# A Thesis/Project/Dissertation Report

on

# Text Summarization using Machine Learning

***Submitted in partial fulfillment of the***

***requirement for the award of the degree of***

Bachelor of Technology in Computer Science and Engineering


**Under The Supervision of**

**Name of Supervisor : Dr A . John**

**Professor, Galgotias University**

Submitted By

**Ateev Mishra - 18SCSE1010585**

**Kaushlendra Kumar - 18SCSE1010430**

**SCHOOL OF COMPUTING SCIENCE AND ENGINEERING DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**GALGOTIAS UNIVERSITY, GREATER NOIDA**

**INDIA**

**DECEMBER, 2021**

**SCHOOL OF COMPUTING SCIENCE AND ENGINEERING**
**GALGOTIAS UNIVERSITY, GREATER NOIDA**

## CANDIDATE'S DECLARATION

We hereby certify that the work which is being presented in the thesis/project/dissertation, entitled **"TEXT SUMMARIZATION USING MACHINE LEARNING"** in partial fulfillment of the requirements for the award of the Bachelor of Technology in Computer Science and Engineering submitted in the School of Computing Science and Engineering of Galgotias University, Greater Noida, is an original work carried out during the period of September, 2021 to December, 2021, under the supervision of Dr A. John , Professor, Department of Computer Science and Engineering, of School of Computing Science and Engineering , Galgotias University, Greater Noida.

The matter presented in the thesis/project/dissertation has not been submitted by me/us for the award of any other degree of this or any other places.

Ateev Mishra, 18SCSE1010585
Kaushlendra Kumar, 18SCSE1010430

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Dr A. John
Professor, SCSE, GU

## CERTIFICATE

The Final Thesis/Project/ Dissertation Viva-Voce examination of Ateev Mishra:18SCSE1010585; Kaushlendra Kumar: 18SCSE1010430, has been held on _____and his/her work is recommended for the award of Bachelor of Technology in Computer Science and Engineering:

**Signature of Examiner(s)** **Signature of Supervisor(s)**

**Signature of Project Coordinator** **Signature of Dean**

Date: December, 2021

Place: Greater Noida

## Acknowledgement

# ABSTRACT

Text Summarization involves condensing a piece of text into a shorter version, reducing the size of the original text while preserving key information and the meaning of the content. Since manual text synthesis is a long and generally laborious task, task automation is gaining in popularity and therefore a strong motivation for academic research.

We will use “Abstractive Summarization”, This is a very interesting approach as, we will generate new sentences from the original text.

This way of summarization is way better than “Extractive Summarization” which summarizes a given piece of text by using the sentences already present in the text.

Our objective is to build a text summarizer where the input is a long sequence of words (in a text body), and the output is a short summary (which is a sequence as well). So, we will model this as a Many-to-Many Sequence to Sequence problem(S2S).

Hence we will utilize the power of Natural Language Processing(NLP) which is a part of Deep Learning to resolve and complete this task.

Contents

# CHAPTER 1: INTRODUCTION

## 1.INTRODUCTION

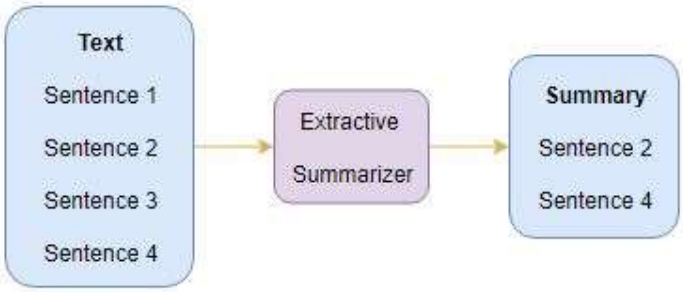
Broadly there are two different approaches used for text summarization:

1)Extractive Summarization

2)Abstractive Summarization

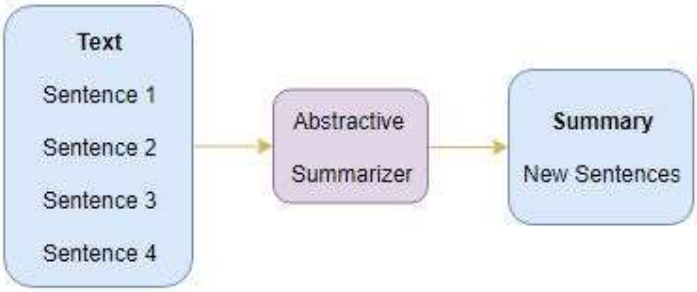
1) Extractive Summarization

The name iself tells what this approach does. We must first identify the important sentences or phrases from the original text and then extract only those ones from the text.




2) Abstractive Summarization

This is a very interesting approach. Here, we actually generate new sentences from the original text. This is in contrast to the extractive approach we saw earlier where we used only the sentences that were already present in the original text.




We used “Abstractive Summarization” approach to build a text summarizer where the input is a long sequence of words (in a text body), and the output is a small summary (which is a sequence as well). So, we will model this as a Many-to-Many Sequence to Sequence problem(S2S).

We build a Seq2Seq(Sequence to Sequence) model on any problem which involves sequential information. This includes Sentiment classification, Neural Machine Translation, and Named Entity Recognition – some very common applications of sequential information.

In the case of Neural Machine Translation, the input is a text in one language and the output is also a text in another language. In the Named Entity Recognition, the input is a sequence of words and the output is a sequence of tags for every word in the input sequence.

Our objective is to build a text summarizer where the input is a long sequence of words (in a text body), and the output is a short summary (which is a sequence as well). So, we can model this as a Many-to-Many Seq2Seq problem.

# CHAPTER 2.LITERATURE REVIEW

Automatic text summarization is the task of producing a concise and fluent summary while preserving key information content and overall meaning.

Along with the growth of the internet and big data, making people overwhelmed by the large information and documents on the internet. This triggers the desire of many researchers to develop a technological approach that can summarize texts automatically.

The area of text summarization research has been studied since the mid-20$^{th}$ century, which was first discussed openly by Lun (1958) with a statistical technique namely word frequency diagrams. Many different approaches have been created to date. Based on the number of the document, there is single and multi-document summarization. Meanwhile, based on the summary results there are the extractive and abstractive results.

During the times of the pandemic, many new things have come to pass. In these times we have come to realize the importance of machine learning and how it can help multiple organisations, companies and hospitals in these times when the number of patients diagnosed with COVID19 has far exceed the predicted numbers and even after one year, the pandemic has not stopped its rampage across the globe. The Text Summarization has wide applications it can be used in hospitals where huge number of patients come in every day and is reading each and every person detail is too much for even a group of people

doing the job. This technology will be helpful in these times not only for the doctors but also for the patients. Doctors will be able to get an overview of the patients health by simply reading a summary of it , and the patient would be able to know about important elements in his/her reports by just reading few lines of necessary information. Using “Text Summarization” will also reduce the huge queues of patients waiting to submit their forms to the required department in the hospital and thus the process of submitting and reading the forms can be accelerated .

# CHAPTER 3. REQUIRED TOOLS

## 3.1 Minimum Hardware requirements –

1)CPU: 2.0 GHz Quad core processor

2)RAM: 8GB

3)GPU: 3 GB VRAM

## 3.2 Minimum Software requirements –

1)Operating System: Windows 7

2)Platform: Google Colab/ Jupyter Notebook

3) Python (3.7.4 used)

## 3.3 Libraries required -

1) Numpy (version 1.16.5)

2) Keras (version 2.3.1)

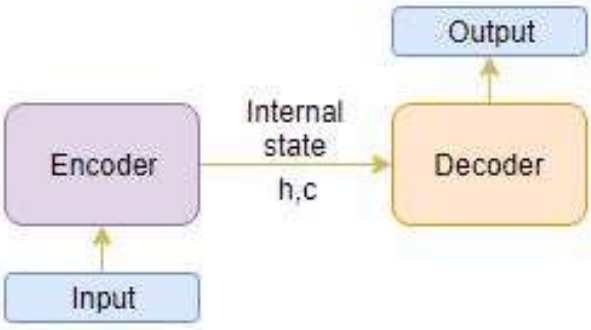
3) Tensorflow (Keras also uses TensorFlow in backend) (version 2.0.0)

4) Matplotlib (version 3.1.1)

5) Pandas (version 0.25.1)

# CHAPTER 4. PROCEDURE

We follow the Encoder-Decoder architecture. This architecture is mainly used to solve the sequence-to-sequence (Seq2Seq) problem where the input and output sequences are of different lengths.




Generally, variants of Recurrent Neural Networks (RNNs), i.e. Gated Recurrent Neural Network (GRU) or Long Short Term Memory (LSTM), are preferred as the encoder and decoder components. This is because they are capable of capturing long term dependencies by overcoming the problem of vanishing gradient.

We can set up the Encoder-Decoder in 2 phases:

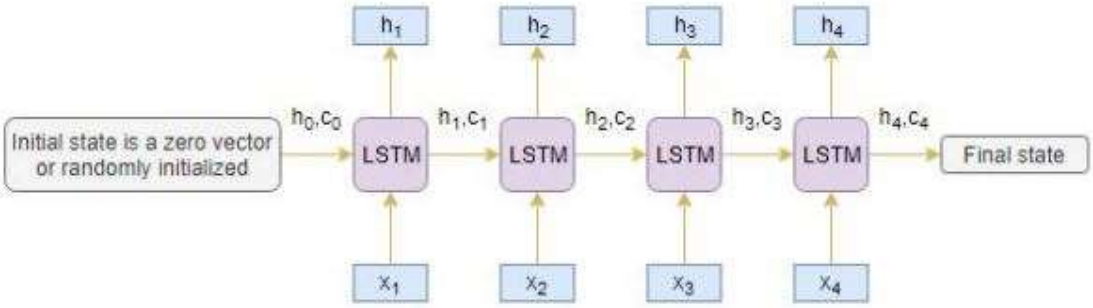
1.Training Phase
2.Inference Phase

## 1)Training Phase

In the training phase, we will first set up the encoder and decoder. We will then train the model to predict the target sequence offset by one timestep. Let us see in detail on how to set up the encoder and decoder.

Encoder

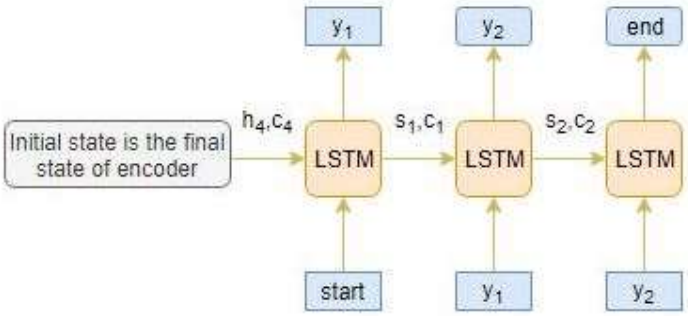
An Encoder Long Short Term Memory model (LSTM) reads the entire input sequence wherein, at each timestep, one word is fed into the encoder. It then processes the information at every timestep and captures the contextual information present in the input sequence.




The hidden state (hi) and cell state (ci) of the last time step are used to initialize the decoder.

Decoder

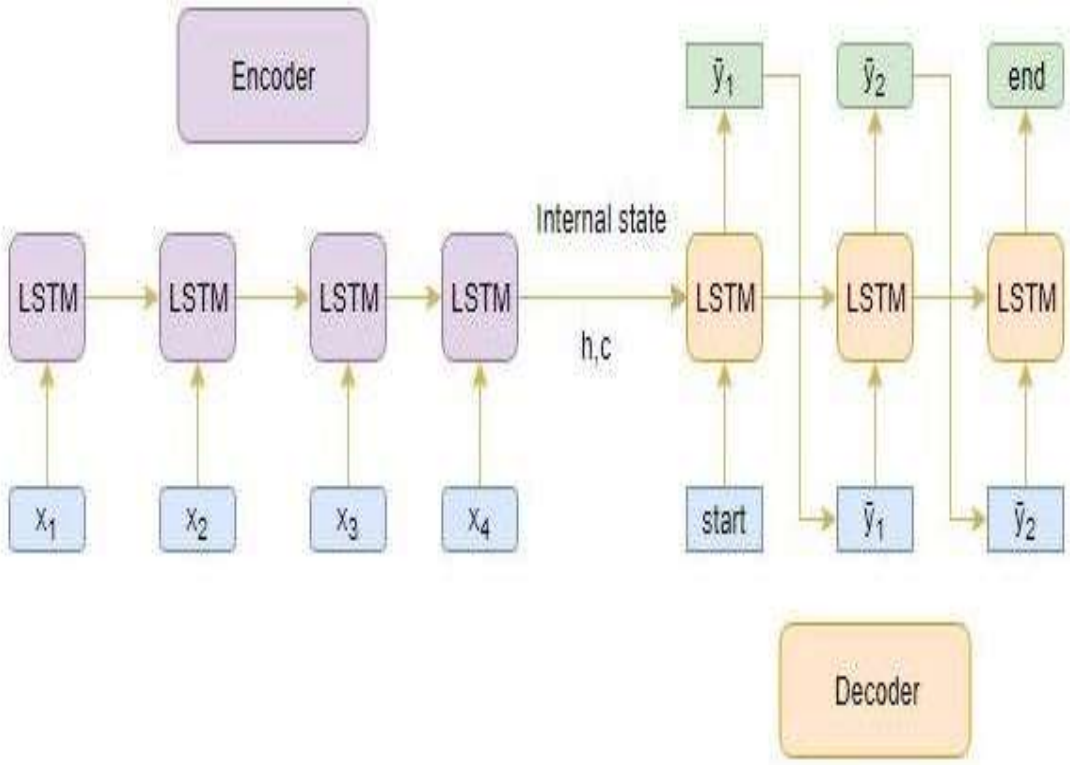
The decoder is also an LSTM network which reads the entire target sequence word-by-word and predicts the same sequence offset by one timestep. The decoder is trained to predict the next word in the sequence given the previous word.




“start” and “end” are the special tokens which are added to the target sequence before feeding it into the decoder. The target sequence is unknown while decoding the test sequence. So, we start predicting the target sequence by passing the first word into the decoder which would be always the <start> token. And the <end> token signals the end of the sentence.

## 2)Inference Phase

After training, the model is tested on new source sequences for which the target sequence is unknown. So, we need to set up the inference architecture to decode a test sequence.



Decoding the Test Sequence:

1)Encode the entire input sequence and initialize the decoder with internal states of the encoder .
2)Pass <start> token as an input to the decoder .
3)Run the decoder for one timestep with the internal states .
4)The output will be the probability for the next word. The word with the maximum probability will be selected .
5)Pass the sampled word as an input to the decoder in the next timestep and update the internal states with the current time step .
6)Repeat steps 3 – 5 until we generate <end> token or hit the maximum length of the target sequence .

## 3)The Attention Mechanism

As useful as this encoder-decoder architecture is, there are certain limitations that come with it.

i)The encoder converts the entire input sequence into a fixed length vector and then the decoder predicts the output sequence. This works only for short sequences since the decoder is looking at the entire input sequence for the prediction .

ii)Here comes the problem with long sequences. It is difficult for the encoder to memorize long sequences into a fixed length vector .

To resolve this problem the concept of attention mechanism comes into the picture.

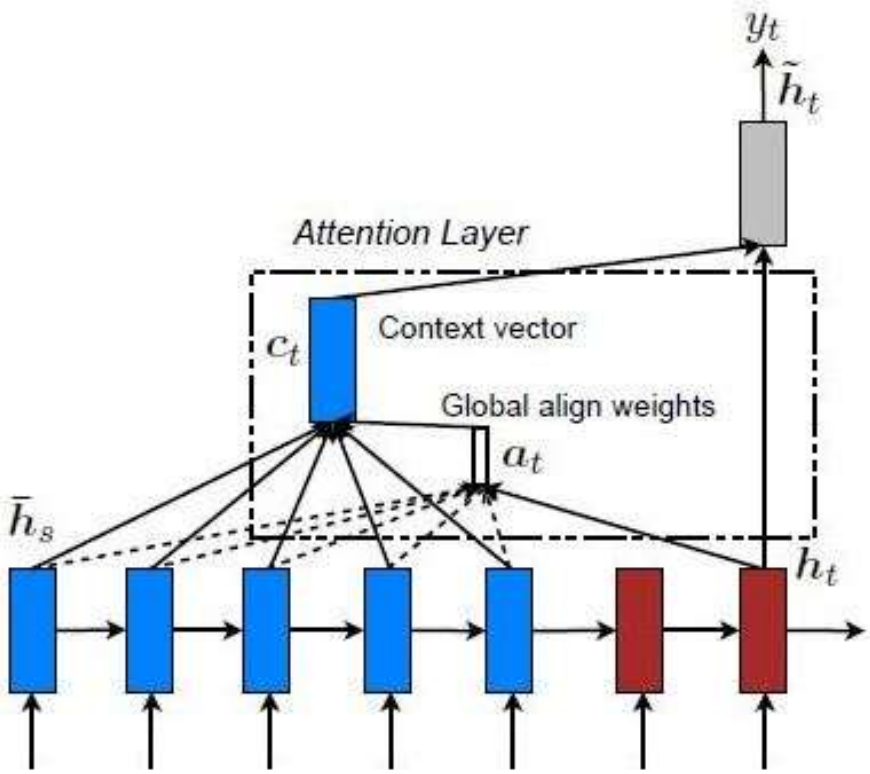
Instead of looking at all the words in the source sequence, we can increase the importance of specific parts of the source sequence that result in the target sequence. This is the basic idea behind the use of the attention mechanism.

There are 2 different classes of attention mechanism depending on the way the attended context vector is derived:

1)Global Attention
2)Local Attention

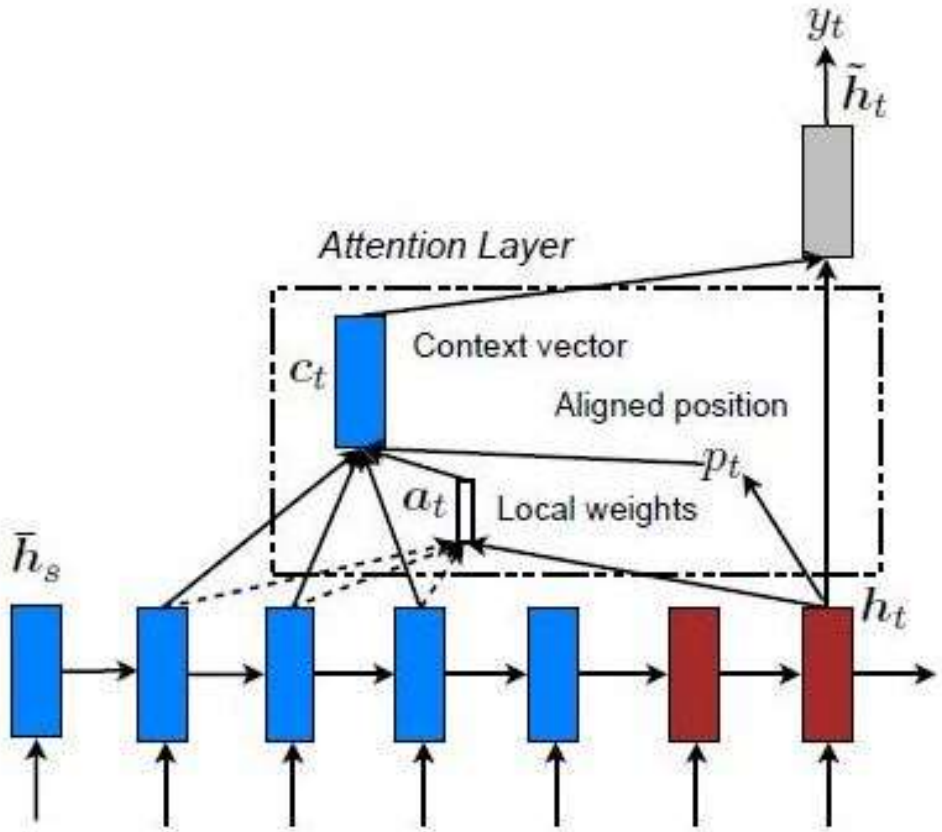
1)Global Attention

Here, the attention is placed on all the source positions. In other words, all the hidden states of the encoder are considered for deriving the attended context vector.



2)Local Attention

Here, the attention is placed on only a few source positions. Only a few hidden states of the encoder are considered for deriving the attended context vector.




In our project we will be using the Global Attention Mechanism.

## Working of Attention Mechanism

- The encoder outputs the hidden state ($\mathbf{h_j}$) for every time step **j** in the source sequence
- Similarly, the decoder outputs the hidden state ($\mathbf{s_i}$) for every time step **i** in the target sequence
- We compute a score known as an **alignment score ($\mathbf{e_{ij}}$)** based on which the source word is aligned with the target word using a score function. The alignment score is computed from the source hidden state $\mathbf{h_j}$ and target hidden state $\mathbf{s_i}$ using the score function. This is given by:

$$\mathbf{e_{ij} = score\ (s_i, h_j\ )}$$

where $\mathbf{e_{ij}}$ denotes the alignment score for the target timestep **i** and source time step **j**.

- We normalize the alignment scores using softmax function to retrieve the attention weights ($\mathbf{a_{ij}}$):

$$a_{ij} = e^{eij} / \sum_{k=1}^{Tx} e^{eik}$$

- We compute the linear sum of products of the attention weights $\mathbf{a_{ij}}$ and hidden states of the encoder $\mathbf{h_j}$ to produce the attended context vector ($\mathbf{C_i}$):

$$C_i = \sum_{j=1}^{Tx} a_{ij}\, h_j$$

- The attended context vector and the target hidden state of the decoder at timestep **i** are concatenated to produce an attended hidden vector $\mathbf{S_i}$

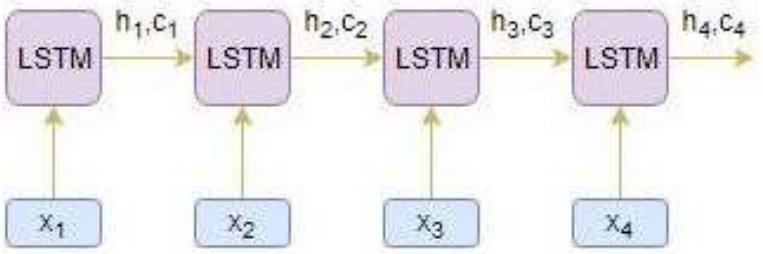
$$\mathbf{S_i = concatenate([s_i; C_i])}$$

- The attended hidden vector $\mathbf{S_i}$ is then fed into the dense layer to produce $\mathbf{y_i}$

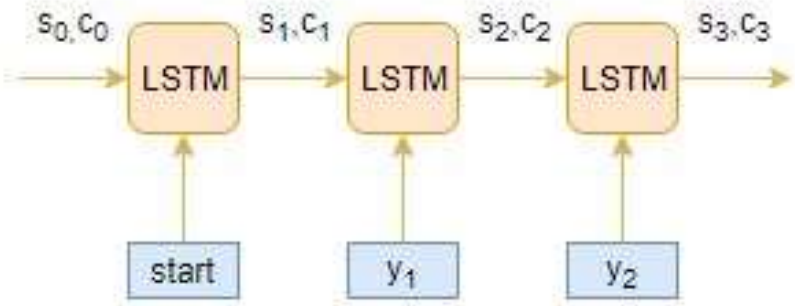
$$\mathbf{y_i = dense(S_i)}$$

Let's understand the above attention mechanism steps with the help of an example. Consider the source sequence to be [$x_1$, $x_2$, $x_3$, $x_4$] and target sequence to be [$y_1$, $y_2$].

- The encoder reads the entire source sequence and outputs the hidden state for every timestep, say **$h_1$, $h_2$, $h_3$, $h_4$**




- The decoder reads the entire target sequence offset by one timestep and outputs the hidden state for every timestep, say **$s_1$, $s_2$, $s_3$**



# CHAPTER 5. IMPLEMENTATION

### Understanding the Problem Statement

Customer reviews can often be long and descriptive. Analyzing these reviews manually, is really time-consuming. This is where the brilliance of Natural Language Processing can be applied to generate a summary for long reviews.

Our objective here is to generate a summary for the Amazon Fine Food reviews using the abstraction-based approach we learned about above.

### Custom Attention Layer

Keras does not officially support attention layer. So, we can either implement our own attention layer or use a third-party implementation. We will go with the latter option for this article.

Let's import it into our environment:

In [0]:
```
from attention import AttentionLayer
```

### Import the Libraries

In [0]:
```
import numpy as np
import pandas as pd
import re
from bs4 import BeautifulSoup
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from nltk.corpus import stopwords
from tensorflow.keras.layers import Input, LSTM, Embedding, Dense, Concatenate, TimeDistributed
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping
import warnings
pd.set_option("display.max_colwidth", 200)
warnings.filterwarnings("ignore")
```

Using TensorFlow as backend.

### Read the dataset

This dataset consists of reviews of fine foods from Amazon. The data spans a period of more than 10 years, including all ~500,000 reviews up to October 2012. These reviews include product and user information, ratings, plain text review, and summary. It also includes reviews from all other Amazon categories.

We'll take a sample of 100,000 reviews to reduce the training time of our model.

```
In [0]: data=pd.read_csv("../input/amazon-fine-food-reviews/Reviews.csv",nrows=100000)
```

## Drop Duplicates and NA values

```
In [0]: data.drop_duplicates(subset=['Text'],inplace=True)#dropping duplicates
        data.dropna(axis=0,inplace=True)#dropping na
```

## Information about dataset

Let us look at datatypes and shape of the dataset

```
In [0]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 88421 entries, 0 to 99999
Data columns (total 10 columns):
Id                      88421 non-null int64
ProductId               88421 non-null object
UserId                  88421 non-null object
ProfileName             88421 non-null object
HelpfulnessNumerator    88421 non-null int64
HelpfulnessDenominator  88421 non-null int64
Score                   88421 non-null int64
Time                    88421 non-null int64
Summary                 88421 non-null object
Text                    88421 non-null object
dtypes: int64(5), object(5)
memory usage: 7.4+ MB
```

## Preprocessing

Performing basic pre processing steps is very important before we get to the model building part. Using messy and uncleaned text data is a potentially disastrous move. So in this step, we will drop all the unwanted symbols, characters, etc. from the text that do not affect the objective of our problem.

Here is the dictionary that we will use for expanding the contractions:

In [0]:

```
contraction_mapping = {"ain't": "is not", "aren't": "are not","can't": "cannot", "'cause": "because", "could've": "could have", "couldn't": "could nc
                       "didn't": "did not",  "doesn't": "does not", "don't": "do not", "hadn't": "had not", "hasn't": "has not", "haven't": "have
                       "he'd": "he would","he'll": "he will", "he's": "he is", "how'd": "how did", "how'd'y": "how do you", "how'll": "how will",
                       "I'd": "I would", "I'd've": "I would have", "I'll": "I will", "I'll've": "I will have","I'm": "I am", "I've": "I have", "i
                       "i'd've": "i would have", "i'll": "i will",  "i'll've": "i will have","i'm": "i am", "i've": "i have", "isn't": "is not",
                       "it'd've": "it would have", "it'll": "it will", "it'll've": "it will have","it's": "it is", "let's": "let us", "ma'am": "m
                       "mayn't": "may not", "might've": "might have","mightn't": "might not","mightn't've": "might not have", "must've": "must ha
                       "mustn't": "must not", "mustn't've": "must not have", "needn't": "need not", "needn't've": "need not have","o'clock": "of
                       "oughtn't": "ought not", "oughtn't've": "ought not have", "shan't": "shall not", "sha'n't": "shall not", "shan't've": "sha
                       "she'd": "she would", "she'd've": "she would have", "she'll": "she will", "she'll've": "she will have", "she's": "she is",
                       "should've": "should have", "shouldn't": "should not", "shouldn't've": "should not have", "so've": "so have","so's": "so a
                       "this's": "this is","that'd": "that would", "that'd've": "that would have", "that's": "that is", "there'd": "there would",
                       "there'd've": "there would have", "there's": "there is", "here's": "here is","they'd": "they would", "they'd've": "they wc
                       "they'll": "they will", "they'll've": "they will have", "they're": "they are", "they've": "they have", "to've": "to have",
                       "wasn't": "was not", "we'd": "we would", "we'd've": "we would have", "we'll": "we will", "we'll've": "we will have", "we'r
                       "we've": "we have", "weren't": "were not", "what'll": "what will", "what'll've": "what will have", "what're": "what are",
                       "what's": "what is", "what've": "what have", "when's": "when is", "when've": "when have", "where'd": "where did", "where's
                       "where've": "where have", "who'll": "who will", "who'll've": "who will have", "who's": "who is", "who've": "who have",
                       "why's": "why is", "why've": "why have", "will've": "will have", "won't": "will not", "won't've": "will not have",
                       "would've": "would have", "wouldn't": "would not", "wouldn't've": "would not have", "y'all": "you all",
                       "y'all'd": "you all would","y'all'd've": "you all would have","y'all're": "you all are","y'all've": "you all have",
                       "you'd": "you would", "you'd've": "you would have", "you'll": "you will", "you'll've": "you will have",
                       "you're": "you are", "you've": "you have"}
```

We will perform the below preprocessing tasks for our data:

1.Convert everything to lowercase

2.Remove HTML tags

3.Contraction mapping

4.Remove (‘s)

5.Remove any text inside the parenthesis ( )

6.Eliminate punctuations and special characters

7.Remove stopwords

8.Remove short words

Let's define the function:

In [0]:
```
stop_words = set(stopwords.words('english'))

def text_cleaner(text,num):
    newString = text.lower()
    newString = BeautifulSoup(newString, "lxml").text
    newString = re.sub(r'\([^)]*\)', '', newString)
    newString = re.sub('"','', newString)
    newString = ' '.join([contraction_mapping[t] if t in contraction_mapping else t for t in newString.split(" ")])
    newString = re.sub(r"'s\b","",newString)
    newString = re.sub("[^a-zA-Z]", " ", newString)
    newString = re.sub('[m]{2,}', 'mm', newString)
    if(num==0):
        tokens = [w for w in newString.split() if not w in stop_words]
    else:
        tokens=newString.split()
    long_words=[]
    for i in tokens:
        if len(i)>1:                                 #removing short word
            long_words.append(i)
    return (" ".join(long_words)).strip()
```

In [0]:
```
#call the function
cleaned_text = []
for t in data['Text']:
    cleaned_text.append(text_cleaner(t,0))
```

Let us look at the first five preprocessed reviews

In [0]:
```
cleaned_text[:5]
```

Out[0]:
```
['bought several vitality canned dog food products found good quality product looks like stew processed meat smells better labrador finicky appreciat
es product better',
 'product arrived labeled jumbo salted peanuts peanuts actually small sized unsalted sure error vendor intended represent product jumbo',
 'confection around centuries light pillowy citrus gelatin nuts case filberts cut tiny squares liberally coated powdered sugar tiny mouthful heaven c
hewy flavorful highly recommend yummy treat familiar story lewis lion witch wardrobe treat seduces edmund selling brother sisters witch',
 'looking secret ingredient robitussin believe found got addition root beer extract ordered made cherry soda flavor medicinal',
 'great taffy great price wide assortment yummy taffy delivery quick taffy lover deal']
```

In [0]:
```
#call the function
cleaned_summary = []
for t in data['Summary']:
    cleaned_summary.append(text_cleaner(t,1))
```

Let us look at the first 10 preprocessed summaries

In [0]:
```
cleaned_summary[:10]
```

Out[0]:
```
['good quality dog food',
 'not as advertised',
 'delight says it all',
 'cough medicine',
 'great taffy',
 'nice taffy',
 'great just as good as the expensive brands',
 'wonderful tasty taffy',
 'yay barley',
 'healthy dog food']
```

In [0]:
```
data['cleaned_text']=cleaned_text
data['cleaned_summary']=cleaned_summary
```

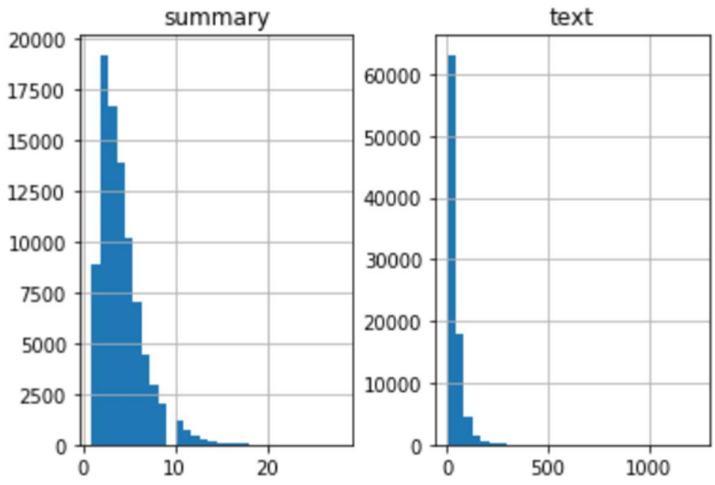
## Drop empty rows

```
In [0]: data.replace('', np.nan, inplace=True)
        data.dropna(axis=0,inplace=True)
```

## Understanding the distribution of the sequences

Here, we will analyze the length of the reviews and the summary to get an overall idea about the distribution of length of the text. This will help us fix the maximum length of the sequence:

```
In [0]: import matplotlib.pyplot as plt

        text_word_count = []
        summary_word_count = []

        # populate the lists with sentence lengths
        for i in data['cleaned_text']:
              text_word_count.append(len(i.split()))

        for i in data['cleaned_summary']:
              summary_word_count.append(len(i.split()))

        length_df = pd.DataFrame({'text':text_word_count, 'summary':summary_word_count})

        length_df.hist(bins = 30)
        plt.show()
```




We can fix the maximum length of the summary to 8 since that seems to be the majority summary length.

Let us understand the proportion of the length of summaries below 8

In [0]:
```
cnt=0
for i in data['cleaned_summary']:
    if(len(i.split())<=8):
        cnt=cnt+1
print(cnt/len(data['cleaned_summary']))
```

0.9424907471335922

We observe that 94% of the summaries have length below 8. So, we can fix maximum length of summary to 8.

Let us fix the maximum length of review to 30

In [0]:
```
max_text_len=30
max_summary_len=8
```

Let us select the reviews and summaries whose length falls below or equal to **max_text_len** and **max_summary_len**

In [0]:
```
cleaned_text =np.array(data['cleaned_text'])
cleaned_summary=np.array(data['cleaned_summary'])

short_text=[]
short_summary=[]

for i in range(len(cleaned_text)):
    if(len(cleaned_summary[i].split())<=max_summary_len and len(cleaned_text[i].split())<=max_text_len):
        short_text.append(cleaned_text[i])
        short_summary.append(cleaned_summary[i])

df=pd.DataFrame({'text':short_text,'summary':short_summary})
```

We now add the START and END special tokens at the beginning and end of the summary. Here, I have chosen sostok and eostok as START and END tokens

Note: Be sure that the chosen special tokens never appear in the summary

In [0]:
```
df['summary'] = df['summary'].apply(lambda x : 'sostok '+ x + ' eostok')
```

We are getting closer to the model building part. Before that, we need to split our dataset into a training and validation set. We'll use 90% of the dataset as the training data and evaluate the performance on the remaining 10% (holdout set):

In [0]:
```
from sklearn.model_selection import train_test_split
x_tr,x_val,y_tr,y_val=train_test_split(np.array(df['text']),np.array(df['summary']),test_size=0.1,random_state=0,shuffle=True)
```

## Preparing the Tokenizer

A tokenizer builds the vocabulary and converts a word sequence to an integer sequence.

## Text Tokenizer

In [0]:

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

#prepare a tokenizer for reviews on training data
x_tokenizer = Tokenizer()
x_tokenizer.fit_on_texts(list(x_tr))
```

## Rarewords and its Coverage

Let us look at the proportion rare words and its total coverage in the entire text

Here, we define the threshold to be 4 which means word whose count is below 4 is considered as a rare word

In [0]:

```
thresh=4

cnt=0
tot_cnt=0
freq=0
tot_freq=0

for key,value in x_tokenizer.word_counts.items():
    tot_cnt=tot_cnt+1
    tot_freq=tot_freq+value
    if(value<thresh):
        cnt=cnt+1
        freq=freq+value

print("% of rare words in vocabulary:",(cnt/tot_cnt)*100)
print("Total Coverage of rare words:",(freq/tot_freq)*100)
```

% of rare words in vocabulary: 66.12339930151339
Total Coverage of rare words: 2.953684513790566

Remember:

tot_cnt gives the size of vocabulary (which means every unique words in the text)

cnt gives me the no. of rare words whose count falls below threshold

tot_cnt - cnt gives me the top most common words

Let us define the tokenizer with top most common words for reviews.

In [0]:
```
#prepare a tokenizer for reviews on training data
x_tokenizer = Tokenizer(num_words=tot_cnt-cnt)
x_tokenizer.fit_on_texts(list(x_tr))

#convert text sequences into integer sequences
x_tr_seq    =   x_tokenizer.texts_to_sequences(x_tr)
x_val_seq   =   x_tokenizer.texts_to_sequences(x_val)

#padding zero upto maximum length
x_tr    =   pad_sequences(x_tr_seq,  maxlen=max_text_len, padding='post')
x_val   =   pad_sequences(x_val_seq, maxlen=max_text_len, padding='post')

#size of vocabulary ( +1 for padding token)
x_voc   =  x_tokenizer.num_words + 1
```

In [0]:
```
x_voc
```

Out[0]:
```
8440
```

## Summary Tokenizer

In [0]:
```
#prepare a tokenizer for reviews on training data
y_tokenizer = Tokenizer()
y_tokenizer.fit_on_texts(list(y_tr))
```

## Rarewords and its Coverage

Let us look at the proportion rare words and its total coverage in the entire summary

Here, I am defining the threshold to be 6 which means word whose count is below 6 is considered as a rare word

In [0]:
```
thresh=6

cnt=0
tot_cnt=0
freq=0
tot_freq=0

for key,value in y_tokenizer.word_counts.items():
    tot_cnt=tot_cnt+1
    tot_freq=tot_freq+value
    if(value<thresh):
        cnt=cnt+1
        freq=freq+value

print("% of rare words in vocabulary:",(cnt/tot_cnt)*100)
print("Total Coverage of rare words:",(freq/tot_freq)*100)
```

% of rare words in vocabulary: 78.12740675541863
Total Coverage of rare words: 5.3921899389571895

Let us define the tokenizer with top most common words for summary.

In [0]:
```
#prepare a tokenizer for reviews on training data
y_tokenizer = Tokenizer(num_words=tot_cnt-cnt)
y_tokenizer.fit_on_texts(list(y_tr))

#convert text sequences into integer sequences
y_tr_seq    =   y_tokenizer.texts_to_sequences(y_tr)
y_val_seq   =   y_tokenizer.texts_to_sequences(y_val)

#padding zero upto maximum length
y_tr    =   pad_sequences(y_tr_seq, maxlen=max_summary_len, padding='post')
y_val   =   pad_sequences(y_val_seq, maxlen=max_summary_len, padding='post')

#size of vocabulary
y_voc  =   y_tokenizer.num_words +1
```

Let us check whether word count of start token is equal to length of the training data

In [0]:
```
y_tokenizer.word_counts['sostok'],len(y_tr)
```

Out[0]: (42453, 42453)

We are now deleting the rows that contain only START and END tokens

In [0]:
```
ind=[]
for i in range(len(y_tr)):
    cnt=0
    for j in y_tr[i]:
        if j!=0:
            cnt=cnt+1
    if(cnt==2):
        ind.append(i)

y_tr=np.delete(y_tr,ind, axis=0)
x_tr=np.delete(x_tr,ind, axis=0)
```

In [0]:
```
ind=[]
for i in range(len(y_val)):
    cnt=0
    for j in y_val[i]:
        if j!=0:
            cnt=cnt+1
    if(cnt==2):
        ind.append(i)

y_val=np.delete(y_val,ind, axis=0)
x_val=np.delete(x_val,ind, axis=0)
```

**Model building**

Now we build the model.

Return Sequences = True: When the return sequences parameter is set to True, LSTM produces the hidden state and cell state for every timestep

Return State = True: When return state = True, LSTM produces the hidden state and cell state of the last timestep only

Initial State: This is used to initialize the internal states of the LSTM for the first timestep

Stacked LSTM: Stacked LSTM has multiple layers of LSTM stacked on top of each other. This leads to a better representation of the sequence.

Here, we are building a 3 stacked LSTM for the encoder:

In [0]:

```
from keras import backend as K
K.clear_session()

latent_dim = 300
embedding_dim=100

# Encoder
encoder_inputs = Input(shape=(max_text_len,))

#embedding layer
enc_emb =  Embedding(x_voc, embedding_dim,trainable=True)(encoder_inputs)

#encoder lstm 1
encoder_lstm1 = LSTM(latent_dim,return_sequences=True,return_state=True,dropout=0.4,recurrent_dropout=0.4)
encoder_output1, state_h1, state_c1 = encoder_lstm1(enc_emb)

#encoder lstm 2
encoder_lstm2 = LSTM(latent_dim,return_sequences=True,return_state=True,dropout=0.4,recurrent_dropout=0.4)
encoder_output2, state_h2, state_c2 = encoder_lstm2(encoder_output1)

#encoder lstm 3
encoder_lstm3=LSTM(latent_dim, return_state=True, return_sequences=True,dropout=0.4,recurrent_dropout=0.4)
encoder_outputs, state_h, state_c= encoder_lstm3(encoder_output2)

# Set up the decoder, using `encoder_states` as initial state.
decoder_inputs = Input(shape=(None,))

#embedding layer
dec_emb_layer = Embedding(y_voc, embedding_dim,trainable=True)
dec_emb = dec_emb_layer(decoder_inputs)
```

```
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True,dropout=0.4,recurrent_dropout=0.2)
decoder_outputs,decoder_fwd_state, decoder_back_state = decoder_lstm(dec_emb,initial_state=[state_h, state_c])

# Attention layer
attn_layer = AttentionLayer(name='attention_layer')
attn_out, attn_states = attn_layer([encoder_outputs, decoder_outputs])

# Concat attention input and decoder LSTM output
decoder_concat_input = Concatenate(axis=-1, name='concat_layer')([decoder_outputs, attn_out])

#dense layer
decoder_dense =  TimeDistributed(Dense(y_voc, activation='softmax'))
decoder_outputs = decoder_dense(decoder_concat_input)

# Define the model
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)

model.summary()
```

Using sparse categorical cross-entropy as the loss function since it converts the integer sequence to a one-hot vector on the fly. This overcomes any memory issues.

```
In [0]: model.compile(optimizer='rmsprop', loss='sparse_categorical_crossentropy')
```

Now we use the concept of early stopping, It is used to stop training the neural network at the right time by monitoring a user-specified metric. Here, we are monitoring the validation loss (val_loss). Our model will stop training once the validation loss increases:

```
In [0]: es = EarlyStopping(monitor='val_loss', mode='min', verbose=1,patience=2)
```

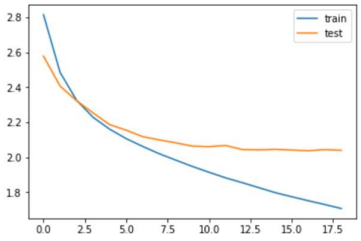
We'll train the model on a batch size of 128 and validate it on the holdout set (which is 10% of our dataset):

```
In [0]: history=model.fit([x_tr,y_tr[:,:-1]], y_tr.reshape(y_tr.shape[0],y_tr.shape[1], 1)[:,1:] ,epochs=50,callbacks=[es],batch_size=128, validation_data=([
```

## Understanding the Diagnostic plot

Now, we will plot a few diagnostic plots to understand the behavior of the model over time:

```
In [0]: from matplotlib import pyplot
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```




From the plot, we can infer that validation loss has increased after epoch 17 for 2 successive epochs. Hence, training is stopped at epoch 19.

Building the dictionary to convert the index to word for target and source vocabulary:

In [0]:
```
reverse_target_word_index=y_tokenizer.index_word
reverse_source_word_index=x_tokenizer.index_word
target_word_index=y_tokenizer.word_index
```

## Inference

Set up the inference for the encoder and decoder:

In [0]:
```
# Encode the input sequence to get the feature vector
encoder_model = Model(inputs=encoder_inputs,outputs=[encoder_outputs, state_h, state_c])

# Decoder setup
# Below tensors will hold the states of the previous time step
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_hidden_state_input = Input(shape=(max_text_len,latent_dim))

# Get the embeddings of the decoder sequence
dec_emb2= dec_emb_layer(decoder_inputs)
# To predict the next word in the sequence, set the initial states to the states from the previous time step
decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2, initial_state=[decoder_state_input_h, decoder_state_input_c])

#attention inference
attn_out_inf, attn_states_inf = attn_layer([decoder_hidden_state_input, decoder_outputs2])
decoder_inf_concat = Concatenate(axis=-1, name='concat')([decoder_outputs2, attn_out_inf])

# A dense softmax layer to generate prob dist. over the target vocabulary
decoder_outputs2 = decoder_dense(decoder_inf_concat)

# Final decoder model
decoder_model = Model(
    [decoder_inputs] + [decoder_hidden_state_input,decoder_state_input_h, decoder_state_input_c],
    [decoder_outputs2] + [state_h2, state_c2])
```

Function below is the implementation of the inference process

In [0]:
```
def decode_sequence(input_seq):
    # Encode the input as state vectors.
    e_out, e_h, e_c = encoder_model.predict(input_seq)

    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1,1))

    # Populate the first word of target sequence with the start word.
    target_seq[0, 0] = target_word_index['sostok']

    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:

        output_tokens, h, c = decoder_model.predict([target_seq] + [e_out, e_h, e_c])

        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_token = reverse_target_word_index[sampled_token_index]

        if(sampled_token!='eostok'):
            decoded_sentence += ' '+sampled_token

        # Exit condition: either hit max length or find stop word.
        if (sampled_token == 'eostok'  or len(decoded_sentence.split()) >= (max_summary_len-1)):
            stop_condition = True

        # Update the target sequence (of length 1).
        target_seq = np.zeros((1,1))
        target_seq[0, 0] = sampled_token_index

        # Update internal states
        e_h, e_c = h, c
```

**return** decoded_sentence

Let us define the functions to convert an integer sequence to a word sequence for summary as well as the reviews:

In [0]:
```
def seq2summary(input_seq):
    newString=''
    for i in input_seq:
        if((i!=0 and i!=target_word_index['sostok']) and i!=target_word_index['eostok']):
            newString=newString+reverse_target_word_index[i]+' '
    return newString

def seq2text(input_seq):
    newString=''
    for i in input_seq:
        if(i!=0):
            newString=newString+reverse_source_word_index[i]+' '
    return newString
```

# CHAPTER 6: Result/Output

Here are a few summaries generated by the model:

In [0]:
```
for i in range(0,100):
    print("Review:",seq2text(x_tr[i]))
    print("Original summary:",seq2summary(y_tr[i]))
    print("Predicted summary:",decode_sequence(x_tr[i].reshape(1,max_text_len)))
    print("\n")
```

Review: gave caffeine shakes heart anxiety attack plus tastes unbelievably bad s tick coffee tea soda thanks
Original summary: hour
Predicted summary:  not worth the money

Review: got great course good belgian chocolates better
Original summary: would like to give it stars
Predicted summary:  good

Review: one best flavored coffees tried usually like flavored coffees one great s erve company love

Original summary: delicious
Predicted summary:  great coffee

Review: salt separate area pain makes hard regulate salt putting like salt go ahea d get product
Original summary: tastes ok packaging
Predicted summary:  salt

Review: really like product super easy order online delivered much cheaper buy ing gas station stocking good long drives
Original summary: turkey jerky is great
Predicted summary:  great product

Review: best salad dressing delivered promptly quantities last vidalia onion dres sing compares made oak hill farms sometimes find costco order front door want even orders cut shipping costs
Original summary: my favorite salad dressing
Predicted summary:  great product

Review: think sitting around warehouse long time took long time send got tea ta sted like cardboard red rasberry leaf tea know supposed taste like
Original summary: stale
Predicted summary:  not as good as

Review: year old cat special diet digestive problems also diabetes stopped eatin g usual special formula food tried different kinds catfood one liked easy digestio n diabetes thank newman
Original summary: wonderful
Predicted summary:  cat food

Review: always perfect snack dog loves knows exactly starts ask time evening g ets greenie snack thank excellent product fast delivery

Original summary: greenies buddy treat
Predicted summary:  great treat

Review: dog loves tiny treats keep one car one house
Original summary: dog loves them
Predicted summary:  my dog loves these

Review: liked coffee much subscribing dark rich smooth
Original summary: makes great cup of java
Predicted summary:  good coffee

Review: far dog tried chicken peanut butter flavor absolutely loves love natural makes happy giving dog something healthy treats small soft big plus calories
Original summary: love zuke mini naturals
Predicted summary:  my dog loves these

Review: absolutely delicious satisfy something sweet really filling great early m orning time make breakfast great afternoon snack work feeling sluggish
Original summary: protein bar
Predicted summary:  yummy

Review: aware decaf coffee although showed search decaf cups intended purcha se gift kept recipient drink caffeine favorite means
Original summary: not decaf
Predicted summary:  decaf decaf

Review: wonderful wrote perfect iced cookie one pen writing cookies names ha ppy ca
Original summary: cookie
Predicted summary:  delicious

Review: truffle oil quite good prefer brand france urbani italy expensive oh deli cious tried black white good black bit stronger pungent event healthy alternative butter enjoy

Original summary: delicious but not the best

Predicted summary:  great flavor

Review: enjoy coffee office split right middle loving think worth try order regul arly

Original summary: hit or miss

Predicted summary:  good coffee

Review: husband gluten free food several years tried several different bread mix es first actually enjoys buying amazon saves loaf

Original summary: really good gluten free bread

Predicted summary:  great gluten free bread

Review: hubby eats says good snacks morning done apple flavor

Original summary: really good nice snack

Predicted summary:  great snack

Review: waste money disgusting product chocolate taste tastes like plastic linin g paper carton using milk treated ultra high temperatures like fresh milk go get f resh milk hershey syrup want chocolate milk

Original summary: please do not waste your money

Predicted summary:  yuck

Review: absolutely loves apple chicken happy hips looks forward one morning one night gets soooo excited would eat allowed

Original summary: healthy treats

Predicted summary:  great for training

Review: strong much flavor little aroma tried purchase another time similiar bra nds met standards expected
Original summary: no flavor
Predicted summary:  san francisco bay coffee

Review: company wanted chose order anyway
Original summary: water
Predicted summary:  not good

Review: introduced number people hooked best sour gummy ever great flavors got great price
Original summary: new favorite
Predicted summary:  best licorice

Review: new price attractive however tastes horrible maybe old zico coconut w ater brands might find acceptable
Original summary: do not be by the price
Predicted summary:  terrible

Review: sure ever going buy product way expensive market price
Original summary: too expensive
Predicted summary:  good value

Review: flavor normally find local stores plus buy bulk things take savings add veggies even stir egg noodles cook add nutrition quick meals lot extra
Original summary: good value
Predicted summary:  great deal

Review: item arrived sugar free shipped regular version caramel syrup small int ernal sticker bottle stated sugar free although company label bottle stated regula r version
Original summary: wrong item

Predicted summary:  not as good as

Review: like strong coffee coffee rated found weak sickening taste
Original summary: disapointed
Predicted summary:  not bad

Review: saw peanut butter chocolate cereal knew try pleased eat chocolate brea kfast feel guilty two kids love cereal well great eat alone favorite milk product y ogurt mix homemade granola well
Original summary: the yummy
Predicted summary:  yummy

Review: tulsi green tea great good iced tea well
Original summary: green tea
Predicted summary:  green tea

Review: always put something market couple poof gone best tasting product pe psi
Original summary: best taste
Predicted summary:  great taste

Review: like tomatoes fresh flavorful also come carton welcome alternative met al cans impart flavor sometimes lined plastic containing
Original summary: yummy tomatoes good packaging
Predicted summary:  good product

Review: great get habit forming careful bought whole case save overall versus g oing supermarket rich dark chocolate crisp cookie worth every penny oreo eat h eart
Original summary: delicious
Predicted summary:  love these

Review: else say arrived promptly perhaps time expected expiration date like ne xt day good go
Original summary: baby loves it
Predicted summary:  not what expected

Review: bought local recently advertised cheesy flavor detectable product even salt flavor avoid product
Original summary: no cheese flavor
Predicted summary:  good but not great

Review: big volume coffee morning one great
Original summary: great morning coffee
Predicted summary:  great coffee

Review: drank try keep awake fell asleep minutes drinking feel anything
Original summary: it made me fall
Predicted summary:  great taste

Review: drink cups day verona italian french roast coffee wanted try lower acid version brand coffee smells tastes like vinegar totally unpalatable better drinkin g water acid coffee bothers
Original summary: single worst coffee ever
Predicted summary:  not very good

Review: getting price however afraid stocking anymore reduced price think one trying eat crackers low calorie string cheese breakfast every total calories put br eakfast baggie go
Original summary: am addicted to these
Predicted summary:  not as good as expected

Review: first time using fondarific fondant general one really easy use baby sho wer cake worked indicated also colored made two tier cake final product looked great greasy
Original summary: easy to use
Predicted summary:  great product

Review: work home drink cups cup coffee day good tasting coffee lowest price cup market
Original summary: great coffee great price
Predicted summary:  great coffee

Review: guys say natural really tastes great pleasantly surprised stand flavor car bonated think would even better product time come fed sweet juices aftertaste m ake obvious really natural switch really gets vote
Original summary: great taste all natural
Predicted summary:  not very good

Review: product good goes long way quite good one dd good product less
Original summary: very good
Predicted summary:  good stuff

Review: tea wonderful soothing even soothing get shipped house found hard fin d decaffeinated tea grocery store much easier
Original summary: decaffeinated french vanilla tea yummy
Predicted summary:  great tea

Review: wow little calorie espresso sugar serve cold delicious little shot espress o sugar overly sweet sugar helps offset taste espresso caffe bitter sweet tastes go od really gave afternoon kick pants
Original summary: nice little pick me up
Predicted summary:  best water ever

Review: mayonnaise delicious side side taste test would give hellman edge hell man richer taste
Original summary: excellent but
Predicted summary:  good stuff

Review: love medium full flavored roast smooth taste bitter acidic taste excellen t coffee good value also try timothy kona good also
Original summary: wonderful coffee
Predicted summary:  great coffee

Review: nice item chunks meat good gravy cat fond varieties nice little treat no netheless think item bit pricy per ounce
Original summary: nice but pricey
Predicted summary:  good cat food

Review: bought cookies gifts open last long good make great gifts would definit ely buy
Original summary: mouth watery cookies
Predicted summary:  cookies

Review: great price fast shipping best chips better ingredients less calories snac k foods plus taste like real chips
Original summary: pop chips are the best
Predicted summary:  great chips

Review: taco bell chipotle sauce bold flavorful tried chicken wings tacos salad made dish extremely tasty glad sampled new sauce staple condiment
Original summary: bold flavor
Predicted summary:  great taste

Review: bought seeds make centerpieces really surprised fast grow planted seed s potting soil without ny preparation anything kept watering days super tall read y displayed centerpieces perfect
Original summary: perfect for in days
Predicted summary:  great seeds

Review: every time need sun dried tomatoes local grocery stores conveniently s mall pouches ensure always hand called recipe
Original summary: sun dried tomato bliss
Predicted summary:  great product

Review: love soup eat plain use recipe cannot find area glad amazon
Original summary: soup chicken cheese
Predicted summary:  soup

Review: throw pack one actually taste bad especially compared orange tangerin e like carbonation adds juice flavors need work switch drinks best worst waterm elon strawberry kiwi berry black cherry orange tangerine
Original summary: my favorite of the four tried
Predicted summary:  tastes like

Review: daughter drinking since months old months old still loves snack time h ealthy delicious great addition menu
Original summary: great snack
Predicted summary:  great snack

Review: live guinea africa order products delivered boat every months sometim es disappointed time zero calories zero carbs taste great price zero delivery costs prime ordered different flavors one favorite love
Original summary: love it
Predicted summary:  great product

Review: purchased larger size love size perfect keep purse snack especially time s others dessert snack cannot eat must gluten free spouse touch diet food loves
Original summary: cannot get enough
Predicted summary:  great snack

Review: always house drink favorite mix sprite oh good every day mind larger bottles use much bring
Original summary: am an adult still love this
Predicted summary:  great taste

Review: ginger snaps overpowering ginger go great milk really enjoyed house g reat buy affordable compared alternative diet foods last least week store well
Original summary: you can eat ginger again
Predicted summary:  ginger

Review: give squid one star use might thoroughly disappointed quite possibly c all crazy
Original summary: can for your
Predicted summary:  good stuff

Review: quality seeds excellent begin germinate hours days ready use never spr outed seeds results good easily recommend sprouter whether human consumptio n four legged friends
Original summary: wheat grass seeds
Predicted summary:  great product

Even though the actual summary and the summary generated by our model do not match in terms of words, both of them are conveying the same meaning. Our model is able to generate a legible summary based on the context present in the text.

This is how we can perform text summarization using deep learning concepts in Python.

# CHAPTER 7. FEASIBILITY STUDY

Feasibility analysis can occur when objectives are defined. It starts with creating comprehensive, viable solutions that provide an indication of what the new system should look like. This is where art and imagination are used. Analysts have to come up with new ways of doing things - they have come up with new ideas.

There is no need to log in to a detailed system operation yet. The solution should provide enough information to make reasonable estimates of project costs and provide users with an indication of how the new system will fit into the organization. It is important not to make an effort at this point only to find out if the project is not worthwhile or if there is a need to radically change the original goal.

Feasibility of system means ensuring that the system, which we are going to implement, is efficient and less costly. There are various types of feasibility to be determined. They are :

## 7.1 Economical Feasibility

The development of this application is very economically possible. The only thing that needs to be done is to make room for effective monitoring.

It effectively costs in the sense that you have completely removed the paperwork. The application uses resources already available for free and is highly optimized. If the model is to be used on a huge scale then higher RAM and GPU memory will be required.

### 7.2 Technical Feasibility

The technical requirement of the neural network model is economic and does not use any additional Hardware and additional software. The technical assessment should also assess whether existing systems can be upgraded to use new technologies and whether the organization has the technology to implement them. The model is based on neural networks which requires CUDA cores for faster processing and computations. Opensource libraries are used – Tensorflow, Keras, pandas etc. which are highly optimized.

### 7.3 Operational Feasibility

The operation is easy to use and read because of the user does not need special training to use the system.

Feasibility analysis can occur when objectives are defined. It starts with creating comprehensive, viable solutions that provide an indication of what the new system should look like. This is where art and imagination are used. Analysts have to come up with new ways of doing things - they have come up with new ideas.

If the user needs to make some changes to the existing model or wants to makes some modifications to it, the operating system requires the understanding of the various machine learning libraries used and the working of the neural networks, how the computations occur.

# 8. CONCLUSION

This effective method for “Text Summarization”, has a wide range of applications in a world where the population is booming every year and it will not be humanly possible at one point evaluate document of each and every person.

This machine learning model makes use of various opensource libraries and software which are available for free and are used widely throughout the globe, so in the future if any modifications or changes are required then one can change the working code for the model without worrying about the future support for the software and libraries.

The field of Machine Learning and Artificial Intelligence has increased at an unbelievable pace in the past few years, this suggests that in the future we can GPUs and software would be able to process the model and training data in a very short time, as well as finish the process of testing the training data with higher accuracies.

In the future our model can be used to recognize and summarize the text of various other languages existing in the word and the model can be modified to even search for a specific symbol or character if given the necessary training data.

# REFERENCES


1. Acharya, Divya, et al. "A long short term memory deep learning network for the classification of negative emotions using EEG signals." 2020 International Joint Conference on Neural Networks (IJCNN). IEEE, 2020.
2. C-L. Liu and K. Marukawa, “Normalization Ensemble for".
3. Handwritten Character Recognition”, The Ninth International Workshop on Frontiers in Handwriting Recognition (IWFHR 9), Tokyo, Japan, pp. 69-74, 2004.
4. http://en.wikipedia.org/wiki/ Handwriting_recognition.
5. Rosenfeld,A. and Kak, A.C., Digital Image Processing, Academic Press Inc., 2nd. Ed., 1993 Simon.
6. Tiwari, Smita, Shivani Goel, and Arpit Bhardwaj. "Machine Learning approach for the classification of EEG signals of multiple imagery tasks." 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT). IEEE, 2020.