PES
UNIVERSITY

Dissertation on

"Text summarization for News"

Submitted in partial fulfilment of the requirements for the award of degree of

Bachelor of Technology

in

Computer Science & Engineering

Submitted by:

Aditya Jain	01FB15ECS018
Akansha V Magod	01FB15ECS024
Kartikeya Agarwal	01FB15ECS143

Under the guidance of

Internal Guide

Prof. S Nagasundari

Associate Professor

January – May 2019

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

FACULTY OF ENGINEERING

PES UNIVERSITY

(Established under Karnataka Act No. 16 of2013)

100ft Ring Road, Bengaluru – 560 085, Karnataka, India

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

100ft Ring Road, Bengaluru – 560 085, Karnataka, India

FACULTY OF ENGINEERING

CERTIFICATE

This is to certify that the dissertation entitled

'Text summarization for news'

is a bonafide work carried out by

Aditya Jain	01FB15ECS018
Akansha V Magod	01FB15ECS024
Kartikeya Agarwal	01FB15ECS143

In partial fulfilment for the completion of eighth semester project work in the Program of Study Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period Jan. 2019 – May. 2019. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The dissertation has been approved as it satisfies the 8th semester academic requirements in respect of projectwork.

Signature	Signature	Signature
Prof. S Nagasundari	Dr.Shylaja S S	Dr.B.K.Keshavan
Associate Professor	Chairperson	Dean of Faculty

External Viva

Name of the Examiners	**Signature with Date**
1.________________________	________________________
2.________________________	________________________

DECLARATION

I hereby declare that the project entitled "**Text Summarizer for News**" has been carried out by us under the guidance of Prof. S Nagasundari, Associate Professor and submitted in partial fulfilment of the course requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester January – May 2019. The matter embodied in this report has not been submitted to any other university or institution for the award of any degree.

Aditya Jain	01FB15ECS018
Akansha V Magod	01FB15ECS024
Kartikeya Agarwal	01FB15ECS143

ACKNOWLEDGEMENT

I would like to express my gratitude to Prof. S Nagasundari, Dept. of Computer Science, PES University, for her continuous guidance, assistance and encouragement throughout the development of this project.

I am grateful to the project coordinators, Prof. Keerthana and Prof. Jamuna for organizing, managing and helping out with the entire process.

I take this opportunity to thank Dr Shylaja S S, Chairperson, Department of Computer Science and Engineering, PES University, for all the knowledge and support I have received from the department. I would like to thank Dr B.K. Keshavan, Dean of Faculty, PES University for his help.

I am deeply grateful to Dr M. R. Doreswamy, Chancellor, PES University, Prof. Jawahar Doreswamy, Pro-Chancellor – PES University, Dr K.N.B. Murthy, Vice-Chancellor, PES University for providing to me various opportunities and enlightenment every step of the way. Finally, this project could not have been completed without the continual support and encouragement I have received from my parents.

ABSTRACT

With the rapid growth of broadcast systems, the internet and online information services, more and more information is available and accessible. There is no time to read everything, and yet we have to keep ourselves updated based on whatever information is available.

Being able to develop models that can deliver accurate summaries of longer text will be useful. This will help us to digest a large amount of information in a compressed form. Thus, given a busy schedule of people and an immense amount of information available on the Internet, there is a need for automatic summarization of links based on user query.

The product is a web-based and can be hosted on the internet. It provides a clean and user-friendly interface to the users. The user can enter the URL of the article for which he wants a summarized article. Hence, it provides the user with a more friendly way of doing text summarization.

TABLE OF CONTENTS

CHAPTER-1

INTRODUCTION

Text summarization as the name suggests refers to creating short pieces of important text from the original text, in such a way that the number of words is reduced from the produced summarized article.This should be done in such a way that the summarized article is able to convey and pass the message which the original article was doing.

Humans have well-equipped brains and they have enough knowledge, hence they are very good at doing this task of text summarization. They can easily understand the meaning of the article and can easily extract the salient features of the article to summarize the document.

Automatic text summarization refers to doing this job of text summarization with the help of some software. These softwares should be designed in such that they are able to capture the important points which the article is trying to convey and should present the user the same message which the article was trying to convey and pass. These softwares should take into account the written style of the language, syntax, length etc.

The reasons why we say automatic text summarization is useful are:

1. These generated summaries reduces the reading time. This is especially the case with the news article because the reader wants himself updated on the current affairs, while on the other hand, he may not have time to spend reading the entire news. They just want to read the important points
2. While researching the documents, it makes the selection process easier as to which documents or articles are of important use to them. Researchers may save a lot of time doing a literature survey as they don’t need to read the entire document or article.
3. This technology can also improve the effectiveness of indexing. Google uses its page rank algorithm (which is an important algorithm for text summarization) to index its search results.

4. The summaries generated are not biased and sometimes may pass correct information which humans may fail to do.
5. With data being the new oil and with such big amount of data being circulated and generated every day, there is always a need to develop machine learning algorithm that can automatically shorten long articles and deliver accurate and short summaries which is easy to store and can be easily interpreted.

Automatic text summarization is a very old problem. It was known to have existed for 50 years. This problem is a very common area of research in machine learning and natural language processing(NLP). This problem is still very wide open as no proper methods which work well are known to have existed.

1.1 Types of text summarization.

Text summarization methods are basically of different types :

1. **Based on the input type**
   a. **A single document**, where the input length of the text is very less. Earlier approaches dealt with this kind of single document text summarization.
   b. **Multiple documents,** where the input length of the text is very large. It can be arbitrarily very long. Recent approaches dealt with this kind of text summarization.
2. **Based on the purpose**
   a. **Generic**. In this, the model and the software makes no assumption on the domain and the content of the given text. It can be either news, literature etc. So this treats all inputs as homogenous. Current approaches deal with this kind of input.
   b. **Domain-specific.** In this, the model knows beforehand as to what is the domain of the input. This helps to generate more accurate summaries. For example news, literature etc.
3. **Based on the output type**

a. Extractive summarization

An intuitive explanation for this can be thought of as taking a highlighter and highlighting the important sentences with a colour of your choice. So we have an input and we mark the important sentences which are of relevance to me. There may be some metric involved to mark these important sentences. After marking the relevant sentences these sentences are joined to make a summarized article.

An example of this is shown below:

Original Text: "**Alice and Bob** took the train to **visit the zoo**. They **saw a** baby giraffe, a lion, and a **flock of** colourful tropical **birds**."

Extractive Summary: "Alice and Bob visit the zoo. saw a flock of birds"

We can see that in the above important words are marked in bold and they are combined to form a summarized article. We can easily observe that sometimes these extractive content can make our summarized text look grammatically awkward. So we can say that this is a constraint involved with extractive text summarization.

b. Abstractive summarization

An intuitive explanation for this can be thought of as reading the entire text and generating the summary based on that. We, humans, do this in the same way. So this involves paraphrasing the section of the original text and input. This suggests that abstractive text summarization condenses the text more strongly and in a very appropriate manner.

Original Text: "**Alice** and **Bob** took the train to visit the zoo. They saw a baby giraffe, a lion, and a flock of colourful tropical birds."

Abstractive Summary: "Alice and Bob visited the zoo and saw animals and birds."

As we can see that it generates the summary in a more condensed form since it does not uses words which are these in the original text. It modified it to and made it look more natural and abstract. This is also grammatically correct.

On having a look at both of these it is very much clear that abstractive is the way to go for a better summarization output. But this has some major shortcomings:

1. It is very hard to implement. It may seem easy but this will break your head.
2. This is computationally very expensive. So this may be suited for low-end devices and mobile devices.
3. This needs lots of resource and time.
4. Accuracy may not be higher when compared with extractive text summarization.

Initially, we focused on abstractive text summarization but later following some suggestions we switched to extractive text summarization. We came up with a method which is not researched and worked on. Also, our model is computationally inexpensive and therefore can be run on low-end devices.

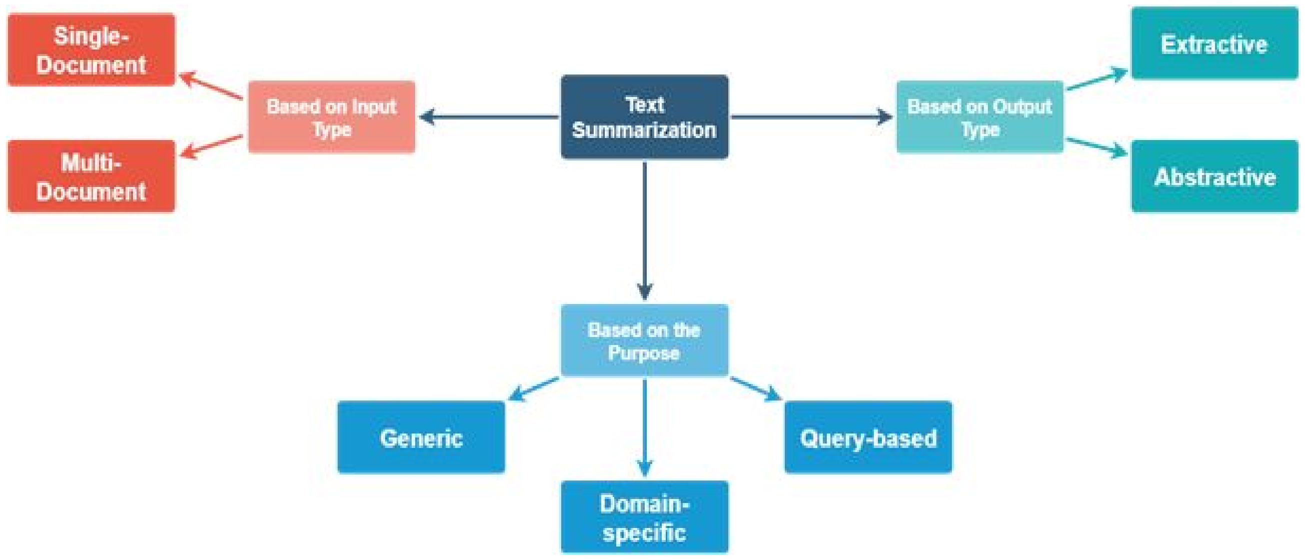


Figure: Flow chart representing text summarization methods.

CHAPTER-2

PROBLEM DEFINITION

With data being the new oil and with such humongous amount of data being circulated and generated every day, there is always a need to develop machine learning algorithm that can automatically shorten long articles and deliver accurate and short summaries which is easy to store and can be easily interpreted.

This is especially the case with the news article because the reader wants himself updated on the current affairs, while on the other hand, he may not have time to spend reading the entire news. They just want to read the important points. What if a model or a software is developed which can automatically condense the text to produce the appropriate summary.

Thus given a busy schedule of working professional and with such humongous amount of data being circulated and generated every day, it is always a profit if we create a system which gives us a condensed summarized article based on the user query.

Hence after creating the model we also need to add an interactive user interface, which lets the user get a summarized article based on the URL of the article.

The developed product is a web-based product and can be hosted on the internet, providing the user with a clean and interactive user interface.

CHAPTER-3

LITERATURE SURVEY

Our entire project earlier was based on abstractive text summarization. We explored many papers and got an idea as to what research has been done in the field of text summarization. Our current project deals with extractive text summarization which was decided because of the lack of resources which it requires in terms of computation power and time. The papers referred used recurrent neural network and LSTM networks at its core. So we started with exploring more on recurrent neural networks and LSTM network(long short term memory). After giving an explanation we will dive deep into research papers.

3.1 Explanation of RNN's

As humans, we are able to understand the meaning of a sentence based on the meaning of previous sentences and the context in which that sentence appears. Similarly, machines understand or predict a word based on their understanding of previous words and the context in which it appears, However, traditional neural networks lack this sense of persistence. The recurrent neural network is the first of its kind algorithm which can remember its output because of internal memory. They also use loops to persist the information they have learnt. So technically they are not much different from traditional neural networks, they are just multiple copies of the same network

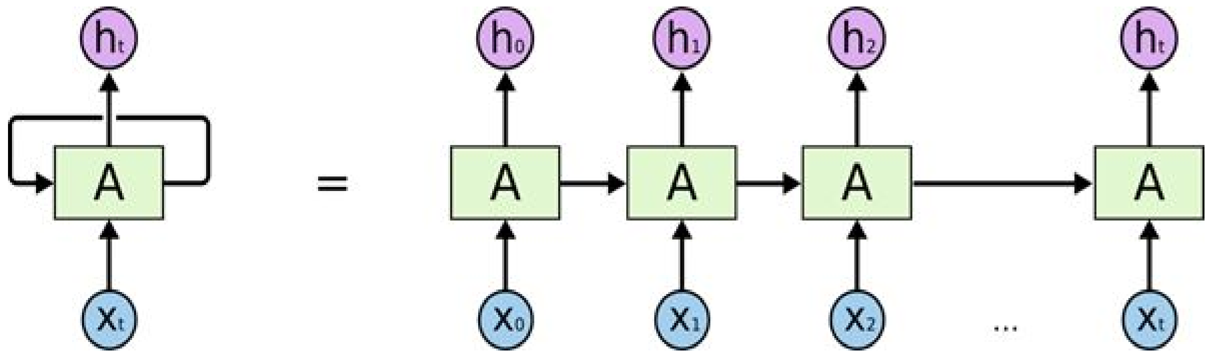


Figure: Recurrent neural networks as the normal feed-forward neural network.

One issue with recurrent neural networks is that they can not persist the information for a long period. For our case of text summarization when the gap between the previous information and current information is very large then it can be used directly. So if the user provides with a large piece of text, then it may fail to learn. Hence it is necessary to add some additional unit which can store information

3.2 LSTM networks

This is used to overcome the shortcomings of RNN's. Remembering information for long periods of time is practically their default behaviour, not something they struggle to learn.
So the LSTM can be called a special case of RNN's which are capable of learning long term and short term dependency (memory).

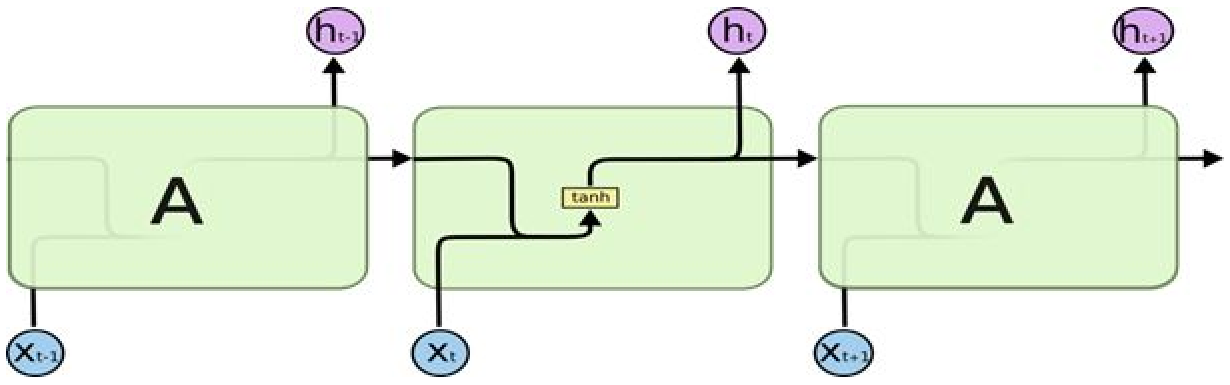


Figure: Architecture of standard RNNs without LSTM

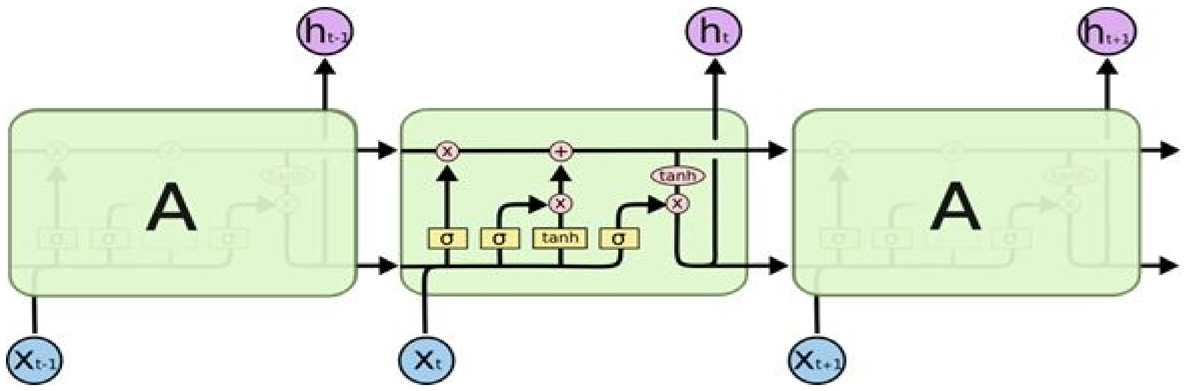


Figure: Repeating module in an LSTM containing four interacting layer

LSTM permits or disallows information to cell state, with the help of gates. They contain a sigmoid neural network layer and a pointwise multiplication operation.

3.3 Text summarization as a classification

Traditional rule-based neural network approaches on abstractive text summarization performed poorly. Some researchers have utilized the advantages of neural network learning capabilities to learn summary sentence attributes. A neural network is trained to learn the relevant features of sentences that should be included in the summary of the article. The neural network is then modified to generalize and combine the relevant features apparent in summary sentences. Finally, the modified neural network is used as a filter to summarize news articles. [1] used this approach There are also other machine learning methods used for text summarization. [2] Proposed a method in which they first classified important sentences. They used the decision tree model to train their classifier. They were able to outperform previous approach results.

3.4 Domain specific summarization approaches

There have been different proposed methods specific to the domain. These include domains specific to news summarization, document summarization, email/blog summarization. These methods include statistical classification (Naïve Bayes theorem etc.). Early work on news summarization includes a system named SUMMONS [3] summarizer. It was built using a template-driven message understanding system. Similar to this there are system [4, 5] named Riptides, newsblaster etc. These methods used information retrieval technology to generate a summarized text.

3.5 Recent approaches using the sequence to sequence neural network

This model has been successfully applied to a variety of NLP problems such as text summarization, machine translation, headlines generation, speech recognition. Traditional models as described above did perform poorly on text summarization. Recent advances in deep learning and machine learning technologies have resulted in motivating people to do more research and work in this field of text summarization.

3.5.1 Simple encoder-decoder network

This method was introduced by Bahdanau. 2014 [6]. This consists of an encoder-decoder RNN with an attention mechanism. This consists of an encoder and a decoder. An encoder neural network reads and encodes a source sentence into a fixed-length vector. A decoder then outputs a translation from the encoded vector. The whole encoder-decoder system, which consists of the encoder and the decoder for a language pair, is jointly trained to maximize the probability of a correct translation given a source sentence [6]. The major issue with this approach was that it failed to scale up on large sentences. This may be the case when sentences are larger than that of training dataset sentences. They also use bidirectional RNN which scans the input from both the left and right to get the context right. The forward RNN read the input sequence as it is ordered and calculates a sequence hidden states. The backward RNN reads the sequence in reverse order and calculates a sequence of backward hidden states. They then concatenate both of them to get final encoder hidden state. This is different from vanilla encode-decoder networks which read the input in a forward manner and encoding it and passing the information forward. The decoder is another RNN which learns to decode the thought vector.

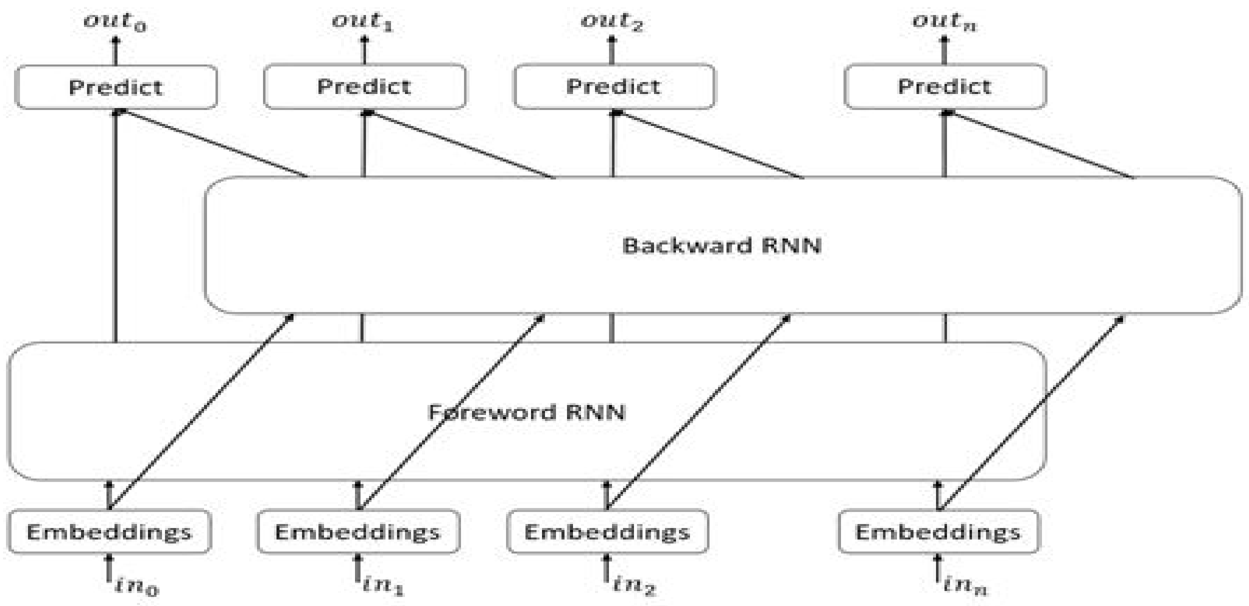


Figure: Representation of the encoder-decoder network

3.5.2 Attention mechanism

The basic encoder-decoder model performs okay on very short sentences but it fails for large sentences. Inspired by the performance of neural machine translation (NMT) [6], Rush et al [7] introduced a neural attention sequence to sequence model with an attention-based encoder and a neural network language model decoder to do the abstractive text summarization. The shortcomings of this model were that it was not able to handle out of vocabulary words. It failed to perform for rare words. It was not able to reproduce the message the original article was trying to convey.

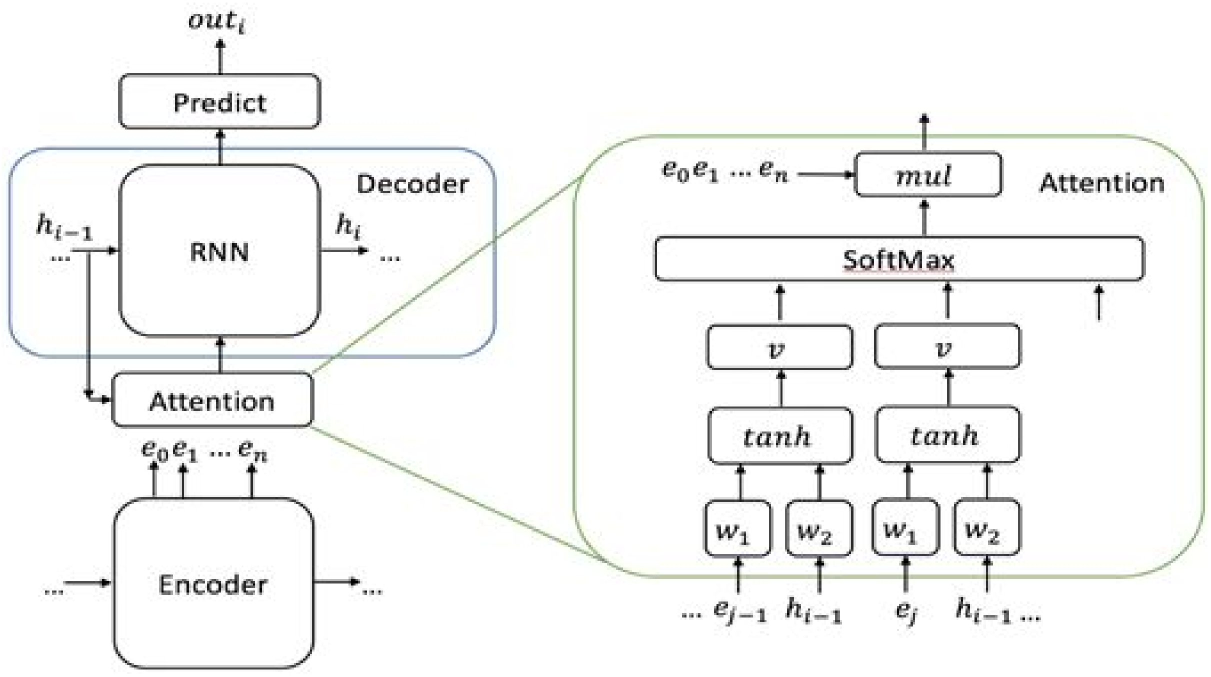


Figure:Representation of attention mechanism

To overcome these shortcomings [8] Liu proposed a model known as **"pointer generator network".** This is a combination of abstractive and extractive text summarization methods. This copies words from the source text and then generates novel words from the training corpus. With the help of this mechanism problem of out of vocabulary words. The problem of convergence and words repetitions has been addressed by the mechanism of intra-decoder attention mechanism [9].

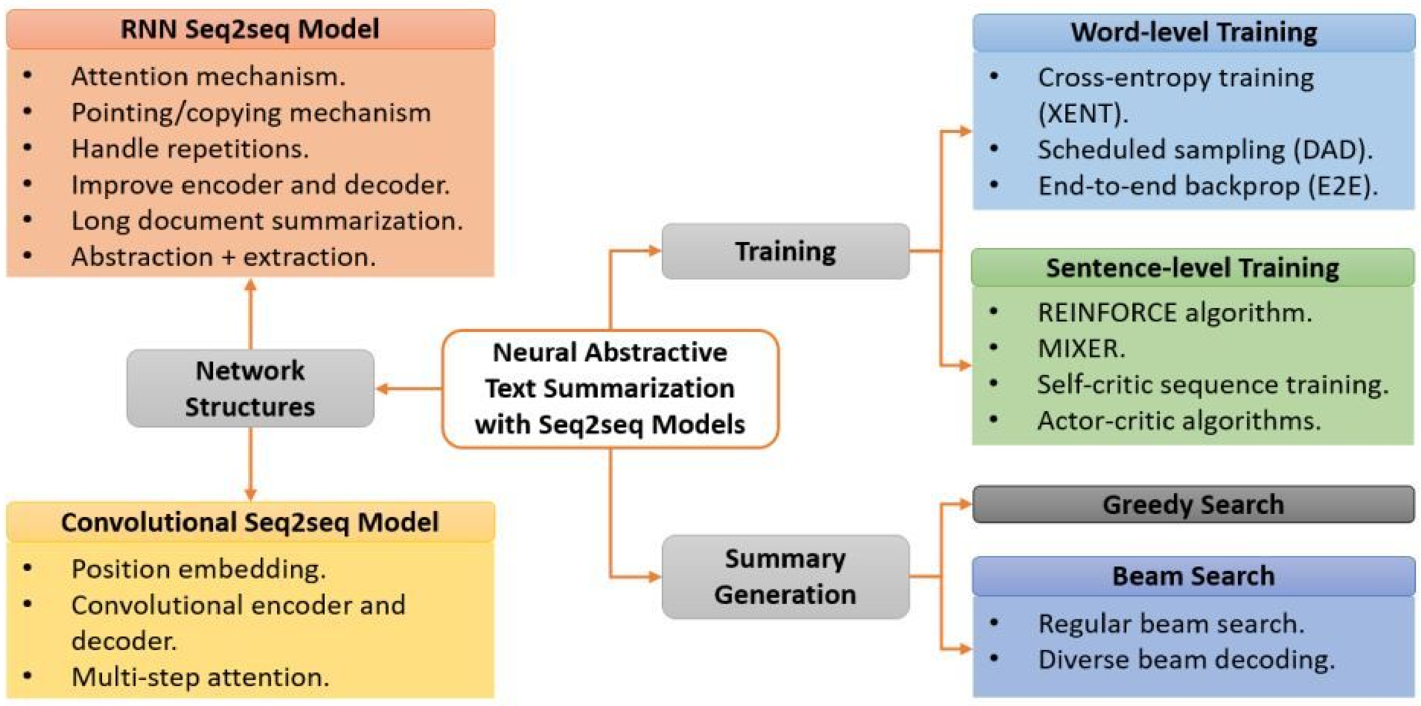

RNN Seq2seq Model
Attention mechanism.
Pointing/copying mechanism
Handle repetitions.
Improve encoder and decoder.
Long document summarization.
Abstraction + extraction.
Network Structures
Neural Abstractive Text Summarization with Seq2seq Models
Convolutional Seq2seq Model
Position embedding.
Convolutional encoder and decoder.
Multi-step attention.
Training
Summary Generation
Word-level Training
Cross-entropy training (XENT).
Scheduled sampling (DAD).
End-to-end backprop (E2E).
Sentence-level Training
REINFORCE algorithm.
MIXER.
Self-critic sequence training.
Actor-critic algorithms.
Greedy Search
Beam Search
Regular beam search.
Diverse beam decoding.

Figure: An overall flow chart of sequence to sequence models

3.6 Extractive text summarization

These methods involve three steps to generate a summary of a given article and text.

1. **Construction of a numeric representation of the input text.**
   There are different methods used here. These are word2vector, fasttext,tf-idf weighted average of the words.
2. **Scoring the sentences based on its modelling in terms of length, representation.**
   Different approaches which exist here are text rank, page rank algorithm etc.
3. **Joining these sentences to generate a summary.**

   3.6.1 Term frequency driven based approaches

This method uses tf-Idf to find the importance of each words. Based on this words with uttermost important are included in the summary.

3.6.2 Cluster based approaches

Documents and articles are written in such a way that they are able to address multiple topics and subtopics. It is very obvious to think that the generated summary should be able convey these topics. Some of the developed models use clutering to incorporate this feature and aspect through clustering. [10] proposes this feature using vector space model.

CHAPTER 4.
PROJECT REQUIREMENTS SPECIFICATION

4.1 Scope of the product

Our product allows the user to enter the URL of the news article for which they want a summary. This is then scraped and is cleaned to store it in a text file. This then passes through our machine learning model to generate a text summary.

Our model faces difficulty in handling large articles and generating their summary. It also cannot handle OOV words properly. Our product currently works for the English language. This is because we did not train our model for other languages. We have trained our model in English datasets.

4.2 Product perspective

This product is a completely independent product for which the development will start from scratch.

- As mentioned earlier it requires us to train a machine learning model which is computationally very expensive so good hardware is required. Probably a GPU machine is required for fast and easy training of the model.
- Our front-end will be web-based so all the tools which are used in the development of website etc will be used.
- For deployment, the user must have a web-browser with an internet connection to be able to use our product.

4.2.1 User Characteristics

The eventual users of our product will be a naive user who knows how to browse on the internet. Therefore our UI/UX must be very clean and easy to use. It should be well designed and should also suite a naive user.

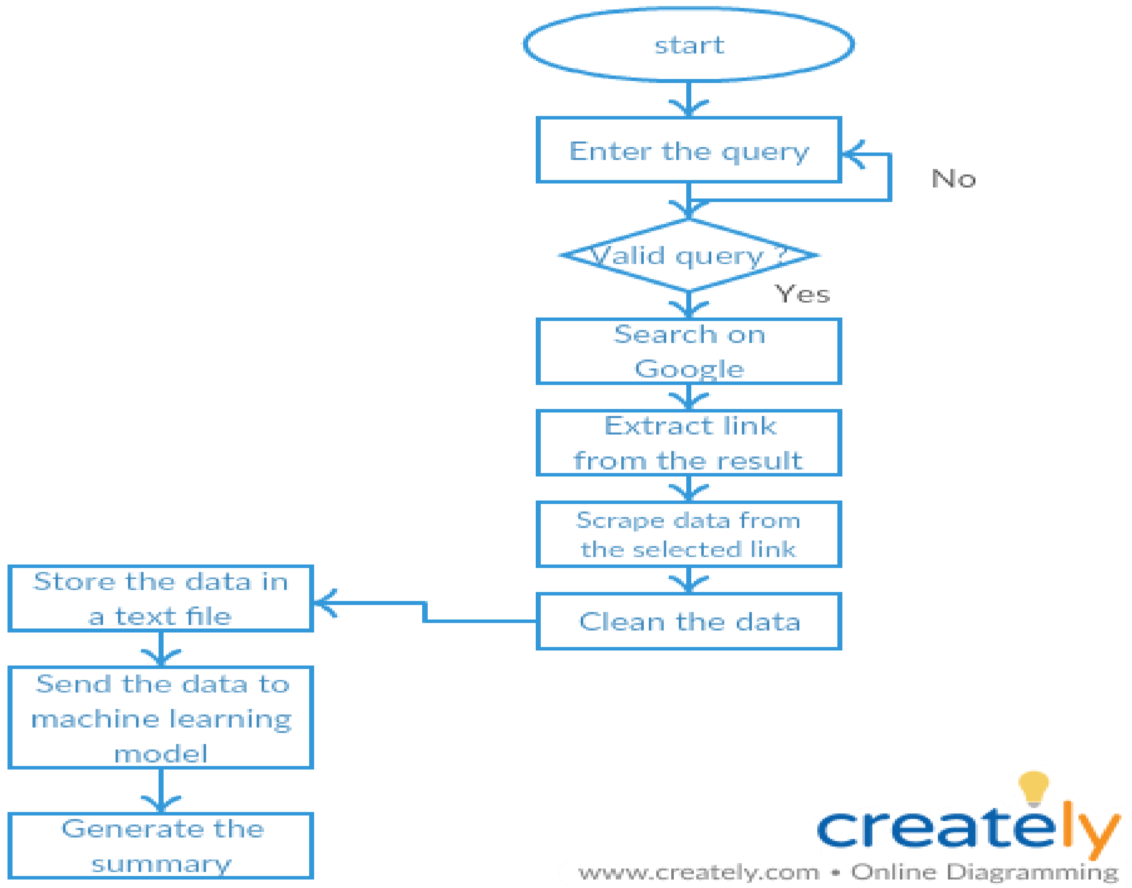
4.2.2 General Constraints, Assumptions and Dependencies

- Hardware limitation in terms of strong GPU.
- Assumptions the user is naive web user.

4.2.3 Risks

As of now, the only risk is burning out of the CPU if the model is trained using it. A computer with a powerful GPU is strongly recommended for the training phase. The other problem which we may face because of not having a powerful GPU is a large training time for our model.

4.3 Sequence diagram


start
Enter the query
No
Valid query ?
Yes
Search on Google
Extract link from the result
Scrape data from the selected link
Clean the data
Store the data in a text file
Send the data to machine learning model
Generate the summary
creately
www.creately.com • Online Diagramming

This is done in the following ways:-

- Allow the user to enter the query. (on a web application)
- If the query is valid, search the query on Google.
- Google will return multiple results related to query, extract all the links on the first page(because the links are highly relevant to user query)
- Scrape and clean the data from the chosen link and store it in a text file
- Send the data to machine learning models to generate a summary(extractive)

4.4 External interface requirement

4.4.1 Hardware Requirements

All the devices which have a web browser with a fast internet connection can use our product. Even though the development of our product requires fancy hardware but when using our product this will not be required.

4.4.2 Software Requirements

The machine used may have 64 bit Windows 10 or Ubuntu 16.04 or Mac OS. It also requires the latest version of a web browser (preferably chrome). Along with it libraries to be used including nltk and beautiful soup must be available. Tensorflow for development of neural network model is needed. Flask framework for running the product server.

4.4.3 Communication Interfaces

Since our product is web-based hence it requires a good internet connection.

4.5 User Interfaces

Our application is a web-based product. Considering a naive user, the UI/UX must be very simple, clean and interactive. The following are the key points we are keeping in mind to design our user interface:-

- Attractive and Innovative
- Simple to use for naive users

- Responsive in short time span
- Clear to understand and use
- Consistent on all interfacing screens and environments

4.6 Performance Requirements

The performance of our product will be based on our UI/UX and the ability to generate accurate summaries. Some of the parameters for this are:-

UI/UX performance metrics.

- clarity—The textual messages a user receives from an application must be clear.
- consistency—Use terms and labels for user interface controls that are familiar to users.
- conciseness—The UI must be very concise. It should be to the point..

Summarization accuracy and other backend metrics.

- We shall evaluate the coherence and quality of our summary.
- Given the target summary, we can use the ROUGE metric to evaluate the extractive summary generated by our model.
- Our database should be secured so as that no information about the system or the personal details of users must not be stolen.

4.7 Special Characteristics

The following actions/ methodology which we will be taking to make our software product more secure:-

- Use of prepared statements to safeguard our software against SQL injection.
- All the credentials will be encrypted.
- CIA triad will be kept in mind during the development phase.

4.8 Help

The help planned to develop our product mainly consists of many research paper which we used to do our literature survey. Other help includes different medium articles etc. Documentations for most of the library is available hence that is a great source of help.

4.9 Other Requirements

4.9.1 Site Adaptation Requirements.

- We will require BBC news dataset or equivalent to train our model.
- We will use Google search engine to search for relevant links as per the needs of the user.

4.9.2 Safety Requirements

The CIA (Confidentiality, Integrity, and Availability) triad of information security is a security benchmark model used to evaluate the information security of an organization. The CIA triad of information security implements security using three key areas related to information systems.These are Confidentiality, Integrity, Availability.

4.10 Packaging

Our software product is a web-based product.

CHAPTER 5
HIGH LEVEL DESIGN

5.1 Scope

Our product allows the user to enter the URL of the news article for which they want a summary. This is then scraped and is cleaned to store it in a text file. This then passes through our machine learning model to generate a text summary.

Our model faces difficulty in handling large articles and generating their summary. It also cannot handle OOV words properly. Our product currently works for the English language. This is because we did not train our model for other languages. We have trained our model in English datasets.

5.2 Design Constraints, Assumptions and Dependencies

We are constrained by the length of sentences as machine learning model needs fixed length vectors,We are hence restricting the maximum length to 35. If the sentence length is than 35 then we append a vector of 0's to it.

GUI constraints include multiple users to be able to access the application and select the news they want to read and give the summaries for the articles.

Since we have used BBC news article for all our training purpose hence there may be a chance of our model being biased towards BBC news.

One other problem is that summary which the dataset contains is not fully extractive by nature, so don't know how the model behaves while doing extractive text summarization.

Assumption: Importance of words is given by its context

5.3 Design descriptors

5.3.1 Use case diagram

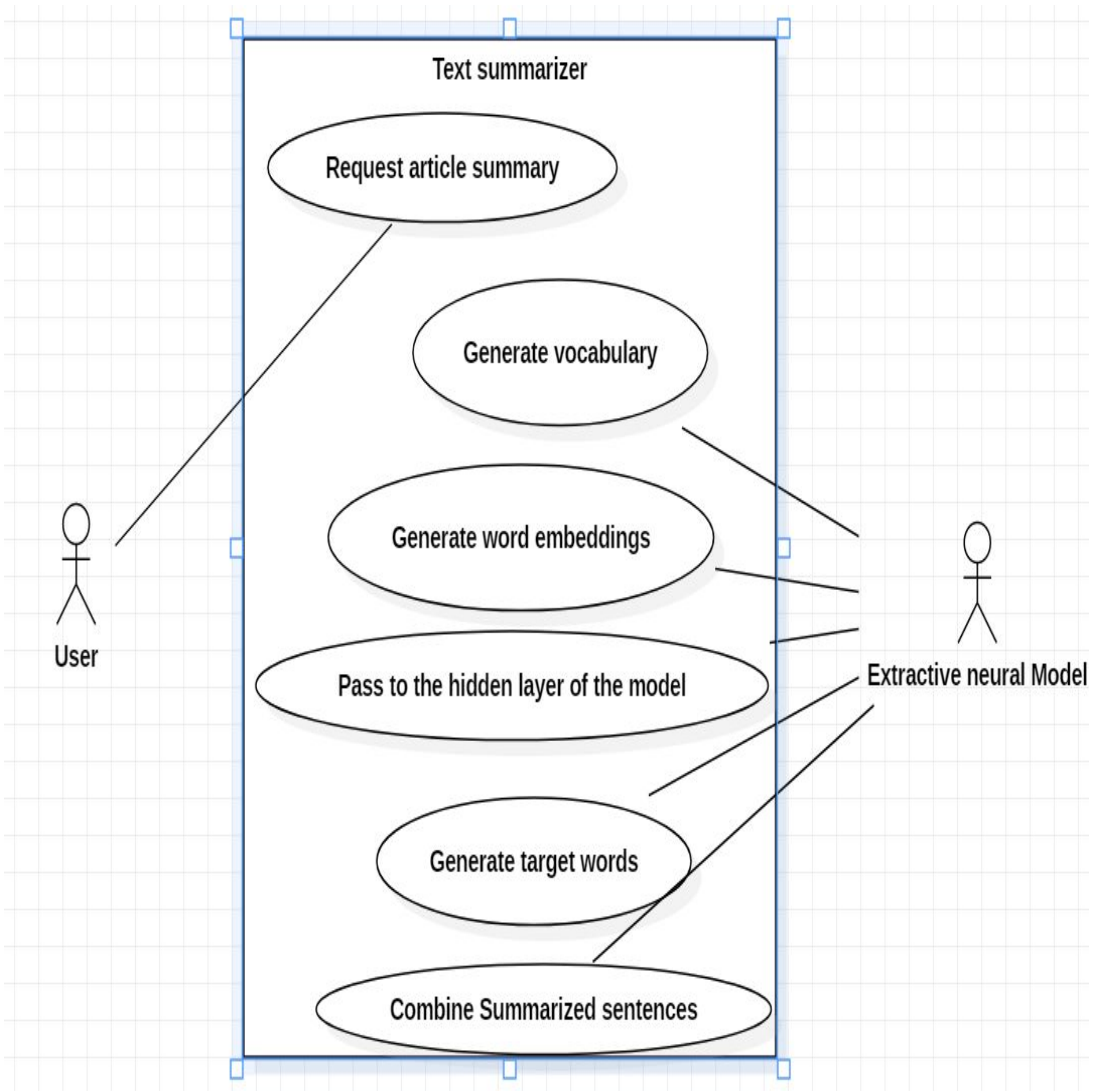


Figure: Use case diagram

As seen in the given diagram,where it is depicted that the sole responsibility of the user is only to request the summary for an article and rest will be taken care by the summarizer i.e.,vocabulary is generated and so are word embeddings. the word embeddings are passed to a machine learning model and target words are generated which are combined to form sentences.

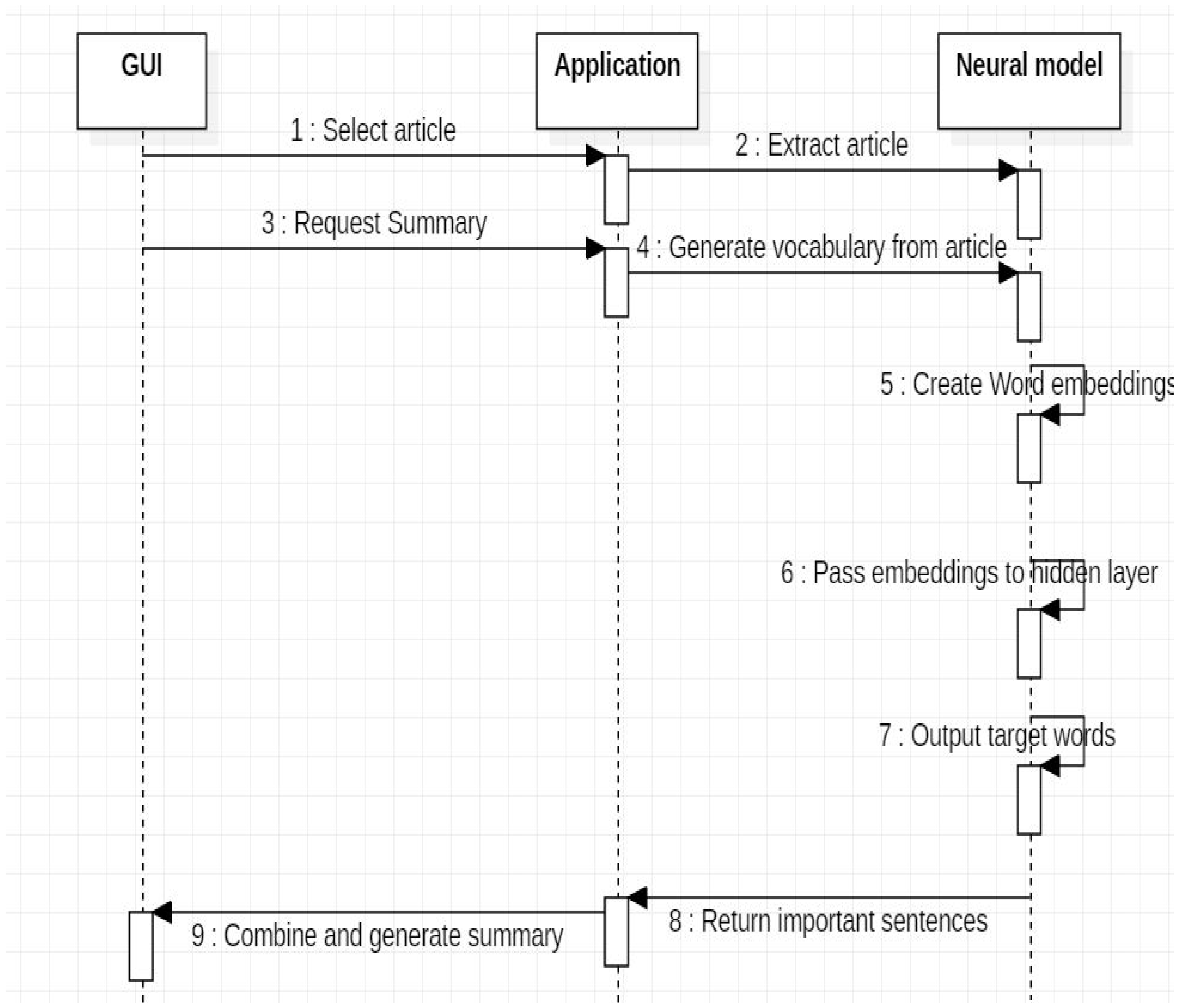

GUI
Application
Neural model
1 : Select article
2 : Extract article
3 : Request Summary
4 : Generate vocabulary from article
5 : Create Word embeddings
6 : Pass embeddings to hidden layer
7 : Output target words
8 : Return important sentences
9 : Combine and generate summary

Figure: Sequence Diagram

5.4 Modules

5.4.1 Module 1 : Structuring the dataset and cleaning it

Our data set has an article and a corresponding extractive summary of that article. Each article and its summary is structured in different categories of news viz entertainment, politics, business, sports and tech. This module will be responsible for converting each article into list and then cleaning it.

5.4.2 Module 2 :Word embeddings

This module is responsible for converting words into its embeddings. Machine Learning is, after all, a bunch of mathematical operations translated to a computer via low-level programming languages. Computer is brilliant when they work with numerical data. The two approaches used are :

5..4.2.1 Word2vector model

In this word embeddings and importance of word is given by its context. The architecture for that is :

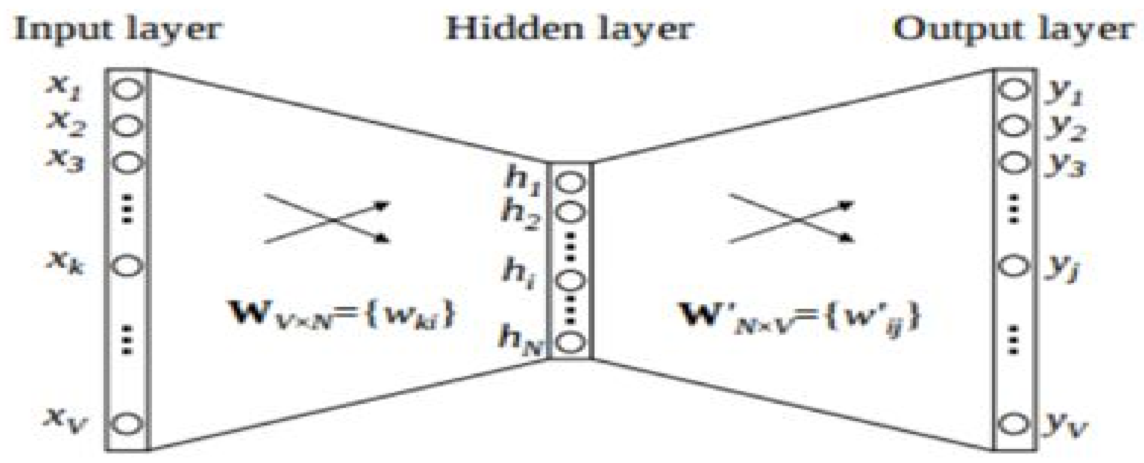


Figure: Architecture of word2vector model.

5.4.2.2 Fast text model for word embeddings

The issue with word2vec approach was that it was not able to handle out of vocabulary words. Hence, we used fast text model which was developed by Facebook AI research team. On using them we arrived at a conclusion that this is better than word2vector model. This has following advantages:

- Generate better word embeddings for rare words (even if words are rare their character n grams are still shared with other words - hence the embeddings can still be good).
- It handles and generates embeddings for OOV words.

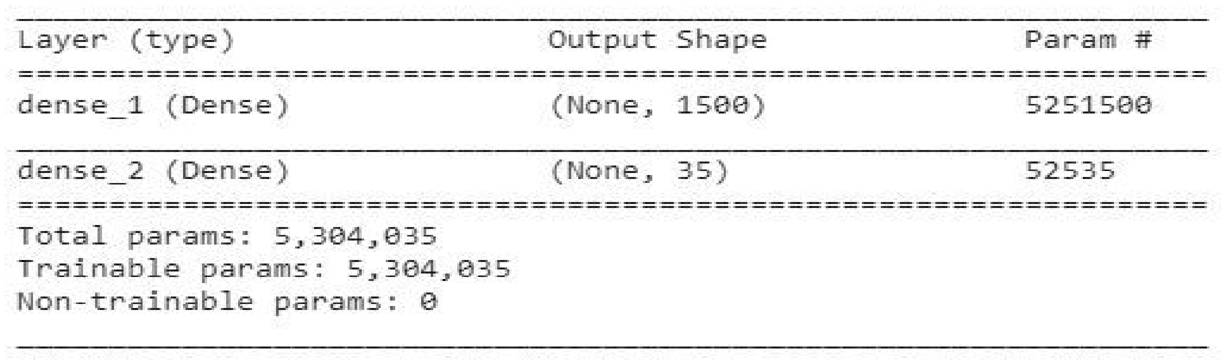
- This is fast in terms of training also when compared with word2vector model.

5.4.3 Module 3 : preparing the inputs

In the next stage of our implementation we prepare the input for our neural network. We divide our entire article into sentence and then vectorize each word(token) to infer the word embedding. Each article is divided into sentences. So now our entire training set is made of sentences. For the summary articles we do not infer a word embedding. This target is whether a word in the sentence appears in the summary. This target is vectorized. Now for each sentence we have its corresponding summary.

5.4.4 Module 4: Building the neural network model and train it.

We have used kearas with backend as tensorflow for the development of neural network In this module we build our machine learning model, and we will train it.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1500)              5251500
_________________________________________________________________
dense_2 (Dense)              (None, 35)                52535
=================================================================
Total params: 5,304,035
Trainable params: 5,304,035
Non-trainable params: 0
_________________________________________________________________
```

Figure: Neural network model parameters

5.4.5 Module 5: Performance evaluation and testing

In this stage of our implementation we test our model. What we obtain from the output is probability of whether a word should be included in summary or not. This is because we have applied softmax activation function to the output. To generate summary we have chosen top 15 probabilities from the max length of 35(tokens).

5.4.6 Module 6: Testing using ROUGE metric.

There is always a need to test the model using some mathematical function. One such performance evaluation metric is ROUGE metric. It stands for "**Recall-Oriented Understudy for Gisting Evaluation**".

5.4.7 Module 7: Flask app routes.

These are application URLs mapped to each module of the product. This is a decorator which tells which function(module) should be called for an application URL.

5.5 User Interface Diagram

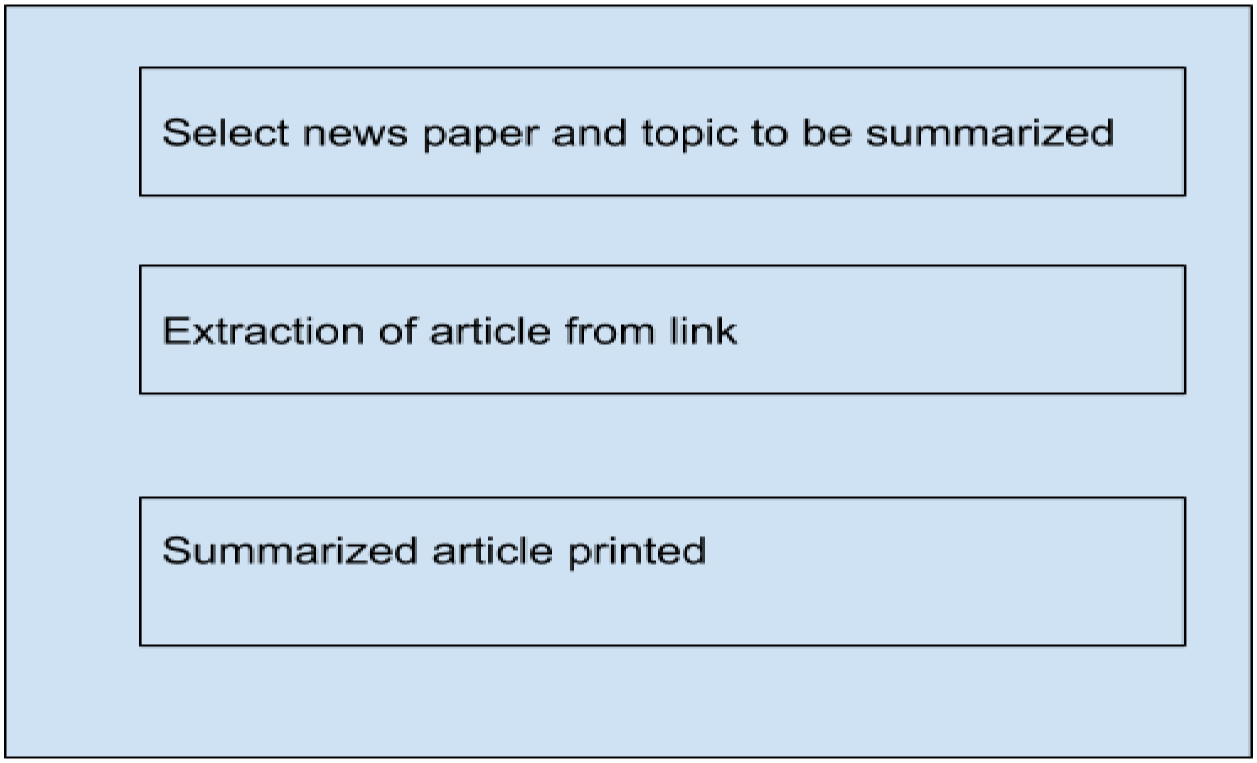

Select news paper and topic to be summarized
Extraction of article from link
Summarized article printed

Figure: External Interface Diagram

5.6 Package Diagram

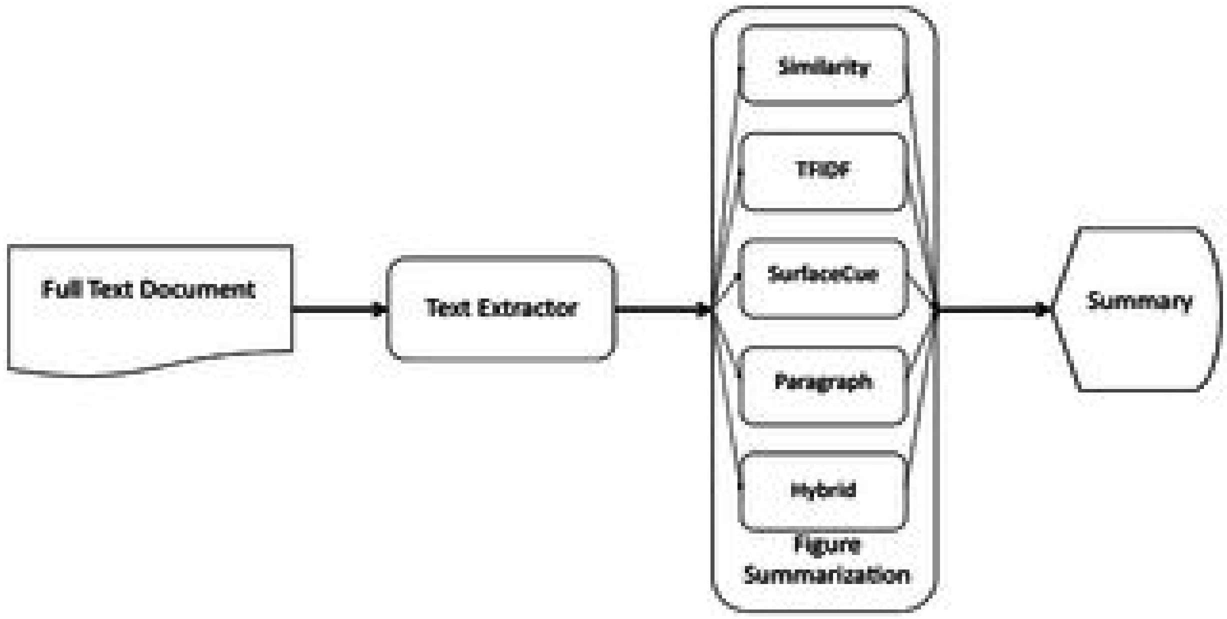


Figure: Package Diagram for our text summarization system

5.7 Help

The help planned to develop our product mainly consists of many research paper which we used to do our literature survey. Other help includes different medium articles etc. Documentations for most of the library is available hence that is a great source of help. These are mentioned in the references as well.

5.8 Reusability considerations

The neural model can be reused for various other text and semantic applications in NLP. The word embedding model can also be used for other semantic analysis tasks. We can even open source our embeddings and people could just use it for their researches and projects.

CHAPTER 6

LOW LEVEL SYSTEM DESIGN

6.1 Design constraints.

We are constrained by the length of sentences as machine learning model needs fixed length vectors,We are hence restricting the maximum length to 35. If the sentence length is than 35 then we append a vector of 0’s to it.

GUI constraints include multiple users to be able to access the application and select the news they want to read and give the summaries for the articles.

Since we have used BBC news article for all our training purpose hence there may be a chance of our model being biased towards BBC news.

One other problem is that summary which the dataset contains is not fully extractive by nature, so don’t know how the model behaves while doing extractive text summarization.

Assumption: Importance of words is given by its context

6.2 Design descriptors

Module 1: Structuring dataset and cleaning it

Descriptor: Cleaning the input and doing language processing. Cleaning involves cleaning the article for punctuations, escape sequences, alphanumeric strings etc. After doing this it is converted into lowercase so that there is no scope of ambiguity between uppercase and lowercase. This further removes outliers from the dataset. For that it uses standard deviation method in which threshold for outlier is the standard deviation.

Parameters: Takes a input article and cleans it for punctuations, stop words, escape sequences, alphanumeric strings etc

Data type: String

Return values: A clean string and creating pkl files locally which will be used by different modules.

Module 2 : Word embeddings

Descriptor: Computer is brilliant when they work with numerical data. Hence, we map each word into 100 dimension vectors. Word embedding should be such that words with similar meanings have similar representations. The two approaches used are :

Word2vector Model

In this word embeddings and importance of word is given by its context. This comes in two flavours CBOW (continuous bag of words) and N-grams. CBOW model tries to predict the current target word based on the source context words(surrounding words). So it has a context window which is used to predict current word based on its context window words. However this model has some drawbacks as it is not able to handle out of vocabulary words. This CBOW model is a neural network which gives us the 100 dimension vector mapped for each word.

FastText Model

It. is basically an extension of word2vec model. The difference is that word2vec learns the vector for individual word while on the other hand fast text learns the vector for its ngrams that are found in each word.

Parameters: Word(tokens) , word embedding model type , window size, threads

Return values: A 100 dimension word vectors mapped for each word is returned. We further store this model locally.

Module 3 : Preparing the input

Descriptor: We divide our entire article into sentence and then vectorize each word(token) to infer the word embedding. Each token is cleaned using the function mentioned in module 1 and then it lemmatized and then stop words are removed to make it clean. Each article is divided into sentences. So now our entire training set is made of sentences. For the summary articles we do not infer a word embedding. We use an entirely new approach in which a target is created for each sentence in the text based on the summary. This target is whether a word in the sentence appears in the summary. This target is vectorized. As such in the returned summary matrix, we have vectors of size 35(maximum sentence length) with 1s and 0s. This is better than previous approaches in which they take a weighted average of the words(tokens) in the sentence. Now for each sentence we have its corresponding summary.

Parameters: Articles, word embedding model,

Return values: Hence, each article is divided into sentences. So now our entire training set is made of sentences. Returned summary matrix, we have vectors of size 35(maximum sentence length) with 1s and 0s

Module 4 :Neural Network training

Descriptor: We build our machine learning model, and we will train it. We will use the numpy array of vectorized text created in module 3. So our source contains 100 dimensional vectors, and the maximum length of each sentence is 35. All such vectors are concatenated on order to form a 100 * 35 vectors. These vectors are fed into the input layer of the neural network. Therefore, our neural network input layer size is 3500(35 · 100). We create a hidden layer of 1500 nodes.

Parameters: Source vector, Target vector

Return values: A trained model with its architecture and weights being stored locally

Module 5 :Performance evaluation and testing

Descriptor: We test our model. What we obtain from the output is probability of whether a word should be included in summary or not. To generate summary we have chosen top 15 probabilities from the max length of 35(tokens). To identify these words from others we have given them a identity by changing it result value to 1. Using this we generate the summary. We also used a module which accepts a URL and scapes the cleaned article from that URL. This will add a feature to our product in which user just need to enter the URL of a news article to obtain its summary.

Parameters: Model architecture and Model weights

Return values: A summarized article for the given text.

Module 6 :Testing with URL

Descriptor: We test our model. We give user a flexibility to give his own URL for text summarization. We have used an API which takes URL as an input and it return the cleaned text scraped from that URL. The name of that API is Content Extractor. It basically return a JSON file containing the title, text, URL of the images in the article. We use text and then we pass this to summarization module

Parameters: Model architecture and Model weights and URL of the article

Return values: A summarized article for the given URL.

Module 7 :Testing with Rouge metric

Descriptor: We test our model with a metric known as ROUGE metric. This metric is used for evaluation of automatic summarization tasks. It works by comparing the model(system) generated summary against the original summary(typically produced by a human). It works by finding the overlapping words(from system generated summary) with the human generated summary. The mathematical formula for this is:

$$ROUGE - N = \frac{\sum_{S \in Ref.Sum} \sum_{gram_n \in S} Count_{match}(gram_n)}{\sum_{S \in Ref.Sum} \sum_{gram_n \in S} Count(gram_n)}$$

Consider: **System Summary**

"*The cat was found under the bed*"

Consider Reference summary

"*the cat was under the bed***"**

Using the formula we get

ROUGE(Precision): 4/5 which is equal to 0.8

Parameters: Model architecture and Model weights and URL of the article

Return values: A score for the evaluated summarized text. We use ROUGE 1 score as this of uttermost relevance to us.

Module 8 :Development of user interface

Descriptor: Development of user friendly user- interface. It should be clean and concise with low responsive time. The UI has an input box where user can just enter the URL of the article to get it summary. This also an input box where a user can type a generic text to get summary of that.

Parameters: User suggestions.

Return values: Highly user friendly user-interface which is clean and concise and has low responsive time.

Module 9 :Flask app routes

Descriptor: Integration of various modules with its user interface. It is done through app routes. These are application URLs mapped to each module of the product

Parameters: User interface, Modules

Return values: Product running on server. Modules are mapped to its routes and application URLs

CHAPTER 7

TEST STRATEGY

7.1 Introduction

We need to test our model with different types of article. Our model was trained on BBC news dataset. So It is biased towards BBC news article. We will test it for various news article. These are varied in terms of the news agency, authors, and its genre. We will be doing a unit testing for each module of our application.

7.2 Scope

Comparing the actual summary of the news article with the output summary of the news articles and finding out the news articles. Each and every word is compared with the original summary. Our metric ROUGE metric finds the overlapping words in the human generated news article and the computer generated news article.

7.3 Performance criteria

The generated summarized news article should be able to convey the message the original article was trying to create. It should not be biased. From what we researched we found that these article may not be grammatically correct, so we will not take the grammar into consideration. We are using ROUGE metric to get a score on the performance.

7.4 Test Environment

Since the product is a web based application, hence it should work irrespective of the environment. So we will test this on various operating system environment. It should work well for all the cases.

7.5 Risk Identification and Contingency Planning

Risk #	Risk	Nature of Impact	Contingency Plan
1)	Large data set	Accuracy may reduce	**Divide in data set in batches**
2)	Biased toward BBC news article	Poor results for other articles.	**Train it for other news dataset**

7.6 Test Schedule

Unit testing was done after the modules were developed. Final stages of testing was done for model and the UI integrated with model. These were done after the development was done..

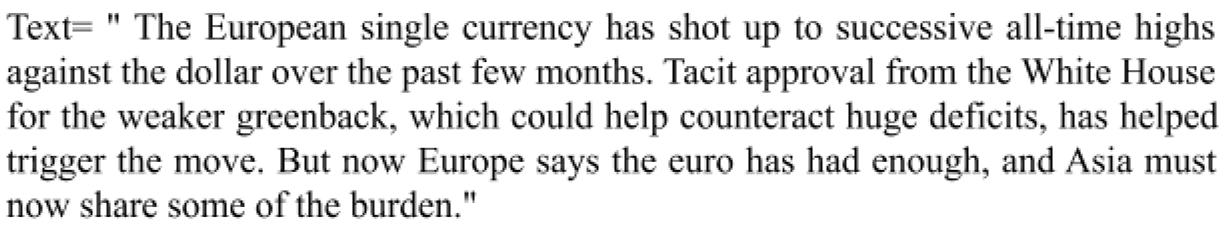
7.7 Acceptance criteria

The accuracy of the code should be high the UI should be user friendly. The code should take minimal time to execute. The summarized article should convey the message the original article was trying to convey.

7.8 Test Data

The dataset was divided into training and testing in the ratio of 80:20 respectively. The model was tested on this 20% of unseen data. Various other random articles were also used to test the application.

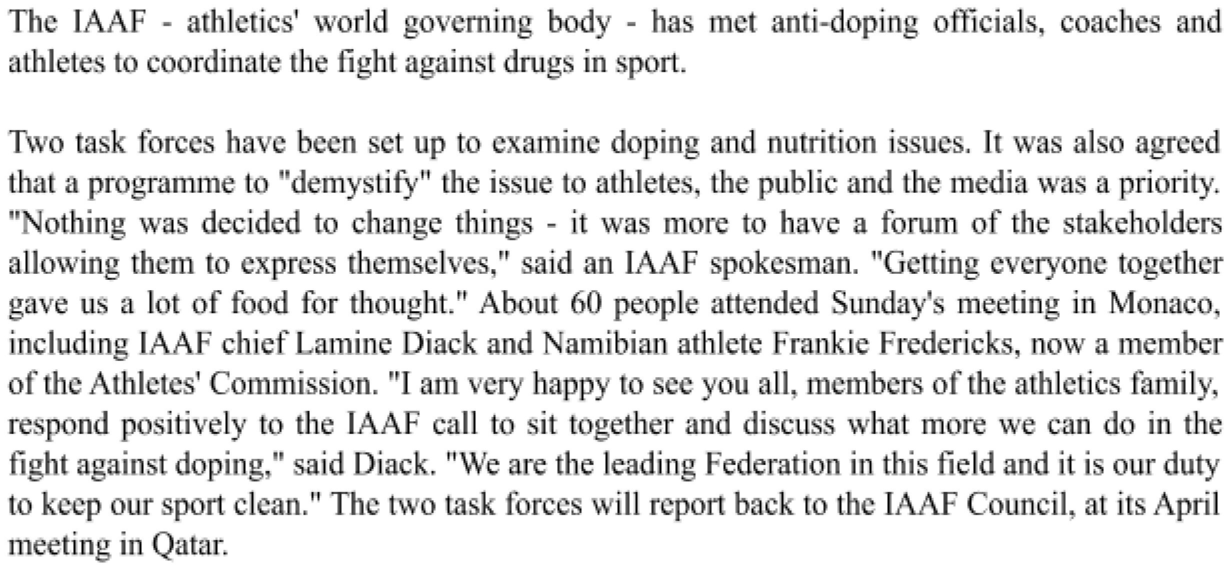
7.6.1 Sample test case 1


Text= " The European single currency has shot up to successive all-time highs against the dollar over the past few months. Tacit approval from the White House for the weaker greenback, which could help counteract huge deficits, has helped trigger the move. But now Europe says the euro has had enough, and Asia must now share some of the burden."

7.6.2 Sample test case 2(A news URL)

http://news.bbc.co.uk/2/hi/business/4236959.stm

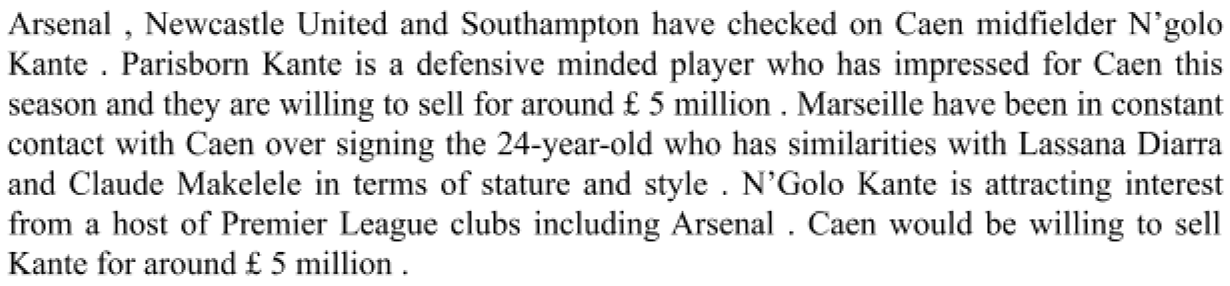
7.6.3 Sample test case 3



The IAAF - athletics' world governing body - has met anti-doping officials, coaches and athletes to coordinate the fight against drugs in sport.

Two task forces have been set up to examine doping and nutrition issues. It was also agreed that a programme to "demystify" the issue to athletes, the public and the media was a priority. "Nothing was decided to change things - it was more to have a forum of the stakeholders allowing them to express themselves," said an IAAF spokesman. "Getting everyone together gave us a lot of food for thought." About 60 people attended Sunday's meeting in Monaco, including IAAF chief Lamine Diack and Namibian athlete Frankie Fredericks, now a member of the Athletes' Commission. "I am very happy to see you all, members of the athletics family, respond positively to the IAAF call to sit together and discuss what more we can do in the fight against doping," said Diack. "We are the leading Federation in this field and it is our duty to keep our sport clean." The two task forces will report back to the IAAF Council, at its April meeting in Qatar.

7.6.4 Sample test case 4



Arsenal , Newcastle United and Southampton have checked on Caen midfielder N'golo Kante . Parisborn Kante is a defensive minded player who has impressed for Caen this season and they are willing to sell for around £ 5 million . Marseille have been in constant contact with Caen over signing the 24-year-old who has similarities with Lassana Diarra and Claude Makelele in terms of stature and style . N'Golo Kante is attracting interest from a host of Premier League clubs including Arsenal . Caen would be willing to sell Kante for around £ 5 million .

CHAPTER 8

IMPLEMENTATION AND PSEUDO-CODE

8.1 Structuring data set and cleaning it.

Our data set has an article and a corresponding extractive summary of that article. Each article and its summary is structured in different categories of news viz entertainment, politics, business, sports and tech.

The first stage of our implementation involved structuring this data set into a list each for news article and its summary. This involved cleaning the article for punctuation s, escape sequences, alphanumeric strings etc. After doing this it is converted into lowercase so that there is no scope of ambiguity between uppercase and lowercase. Following is the snippet of the function used to implement this functionality.

```
def clean_str(string):
    string = re.sub(r"\'s", "", string)
    string = re.sub(r"\'ve", "", string)
    string = re.sub(r"n\'t", "", string)
    string = re.sub(r"\'re", "", string)
    string = re.sub(r"\'d", "", string)
    string = re.sub(r"\'ll", "", string)
    string = re.sub(r",", "", string)
    string = re.sub(r"!", " ! ", string)
    string = re.sub(r"\(", "", string)
    string = re.sub(r"\)", "", string)
    string = re.sub(r"\?", "", string)
    string = re.sub(r"'", "", string)
    string = re.sub(r"[^A-Za-z0-9(),!?\'\`]", " ", string)
    string = re.sub(r"[0-9]\w+|[0-9]","", string)
    string = re.sub(r"\s{2,}", " ", string)
    string = re.sub(r"\n",'',string)
    string = re.sub(r"  ",' ',string)
    return string.strip().lower()
```

After this, we dump these lists locally using the pickle module. After this step, we have two files with the extension.**pkl** for the article and a summary of that article. When we did some analysis of the lists created we noticed that the standard deviation of the article length was about 8 sentences, the minimum length was 1 sentence. So we cleaned it further by removing the outliers (in terms of the number of sentences) from our data set. We used the standard deviation method to remove the outliers from the data set. In this method, the standard deviation is used as a cut-off to identify and remove outliers from the data set. Outliers are removed in such a way that if an article is found to be an outlier than its corresponding summary will also be removed and vice versa.

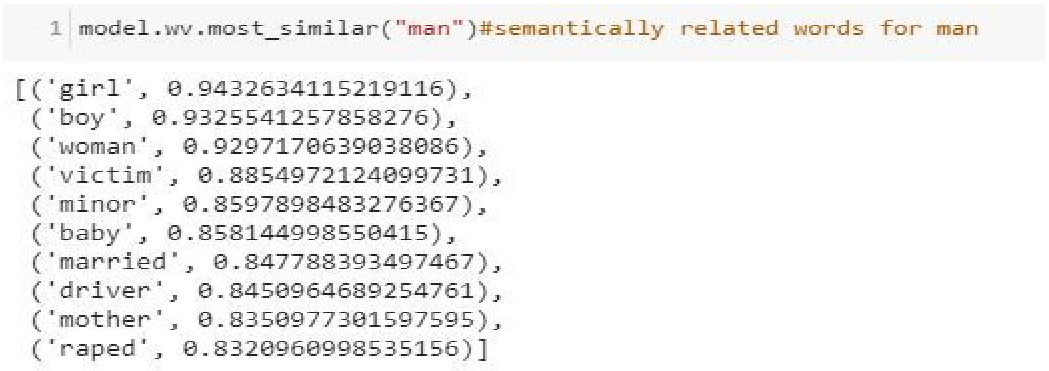
8.2 Word Embedding.

The next stage of our implementation starts with converting words into its embedding. Machine Learning is, after all, a bunch of mathematical operations translated to a computer via low-level programming languages. Computer is brilliant when they work with numerical data. Hence, we map each word into 100 dimension vectors. Word embedding should be such that words with similar meanings have similar representations. Therefore, if we check the similarity of each numeric vector of the mapped word then it should be very close. Also, these embedding should encode the meaning and inherent semantics of the words. Traditional approaches includes one hot vector which essentially is a vector with target element being one and others as zero. Even though this method is fast and easy but it fails to capture the meaning of that word and also the relationship between two words. Hence, we should use a robust method which captures the relationship between the words and its context.
We tried two approaches which help to overcome the issues discussed above

1. **Word2Vector model:-**
   We used the gensim library which implements word2vector models in an optimized way. To do that we first converted our articles into tokens and then passed it for training Following snippet shows how to import the module and how to train it.

```
import multiprocessing
from gensim.models import Word2Vec
articles_tokens=[]
for i in range(len(df1["ctext"])):
    articles_tokens.append([word for word in
word_tokenize(df1["ctext"][i].lower().replace(".","
").translate(string.punctuation)) if len(word)>2])

model = gensim.models.Word2Vec(articles_tokens, min_count=5,size=100,workers=4)
```

Below snippet shows the output when we try to print the semantically related words for ‘man’



```
model.wv.most_similar("man")#semantically related words for man
```

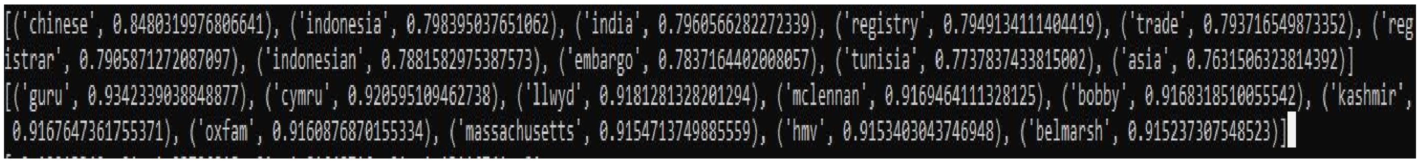
```
[('girl', 0.9432634115219116),
 ('boy', 0.9325541257858276),
 ('woman', 0.9297170639038086),
 ('victim', 0.8854972124099731),
 ('minor', 0.8597898483276367),
 ('baby', 0.858144998550415),
 ('married', 0.847788393497467),
 ('driver', 0.8450964689254761),
 ('mother', 0.8350977301597595),
 ('raped', 0.8320960998535156)]
```

2. **Fasttext model**

   The issue with word2vec approach was that it was not able to handle out of vocabulary words. Hence, we used fast text model which was developed by Facebook AI research team It. is basically an extension of word2vec model. The difference is that word2vec learns the vector for individual word while on the other hand fast text learns the vector for its ngrams that are found in each word. When trained we saw that it was able to generate the vector for that rare words.

```
import multiprocessing
from gensim.models import FastText
articles_tokens=[]
for i in range(len(df1["ctext"])):
    articles_tokens.append([word for word in
word_tokenize(df1["ctext"][i].lower().replace(".","
").translate(string.punctuation)) if len(word)>2])

model_ted = FastText(text_vocab, size=100, window=5, min_count=5,
workers=4,sg=1)
```

Below snippet shows the output when we try to print the semantically related words for **'china'**



```
[('chinese', 0.8480319976806641), ('indonesia', 0.798395037651062), ('india', 0.7960566282272339), ('registry', 0.7949134111404419), ('trade', 0.793716549873352), ('reg
istrar', 0.7905871272087097), ('indonesian', 0.7881582975387573), ('embargo', 0.7837164402008057), ('tunisia', 0.7737837433815002), ('asia', 0.7631506323814392)]
[('guru', 0.9342339038848877), ('cymru', 0.920595109462738), ('llwyd', 0.9181281328201294), ('mclennan', 0.9169464111328125), ('bobby', 0.9168318510055542), ('kashmir',
 0.9167647361755371), ('oxfam', 0.9160876870155334), ('massachusetts', 0.9154713749885559), ('hmv', 0.9153403043746948), ('belmarsh', 0.915237307548523)]
```

8.3 Preparing the input

In the next stage of our implementation we prepare the input for our neural network. We divide our entire article into sentence and then vectorize each word(token) to infer the word embedding. Each token is cleaned using the function mention in 8.1 and then it lemmatized and then stop words are removed to make it clean. We put a constraint that the length of each sentence in terms of words(tokens) should not be more than 35. If the sentence length is less than 35 we append a 100 dimension vector of 0's. Hence, each article is divided into sentences. So now our entire training set is made of sentences. For the summary articles we do not infer a word embedding. We use an entirely new approach in which a target is created for each sentence in the text based on the summary. This target is whether a word in the sentence appears in the summary. This target is vectorized. As such in the returned summary matrix, we have vectors of size 35(maximum sentence length) with 1s and 0s. This is better than previous approaches in which they take a weighted average of the words(tokens) in the sentence. Now for each sentence we have its corresponding summary. We have also put a case to skip the words(rarest of rare) which are not in the training corpus. The snippet for that is shown below.

```
def vectorize_texts_and_summaries(texts, summaries):
    ct=0
    global MAX_SENTENCE_LEN
    nested_list_len = lambda x: sum(len(list) for list in x)
    t_vectors = []
    source_text_vectors = np.zeros((nested_list_len(texts),
MAX_SENTENCE_LEN*EMBEDDING_SIZE))
    s_vectors = []
    target_summary_vectors = np.zeros((nested_list_len(texts), MAX_SENTENCE_LEN))
    vec_idx = 0
    not_vocab=[]
    if(type(texts[0]) == list):
        for i in range(len(texts)):
            summary = summaries[i]
            sentences = texts[i]
            sentences_container = []
            # Get text vector
            for s in sentences:
                sentence_vector = np.array([])
                target_vector = np.array([])

s=remove_stopwords(strip_punctuation(strip_non_alphanum(str(s).lower())))
                s=clean_str(s)
                for w in word_tokenize(s):
                    if(model_ft.__contains__(w)==False):
                      print(w)
                      continue
                    if(len(sentence_vector) < MAX_SENTENCE_LEN*EMBEDDING_SIZE):
                        sentence_vector = np.append(sentence_vector, model_ft[w])
                        target_vector = np.append(target_vector, int(w in summary))
                    else:
                        break
                while(len(sentence_vector) < MAX_SENTENCE_LEN*EMBEDDING_SIZE):
                    sentence_vector =
np.append(sentence_vector,np.zeros(EMBEDDING_SIZE))
                    target_vector = np.append(target_vector, 0)
                source_text_vectors[vec_idx] = sentence_vector
                target_summary_vectors[vec_idx] = target_vector
                vec_idx+=1
    return (source_text_vectors, target_summary_vectors)
```

We then save these vectorized numpy array locally using the snippet given below

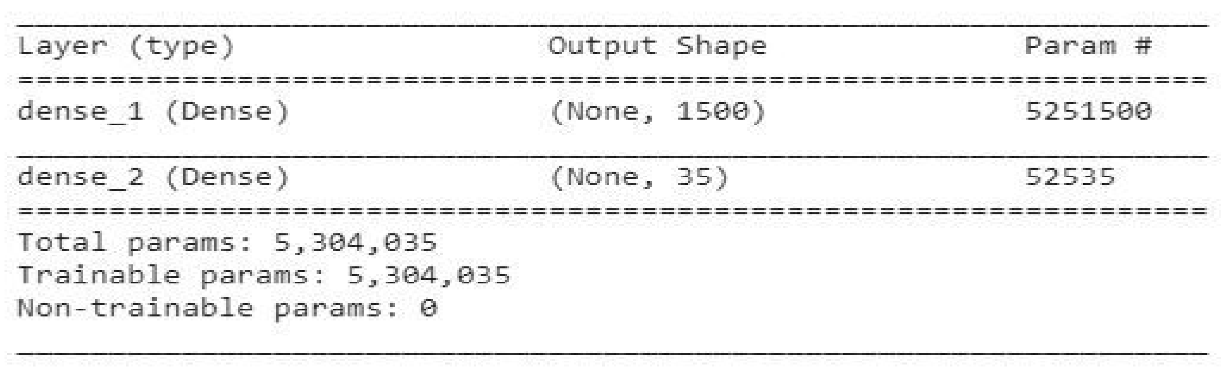
```
# SAVE
np.save('inputs.npy', source_vec, allow_pickle=False)
np.save('outputs.npy', target_vec, allow_pickle=False)
print(len(source_vec),len(target_vec))
```

```
41403 41403
```

8.4 Building the neural network model and training it

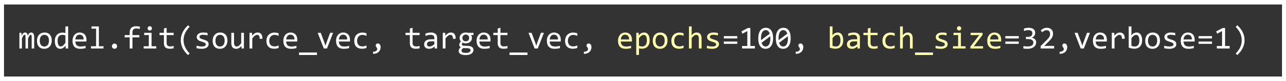
In the stage of our implementation we build our machine learning model, and we will train it. We will use the numpy array of vectorized text created in 8.3. So our source contains 100 dimensional vectors, and the maximum length of each sentence is 35. All such vectors are concatenated on order to form a 100 * 35 vectors. These vectors are fed into the input layer of the neural network. Therefore, our neural network input layer size is 3500(35 · 100). We create a hidden layer of 1500 nodes. The output layer is of length 35 which is the size of our target vector created in 8.3. A softmax activation layer is applied to the output of the last layer. We use Adams optimization. Each obtained vector denoted the weight associated with the corresponding sentence which represents the measure of belief of the sentence being included in the summary Loss from the predicted model is back propagated into the model. This is done for each iteration(epoch). We train our model for 100 epochs. Training is done in batches to avoid memory constraints. The training was done using GPUs available on google co laboratory.

```
model = Sequential()
model.add(Dense(1500, input_dim=3500, activation='relu'))
model.add(Dense(35, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
model.summary()
```

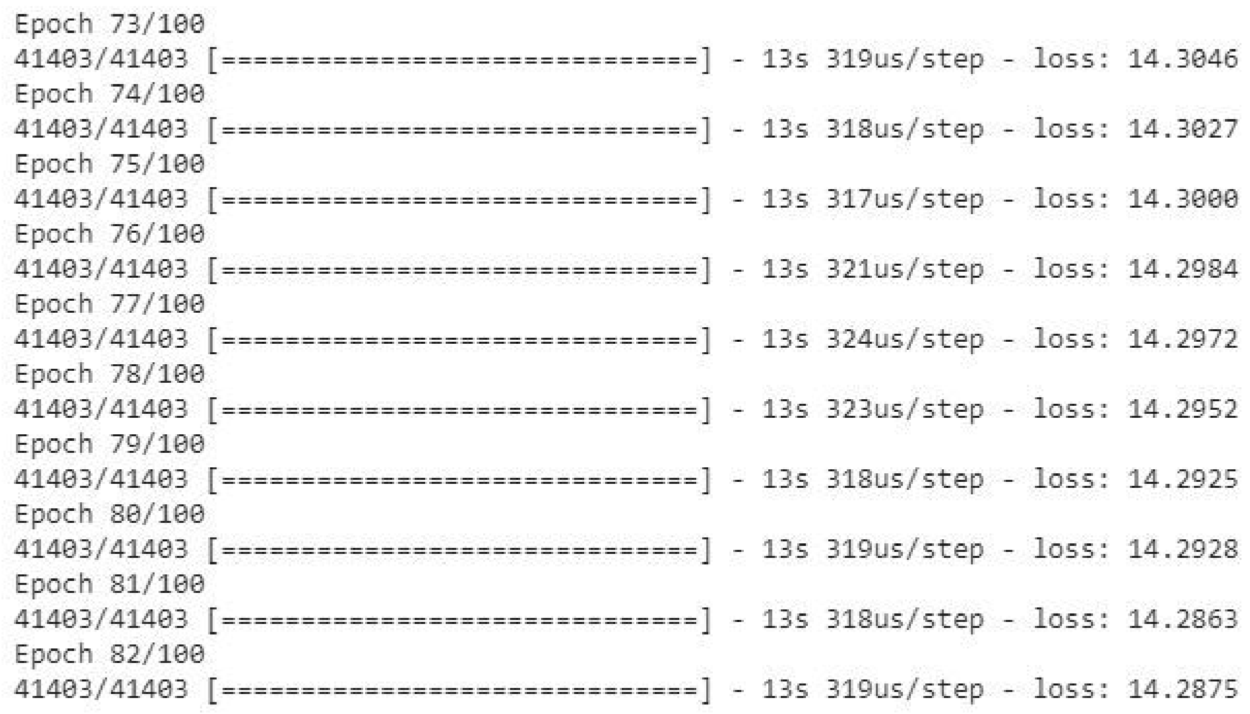


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1500)              5251500
_________________________________________________________________
dense_2 (Dense)              (None, 35)                52535
=================================================================
Total params: 5,304,035
Trainable params: 5,304,035
Non-trainable params: 0
_________________________________________________________________
```

Below is the snippet to train the model.



```
model.fit(source_vec, target_vec, epochs=100, batch_size=32,verbose=1)
```



```
Epoch 73/100
41403/41403 [==============================] - 13s 319us/step - loss: 14.3046
Epoch 74/100
41403/41403 [==============================] - 13s 318us/step - loss: 14.3027
Epoch 75/100
41403/41403 [==============================] - 13s 317us/step - loss: 14.3000
Epoch 76/100
41403/41403 [==============================] - 13s 321us/step - loss: 14.2984
Epoch 77/100
41403/41403 [==============================] - 13s 324us/step - loss: 14.2972
Epoch 78/100
41403/41403 [==============================] - 13s 323us/step - loss: 14.2952
Epoch 79/100
41403/41403 [==============================] - 13s 318us/step - loss: 14.2925
Epoch 80/100
41403/41403 [==============================] - 13s 319us/step - loss: 14.2928
Epoch 81/100
41403/41403 [==============================] - 13s 318us/step - loss: 14.2863
Epoch 82/100
41403/41403 [==============================] - 13s 319us/step - loss: 14.2875
```

Below snippet shows the saving of our model locally. The weights of our model are stored locally along with the model architecture.

8.5 Performance evaluation and testing

In this stage of our implementation we test our model. What we obtain from the output is probability of whether a word should be included in summary or not. This is because we have

Applied softmax activation function to the output. To generate summary we have chosen top 15 probabilities from the max length of 35(tokens). To identify these words from others we have given them a identity by changing it result value to 1. Using this we generate the summary. Following snippet shows this.

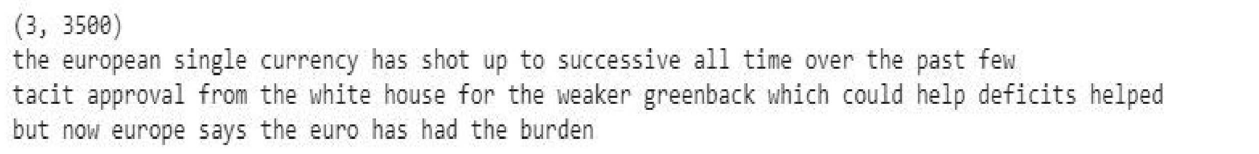
```
vec=make_array_vectorize(text)
json_file = open('model.json', 'r')# load model
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
# load weights into new model
loaded_model.load_weights("model.h5") # load weights
print("Loaded model from disk")

# evaluate loaded model on test data
loaded_model.compile(loss='categorical_crossentropy', optimizer='adam')
results=loaded_model.predict(vec)
for res in results:
  word_idx=sorted(range(len(res)), key=lambda i: res[i],
reverse=True)[:15]#making top 15 values as 1
  for i in word_idx:
    res[i]=1
print(results)
```

We also need to handle the cases when sentence length is greater than 35. If this is the case we removed stop words. If still the length is greater than 35 we just drop the words to make it of length 35. Following snipped shows this.

```
def make_text_35(text):
  words=text.split(' ')
  if(len(words)>35):
    text=remove_stopwords(text)
    words=text.split(' ')
    if(len(words)>35):
      words=words[:35]
    return (clean.join(w for w in words))
  else:
    return(text)
```

Following snippet shows us the code when it is tested for a news article.

```
text="The European single currency has shot up to successive all-time
highs against the dollar over the past few months. Tacit approval
from the White House for the weaker greenback, which could help
counteract huge deficits, has helped trigger the move. But now Europe
says the euro has had enough, and Asia must now share some of the
burden."

arr_text=extract_sentences_from_paragraph(text)
vec=make_array_vectorize(text)
print(vec.shape)
results=loaded_model.predict(vec)
# ck=results.copy()
for res in results:
  word_idx=sorted(range(len(res)), key=lambda i: res[i],
reverse=True)[:15]
  for i in word_idx:
    res[i]=1
# for res in results:
#   for sent in sent_tokenize(text):
# #     print(sent)
#     print( " ".join([word for idx,word in enumerate(sent.split("
")) if(res[idx] == 1)]))
for res,sent in zip(results,arr_text):
  sent=make_text_35(sent)
  print( " ".join([word for idx,word in enumerate(sent.split(" "))
if(res[idx] == 1)]))
```

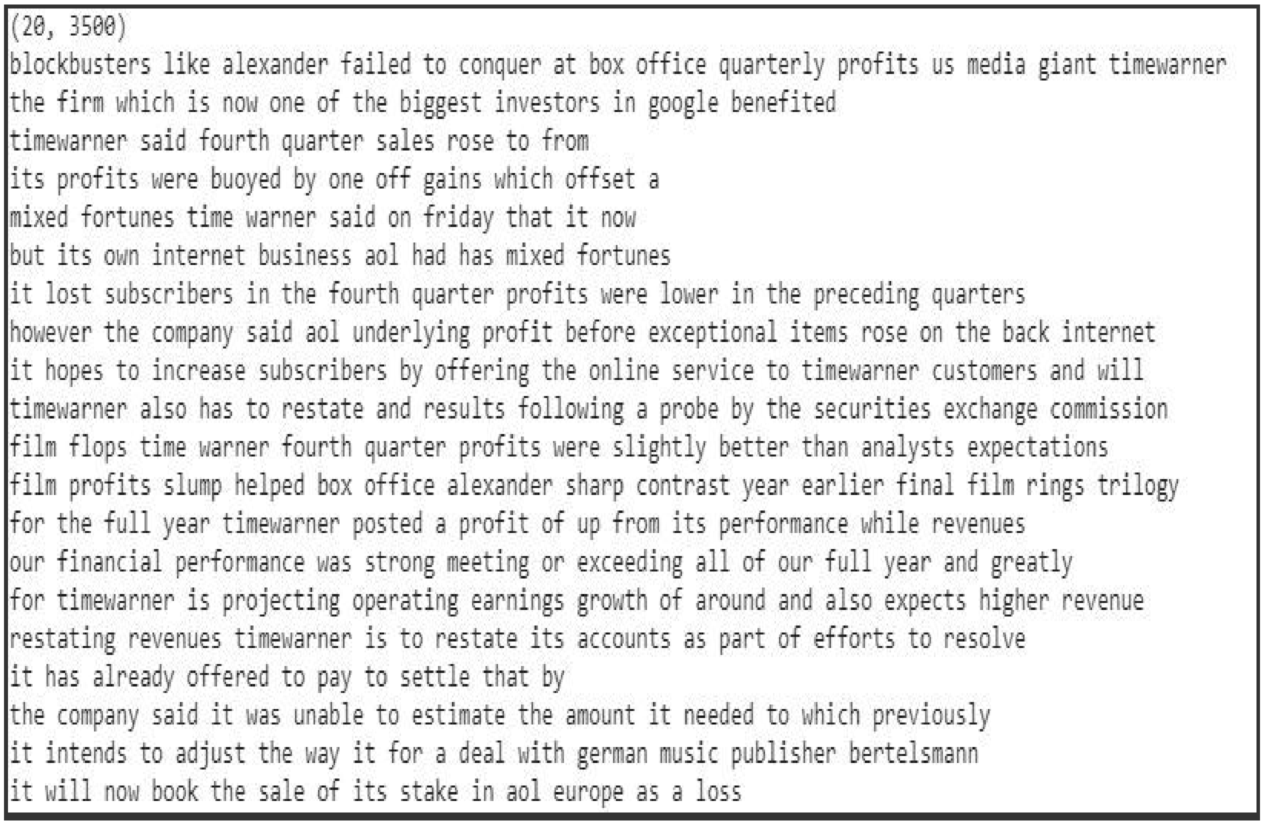


```
(3, 3500)
the european single currency has shot up to successive all time over the past few
tacit approval from the white house for the weaker greenback which could help deficits helped
but now europe says the euro has had the burden
```

Our application also accepts a **url** of the news article, to find its summary. We have used an api for extracting text from the url. Following snipped shows this.

```
def test_with_url(url):
  url='https://contentxtractor.com/api/v1/?a=grab&key=12jopu98&url='+str(url)
  headers={'User-Agent':user_agent,}
  request=urllib.request.Request(url,None,headers) #The assembled request
  response = urllib.request.urlopen(request)
  data = response.read() # The data u need
  output = json.loads(data)
  print (output)
  print(output['data']['text'])
  text=output['data']['text']
  arr_text=extract_sentences_from_paragraph(text)
  print(len(arr_text))
  vec=make_array_vectorize(text)
  print(vec.shape)
  results=loaded_model.predict(vec)
  for res in results:
    word_idx=sorted(range(len(res)), key=lambda i: res[i], reverse=True)[:15]
    for i in word_idx:
      res[i]=1
  sent_sum=' '
  for res,sent in zip(results,arr_text):
    sent=str(make_text_35(sent))
   s_indv=' '.join([word for idx,word in enumerate(sent.split(" ")) if(res[idx]
== 1)])
    sent_sum=str(sent_sum)+str(s_indv)+str(',')
    print(s_indv)
  return sent_sum
```

```
url='http://news.bbc.co.uk/2/hi/business/4236959.stm'
# test_with_url(url)
text_summary=print(test_with_url(url))
```



```
(20, 3500)
blockbusters like alexander failed to conquer at box office quarterly profits us media giant timewarner
the firm which is now one of the biggest investors in google benefited
timewarner said fourth quarter sales rose to from
its profits were buoyed by one off gains which offset a
mixed fortunes time warner said on friday that it now
but its own internet business aol had has mixed fortunes
it lost subscribers in the fourth quarter profits were lower in the preceding quarters
however the company said aol underlying profit before exceptional items rose on the back internet
it hopes to increase subscribers by offering the online service to timewarner customers and will
timewarner also has to restate and results following a probe by the securities exchange commission
film flops time warner fourth quarter profits were slightly better than analysts expectations
film profits slump helped box office alexander sharp contrast year earlier final film rings trilogy
for the full year timewarner posted a profit of up from its performance while revenues
our financial performance was strong meeting or exceeding all of our full year and greatly
for timewarner is projecting operating earnings growth of around and also expects higher revenue
restating revenues timewarner is to restate its accounts as part of efforts to resolve
it has already offered to pay to settle that by
the company said it was unable to estimate the amount it needed to which previously
it intends to adjust the way it for a deal with german music publisher bertelsmann
it will now book the sale of its stake in aol europe as a loss
```

8.6 Testing it with ROUGE metric

Having testing the results analytically, we now move on to test it with a metric known as rouge metric. It stands for "**Recall-Oriented Understudy for Gisting Evaluation**". This metric is used for evaluation of automatic summarization tasks. It works by comparing the model(system) generated summary against the original summary(typically produced by a human). It works by finding the overlapping words(from system generated summary) with the human generated summary. We evaluated this on test data set. We took an average of the scores of all the articles. The results which we got was around 0.55 which is almost the same when compared it with the referenced papers. The maximum score which we got was 0.69.

```
def find_rouge(text,org_sum_text):
#   arr_text=extract_sentences_from_paragraph(text)
  vec=test_vectorize(text)
#   print(len(vec))
  results=loaded_model.predict(vec)
  for res in results:
    word_idx=sorted(range(len(res)), key=lambda i: res[i],
reverse=True)[:15]
    for i in word_idx:
      res[i]=1
  sent_sum=' '
  for res,sent in zip(results,text):
    sent=str(make_text_35(sent))
    s_indv=' '.join([word for idx,word in enumerate(sent.split(" "))
if(res[idx] == 1)])
    sent_sum=str(sent_sum)+str(s_indv)+str(',')
#     print(s_indv)
  #now computing rouge metric
#   print(sent_sum)
  evaluator = rouge.Rouge(metrics=['rouge-1','rouge-2'])
  scores = evaluator.get_scores(org_sum_text, sent_sum)
  return scores[0]['rouge-1']['f']
```

8.7 Language and environment of implementation.

We used python programming language to write the entire code. This was specially chosen to keep in mind the rich libraries it provides. Several libraries were used, some of them are as follows :

- Tensorflow
- Keras
- Numpy
- Nltk
- Gensim etc

```
from __future__ import division
import tensorflow as tf
import numpy as np
import nltk
import pickle
import random
import csv
import os
from gensim.models import FastText,Word2Vec
from gensim.models.phrases import Phrases, Phraser
from gensim.parsing.preprocessing import remove_stopwords, strip_punctuation,
strip_non_alphanum
import nltk
from nltk import sent_tokenize,word_tokenize
# nltk.download('punkt')
# nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import urllib.request
import re
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import np_utils
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from keras.models import model_from_json
import warnings
import flask
import json
from flask import Flask, render_template, request
```

.

The training was done on cloud using google co laboratory because it provides GPUs free for personal use.

8.8 Development of UI

The next stage of our implementation involves development of user friendly user- interface. It should be clean and concise with low responsive time. Various html codes along with its styling was written to make a clean and concise UI.

8.9 Running the product on server

Flask is a micro framework for python. It is used to run the product on a server. It also aids in integration of various modules with its user interface. It is done through app routes. These are application URLs mapped to each module of the product. This is a decorator which tells which function(module) should be called for an application URL.

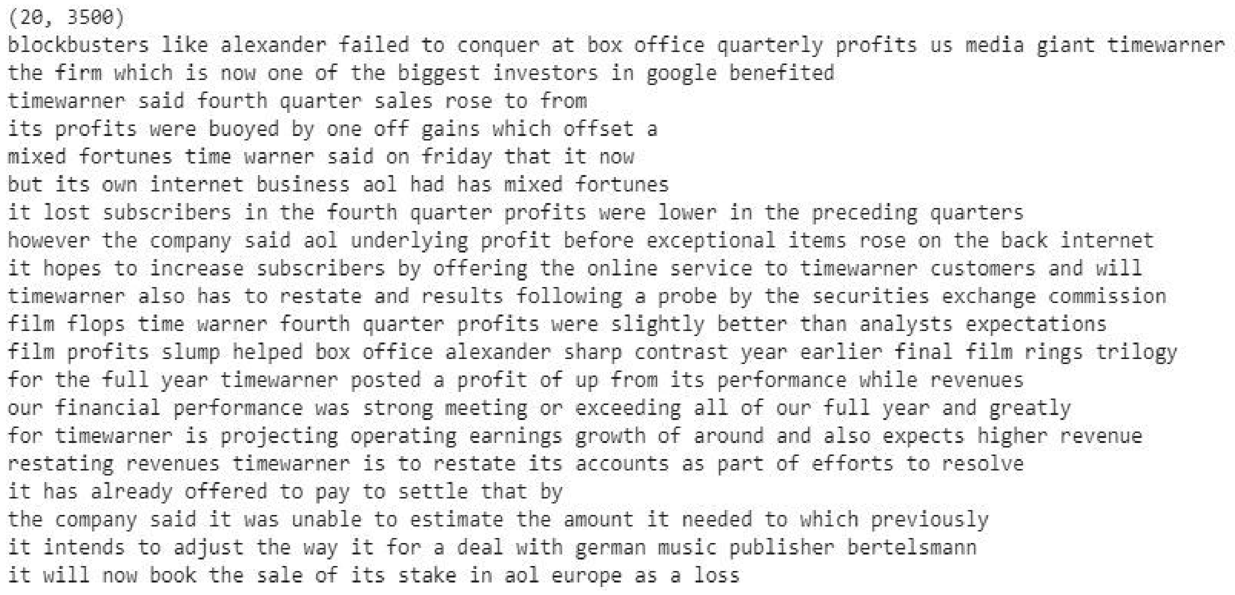
```
@app.route('/',methods=['POST','GET'])
def home():
   # url='http://news.bbc.co.uk/2/hi/business/4236959.stm'
   # print(predict(url))
   # url = request.form['projectFilepath']
   return flask.render_template('final.html')
# @app.route('/predict', methods=['GET','POST'])

@app.route('/result',methods = ['POST', 'GET'])
def result():
   if request.method == 'POST':
       result_url = request.form['url']
       result_text=request.form['text']
       # if(result_url==''):
       #     predict_text(result_text)#text predict
       # if(result_text==''):
       #     summary_text=predict(result_url)#url predict
       # print(res_text)
   return render_template("result.html",bot='hello world!!!!!!!!!!!!')#change
this

@app.route('/example',methods = ['POST', 'GET'])
def example():
   url='http://news.bbc.co.uk/2/hi/business/4236959.stm'
   print(predict(url))
   return render_template("result.html")#change this
```

CHAPTER 9

TESTING

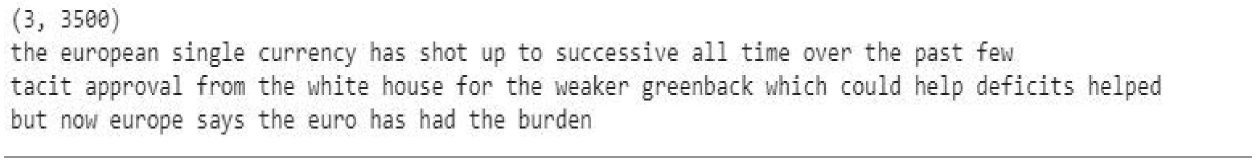
Testing for the project has been carried out in all stages of the various development life-cycle.

The first phase of the testing has been done with respect to the data set. We divided the data set into 80:20 ratio each for training and testing respectively. When tested it performed as it was supposed to perform. We used ROGUE metric to evaluate the performance. It performed well when compared with papers referred. Although the results were not grammatically correct but then one could easily interpret the summarized article.



```
(20, 3500)
blockbusters like alexander failed to conquer at box office quarterly profits us media giant timewarner
the firm which is now one of the biggest investors in google benefited
timewarner said fourth quarter sales rose to from
its profits were buoyed by one off gains which offset a
mixed fortunes time warner said on friday that it now
but its own internet business aol had has mixed fortunes
it lost subscribers in the fourth quarter profits were lower in the preceding quarters
however the company said aol underlying profit before exceptional items rose on the back internet
it hopes to increase subscribers by offering the online service to timewarner customers and will
timewarner also has to restate and results following a probe by the securities exchange commission
film flops time warner fourth quarter profits were slightly better than analysts expectations
film profits slump helped box office alexander sharp contrast year earlier final film rings trilogy
for the full year timewarner posted a profit of up from its performance while revenues
our financial performance was strong meeting or exceeding all of our full year and greatly
for timewarner is projecting operating earnings growth of around and also expects higher revenue
restating revenues timewarner is to restate its accounts as part of efforts to resolve
it has already offered to pay to settle that by
the company said it was unable to estimate the amount it needed to which previously
it intends to adjust the way it for a deal with german music publisher bertelsmann
it will now book the sale of its stake in aol europe as a loss
```

Figure: Snippet of the returned summary when a URL was given



```
(3, 3500)
the european single currency has shot up to successive all time over the past few
tacit approval from the white house for the weaker greenback which could help deficits helped
but now europe says the euro has had the burden
```

Figure: Snippet of the returned summary

When tested with URL and texts which were different from the data set, then also the model performed considerably well. One could easily interpret the summarized article.

For the word embedding model we tested it to check it performance. Different parameter were tried to arrive the best model.

Modules in which processing and cleaning of texts were performed was also tested. Different designs were tested to arrive at the most convenient design.

On integrating the model with the UI, we tested it. We tested the various modules of flask framework. It was verified to check the results if it corresponds to model results. It performed well and worked as it was supposed to do.

CHAPTER 10

RESULTS AND DISCUSSIONS

The following are outcomes and results of our project:

1. With data being the new oil and with such humongous amount of data being circulated and generated every day, there is always a need to develop machine learning algorithm that can automatically shorten long articles and deliver accurate and short summaries which is easy to store and can be easily interpreted.
2. Our model is one such contribution to this. It performs well when the result are interpreted. Although this is grammatically not correct but still the results were able to pass the intended message.
3. When compared with other existing models which are proven to perform good, it performs fairly good. We have used ROGUE metric for our evaluation purpose. It works by comparing the model(system) generated summary against the original summary(typically produced by a human). It works by finding the overlapping words(from system generated summary) with the human generated summary.
4. We have also added an interactive user interface which benefits the user. We have added a feature in which user enters a URL of the article for which he wants the summary. There is also an option in which a user may write the article in the given text box and our model will generate the summaries for it.

CHAPTER 11

SNAPSHOTS

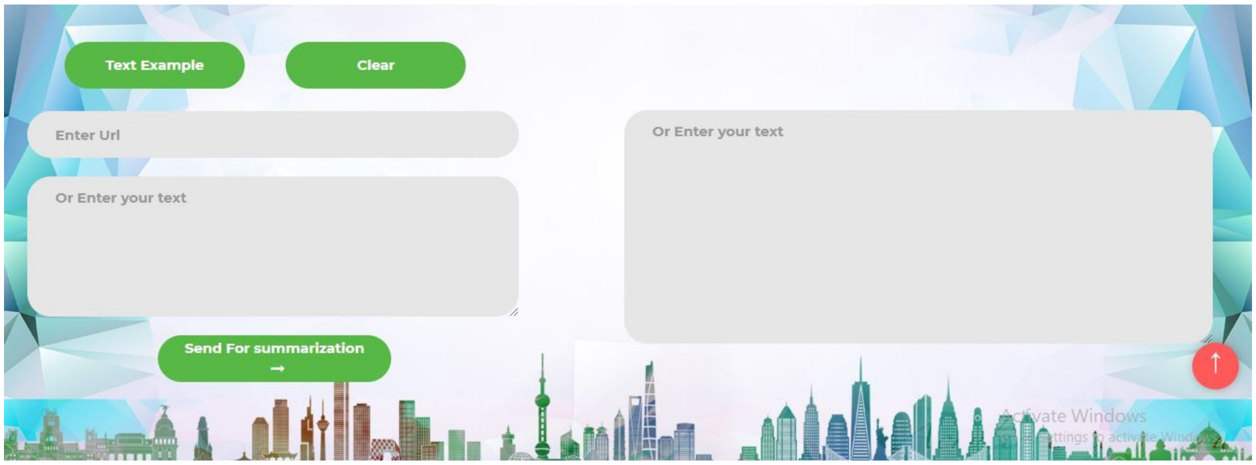
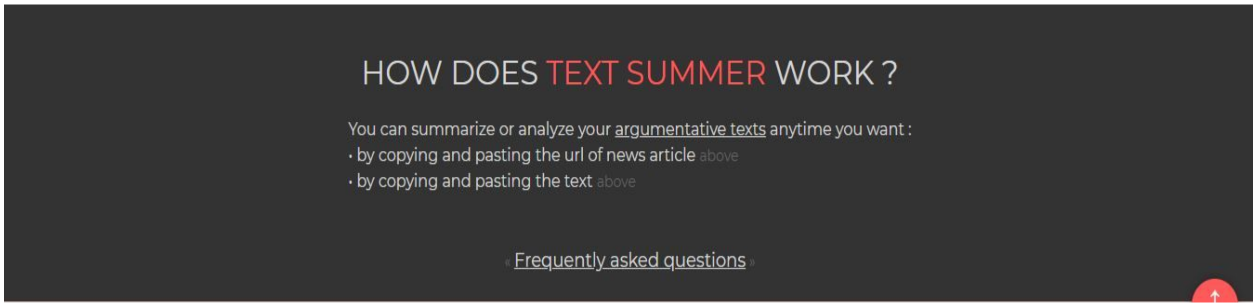


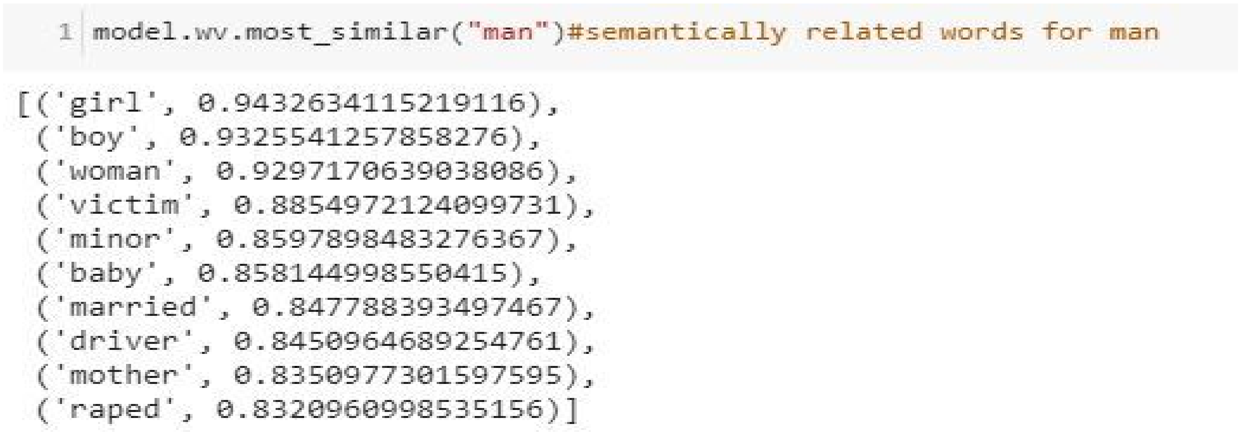
Figure: An interactive UI which allows user to enter the URL of the article or to write the text


HOW DOES TEXT SUMMER WORK ?

You can summarize or analyze your argumentative texts anytime you want :
· by copying and pasting the url of news article above
· by copying and pasting the text above

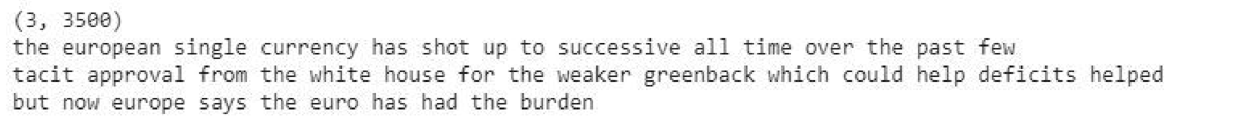
« Frequently asked questions »

Figure: An interactive UI


```
model.wv.most_similar("man")#semantically related words for man
```

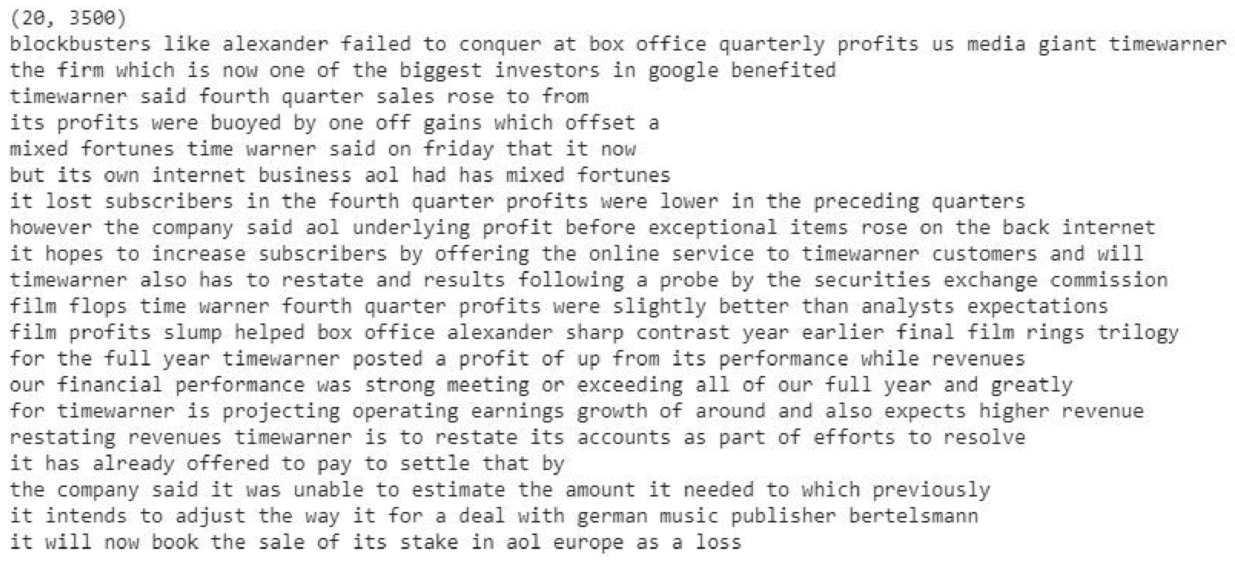
```
[('girl', 0.9432634115219116),
 ('boy', 0.9325541257858276),
 ('woman', 0.9297170639038086),
 ('victim', 0.8854972124099731),
 ('minor', 0.8597898483276367),
 ('baby', 0.858144998550415),
 ('married', 0.847788393497467),
 ('driver', 0.8450964689254761),
 ('mother', 0.8350977301597595),
 ('raped', 0.8320960998535156)]
```

Figure : Output of the word embedding model



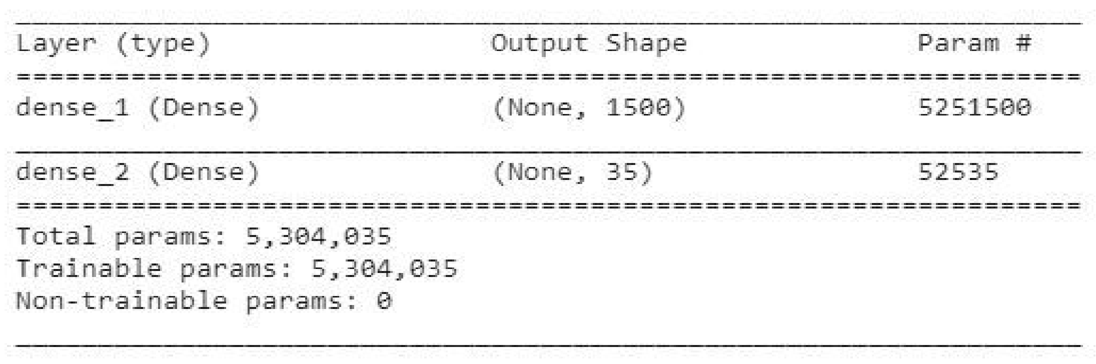
```
(3, 3500)
the european single currency has shot up to successive all time over the past few
tacit approval from the white house for the weaker greenback which could help deficits helped
but now europe says the euro has had the burden
```

Figure : Output for a news article



```
(20, 3500)
blockbusters like alexander failed to conquer at box office quarterly profits us media giant timewarner
the firm which is now one of the biggest investors in google benefited
timewarner said fourth quarter sales rose to from
its profits were buoyed by one off gains which offset a
mixed fortunes time warner said on friday that it now
but its own internet business aol had has mixed fortunes
it lost subscribers in the fourth quarter profits were lower in the preceding quarters
however the company said aol underlying profit before exceptional items rose on the back internet
it hopes to increase subscribers by offering the online service to timewarner customers and will
timewarner also has to restate and results following a probe by the securities exchange commission
film flops time warner fourth quarter profits were slightly better than analysts expectations
film profits slump helped box office alexander sharp contrast year earlier final film rings trilogy
for the full year timewarner posted a profit of up from its performance while revenues
our financial performance was strong meeting or exceeding all of our full year and greatly
for timewarner is projecting operating earnings growth of around and also expects higher revenue
restating revenues timewarner is to restate its accounts as part of efforts to resolve
it has already offered to pay to settle that by
the company said it was unable to estimate the amount it needed to which previously
it intends to adjust the way it for a deal with german music publisher bertelsmann
it will now book the sale of its stake in aol europe as a loss
```

Figure: Output for a news article



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 1500)              5251500   
_________________________________________________________________
dense_2 (Dense)              (None, 35)                52535     
=================================================================
Total params: 5,304,035
Trainable params: 5,304,035
Non-trainable params: 0
_________________________________________________________________
```

Figure: Architecture of machine learning model

CHAPTER 12

CONCLUSION

Automatic text summarization is the process of reducing the number of words and its content in such a way that it only has important points and is able to convey the message it was supposed to do. Our application is supported by an interactive user interface. The user interface allows the user to enter the URL of the news article and also allows the user to type his own text in the text box.

The model used by us is an extractive summarization which is just like highlighting the important points in the article. But this method is computationally **inexpensive** when compared with the other such models for both extractive and abstractive methods. Therefore, the proposed model could be used on mobile devices as well.

The results of our method are able to convey important points which each sentence is trying to convey. This is different from other extractive summarization methods, as the models there basically rank or classify each sentence as important or not important. The results may not be grammatically correct, but then they are able to convey the message the article is trying to convey.

We have used ROGUE metric to evaluate the summary. The results of these are fairly good against the existing systems which are proven to work good.

Although there are still scope for further improvements which can be done but then our model performs fairly well and is able to convey the message the article is trying to create.

CHAPTER 13
FURTHER ENHANCEMENTS

The model developed by us for extractive summarization is performing well but it is grammatically not correct, although it is able to convey important message which the article was trying to convey. There is a scope for improvement in terms of making it grammatically more sound. There is scope of adding recurrence in the neural network. This is almost similar to a pointer generator network which is proposed by one of the papers. But these approaches are hard to implement, and they require a lot of computation.

Currently we have used this just for English news, so there is a scope to extend this towards other language as well. To this we need a data set for other languages as well. We need to explore more on this.

CHAPTER 14

REFERENCES / BIBLIOGRAPHY

[1] - Kaikhah, K., 2004. Text summarization using neural networks. Faculty Publications-Computer Science.

[2] - Annah, M.E. and S. Mukherjee, 2014. A classification based summarization model for summarizing text documents. Int. J. Inform. Commun. Technol.

[3] - McKeown, K. and D.R. Radev, 1995. Generating summaries of multiple news articles. Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Jul. 09-13, Seattle, WA, USA

[4] - McKeown, K.R., R. Barzilay, D. Evans, V. Hatzivassiloglou and J.L. Klavans et al., 2002. Tracking and summarizing news on a daily basis with Columbia's Newsblaster. Proceedings of the 2nd International Conference on Human Language Technology Research, (LTR' 02), ACM

[5] - Lee, C.S., Z.W. Jian and L.K. Huang, 2005. A fuzzy ontology and its application to news summarization. IEEE Trans. Syst., Man Cybernet. Part B: Cybernet., 35: 859-880.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate

[7] Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization.

[8] A. See, P. J. Liu, and C. D. Manning, "Get to the point: Summarization with pointer-generator networks," in Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics.

[10] A. P. Siva kumar, Dr. P. Premchand and Dr. A. Govardhan, "Query Based Summarizer Based on Similarity of Sentences and Word Frequency", International Journal of Data Mining & Knowledge Management Process , vol.1, no.3, May 2011.

BIBLIOGRAPHY

- https://towardsdatascience.com/a-quick-introduction-to-text-summarization-in-machine-learning-3d27ccf18a9f
- https://medium.com/jatana/unsupervised-text-summarization-using-sentence-embeddings-adb15ce83db1
- https://www.analyticsvidhya.com/blog/2018/11/introduction-text-summarization-textrank-python/
- https://github.com/icoxfog417/awesome-text-summarization
- https://medium.com/jatana/unsupervised-text-summarization-using-sentence-embeddings-adb15ce83db1
- https://towardsdatascience.com/word-embedding-with-word2vec-and-fasttext-a209c1d3e12c
- https://radimrehurek.com/gensim/models/fasttext.html
- https://www.quora.com/What-is-the-main-difference-between-word2vec-and-fastText
- https://fasttext.cc/
- https://fasttext.cc/docs/en/crawl-vectors.html
- https://github.com/facebookresearch/fastText

USER MANUAL

Machine learning:- Different algorithms that give computers the ability to learn and improve from experience without being explicitly programmed.

Deep learning:- Deep Learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks.

Rnn:- Recurrent neural network. Recurrent because they perform the same task for every element of a sequence, with the output depending on previous computations.

LSTM:- Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies

.

Encoder:- Encoder RNN first reads in the source string word by word, encoding the information in a hidden state(thought vector).

Decoder:- Decoder is another RNN which learns to decode the thought vector into the output sequence.

OOV- Out of vocabulary words.

CIA Triad - Confidentiality, Integrity, Availability. This model is designed to guide policies for information security.