cos\_7", "column\_cos\_8", "column\_cos\_9", “target"]]

train\_x, test\_x, train\_y, test\_y = train\_test\_split(train3.iloc[:, :-1],

train3.iloc[:, -1], train\_size=0.8, random\_state=5)

* Train dataset using multinomial logistic regression

mul\_lr = linear\_model.LogisticRegression(multi\_class='multinomial', solver='newton-cg')

mul\_lr.fit(train\_x, train\_y)

with open(r'C:\Users\haran\Desktop\fyp\_project\new\_sentence\data\supervised\_predicted\_models\logistic\_regression.pickle', 'wb') as handle:

pickle.dump(mul\_lr, handle)

print("Multinomial Logistic regression Train Accuracy : ", metrics.accuracy\_score(train\_y, mul\_lr.predict(train\_x)))

print("Multinomial Logistic regression Test Accuracy : ", metrics.accuracy\_score(test\_y, mul\_lr.predict(test\_x)))

print(metrics.confusion\_matrix(test\_y, mul\_lr.predict(test\_x)))

print(metrics.classification\_report(test\_y, mul\_lr.predict(test\_x), digits=3))

* Train dataset using random forest

rf = RandomForestClassifier(min\_samples\_leaf=8, n\_estimators=60)

rf.fit(train\_x, train\_y)

print("Random forest Train Accuracy : ", metrics.accuracy\_score(train\_y, rf.predict(train\_x)))

print("Random Forest Test Accuracy : ", metrics.accuracy\_score(test\_y, rf.predict(test\_x)))

print(metrics.confusion\_matrix(test\_y, rf.predict(test\_x)))

print(metrics.classification\_report(test\_y, rf.predict(test\_x), digits=3))

with open(r'C:\Users\haran\Desktop\fyp\_project\new\_sentence\data\supervised\_predicted\_models\random\_forest.pickle', 'wb') as handle:

pickle.dump(rf, handle)

* Train dataset using XGboost classifier

model = xgb.XGBClassifier()

param\_dist = {"max\_depth": [3, 5, 10],

"min\_child\_weight": [1, 5, 10],

"learning\_rate": [0.07, 0.1, 0.2]

}

# run randomized search

grid\_search = GridSearchCV(model, param\_grid=param\_dist, cv=3,

verbose=5, n\_jobs=-1)

grid\_search.fit(train\_x, train\_y)

print(grid\_search.best\_estimator\_)

xg = xgb.XGBClassifier(max\_depth=5)

xg.fit(train\_x, train\_y)

with open(r'C:\Users\haran\Desktop\fyp\_project\new\_sentence\data\supervised\_predicted\_models\xgboost.pickle', 'wb') as handle:

pickle.dump(xg, handle)

print(metrics.classification\_report(test\_y, xg.predict(test\_x), digits=3))

print("XGBoost Train Accuracy : ", metrics.accuracy\_score(train\_y, xg.predict(train\_x)))

print("XGBoost Test Accuracy : ", metrics.accuracy\_score(test\_y, xg.predict(test\_x)))

print(metrics.confusion\_matrix(test\_y, xg.predict(test\_x)))

**finding the most appropriate answer**

* Import Libraries

import json

from transformers import DistilBertTokenizerFast

import torch

from torch.utils.data import DataLoader

from transformers import AdamW

from tqdm import tqdm

from transformers import DistilBertForQuestionAnswering

* Data Preprocessing

def read\_dataset(path):

contexts = []

questions = []

answers = []

for group in squad\_dict['data']:

for passage in group['paragraphs']:

context = passage['context']

for qa in passage['qas']:

question = qa['question']

access = 'answers'

for answer in qa[access]:

contexts.append(context)

questions.append(question)

answers.append(answer)

return contexts, questions, answers

* Create feature start and end index of answer

def add\_end\_idx(answers, contexts):

for answer, context in zip(answers, contexts):

gold\_text = answer['text']

start\_idx = answer['answer\_start']

end\_idx = start\_idx + len(gold\_text)

if context[start\_idx:end\_idx] == gold\_text:

answer['answer\_end'] = end\_idx

else:

for n in [1, 2]:

if context[start\_idx-n:end\_idx-n] == gold\_text:

answer['answer\_start'] = start\_idx - n

answer['answer\_end'] = end\_idx - n

add\_end\_idx(train\_answers, train\_contexts)

add\_end\_idx(val\_answers, val\_contexts)

tokenizer = DistilBertTokenizerFast.from\_pretrained('distilbert-base-uncased')

train\_encodings = tokenizer(train\_contexts, train\_questions, truncation=True, padding=True)

val\_encodings = tokenizer(val\_contexts, val\_questions, truncation=True, padding=True)

train\_encodings.keys()

* Tokenization paragraph and answer

def add\_token\_positions(encodings, answers):

start\_positions = []

end\_positions = []

for i in range(len(answers)):

start\_positions.append(encodings.char\_to\_token(0,answers[0]['answer\_start']))

end\_positions.append(encodings.char\_to\_token(0,answers[0]['answer\_end']))

if start\_positions[-1] is None:

start\_positions[-1] = tokenizer.model\_max\_length

go\_back = 1

while end\_positions[-1] is None:

end\_positions[-1] = encodings.char\_to\_token(0,train\_answers[0]['answer\_end']-go\_back)

go\_back += 1

encodings.update({

'start\_positions': start\_positions,

'end\_positions': end\_positions

})

add\_token\_positions(train\_encodings, train\_answers)

add\_token\_positions(val\_encodings, val\_answers)

train\_encodings.keys()

* Load Model

class QADataset(torch.utils.data.Dataset):

def \_\_init\_\_(self,encodings):

self.encodings = encodings

def \_\_getitem\_\_(self, idx):

return {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}

def \_\_len\_\_(self):

return len(self.encodings.input\_ids)

train\_dataset = QADataset(train\_encodings)

val\_dataset = QADataset(val\_encodings)

# Load already available model

model = DistilBertForQuestionAnswering.from\_pretrained('distilbert-base-uncased')

device = torch.device('cuda') if torch.cuda.is\_available() else torch.device('cpu')

model.to(device)

model.train()

optim = AdamW(model.parameters(), lr=5e-5)

train\_loader = DataLoader(train\_dataset, batch\_size=16, shuffle=True)

train\_encodings.keys()

print(tokenizer.decode(train\_encodings['input\_ids'][0]))

print("started training")

* pretrain Model

for epoch in range(2):

loop = tqdm(train\_loader)

for batch in loop:

optim.zero\_grad()

input\_ids = batch['input\_ids'].to(device)

attention\_mask = batch['attention\_mask'].to(device)

start\_positions = batch['start\_positions'].to(device)

end\_positions = batch['end\_positions'].to(device)

outputs = model(input\_ids, attention\_mask=attention\_mask,

start\_positions=start\_positions, end\_positions=end\_positions)

loss = outputs[0]

loss.backward()

optim.step()

loop.set\_description(Epoch {epoch}')

loop.set\_postfix(loss=loss.item())

# save model

model.save\_pretrained(r"C:\Users\haran\Desktop\fyp\_project\new\_sentence\data\finetune-model")

tokenizer.save\_pretrained(r"C:\Users\haran\Desktop\fyp\_project\new\_sentence\data\finetune-model")

* Evaluating pretrained model accuracy

val\_loader = DataLoader(val\_dataset, batch\_size=16)

preTrain\_acc = []

loop = tqdm(val\_loader)

for batch in loop:

with torch.no\_grad():

input\_ids = batch['input\_ids'].to(device)

attention\_mask = batch['attention\_mask'].to(device)

start\_true = batch['start\_positions'].to(device)

end\_true = batch['end\_positions'].to(device)

outputs = model(input\_ids, attention\_mask=attention\_mask)

start\_pred = torch.argmax(outputs['start\_logits'], dim=1)

end\_pred = torch.argmax(outputs['end\_logits'], dim=1)

preTrain\_acc.append(((start\_pred == start\_true).sum() / len(start\_pred)).item())

preTrain\_acc.append(((end\_pred == end\_true).sum() / len(end\_pred)).item())

print("pretrained\_accuracy : " + str(sum(preTrain\_acc) / len(preTrain\_acc)))

**Flask backend for testing application**

* Import Libraries

import time

import numpy as np

from flask import Flask, request, jsonify, render\_template

import pickle

import warnings

warnings.filterwarnings('ignore')

import pickle

import numpy as np

import pandas as pd

# import json

from textblob import TextBlob

from scipy import spatial

import torch

import csv

from transformers import AutoTokenizer

from transformers import pipeline

from transformers import BertForQuestionAnswering

import ast

* Rest endpoints

@app.route('/')

def home():

return render\_template('index.html')

@app.route('/predict', methods=['POST'])

def predict():

'''

For rendering results on HTML GUI

'''

# print(request.form.values())

sentences = []

context = request.values.get('context')

question = request.values.get('question')

unsupervised = write\_csv(context, question)

# unsupervised = pd.read\_csv(r"C:\Users\haran\Desktop\fyp\_project\new\_sentence\data\unsupervised.csv")

df.to\_csv(r"C:\Users\haran\Desktop\fyp\_project\new\_sentence\data\unsupervised.csv", index=None)

predicted = process\_data(unsupervised)

print(predicted["cosine\_sim"][0])

print(predicted["euclidean\_dis"][0])

print(predicted["pred\_idx\_euc"][0])

print(predicted["pred\_idx\_cos"][0])

prediction\_text = str(predicted["pred\_idx\_cos"][0])

textblob\_object = TextBlob(context)

sentences = textblob\_object.sentences

print(sentences)

sentence = sentences[predicted["pred\_idx\_euc"][0]]

print(sentence)

return render\_template('sentence.html', prediction\_text=prediction\_text, context=context, question=question,

sentence=sentence)

@app.route('/answer', methods=['POST'])

def answer():

'''

For rendering results on HTML GUI

'''

context = request.values.get('context')

question = request.values.get('question')

model = BertForQuestionAnswering.from\_pretrained(r"C:\Users\haran\Desktop\fyp\_project\new\_sentence\data\finetune-model")

tokenizer = AutoTokenizer.from\_pretrained(r"C:\Users\haran\Desktop\fyp\_project\new\_sentence\data\finetune-model")

tokenizer.encode(question, truncation=True, padding=True)

nlp = pipeline('question-answering', model=model, tokenizer=tokenizer)

answer\_body = nlp({

'question': question,

'context': context

})

prediction\_text = answer\_body['answer']

return render\_template('answer.html', prediction\_text=prediction\_text, context=context, question=question)