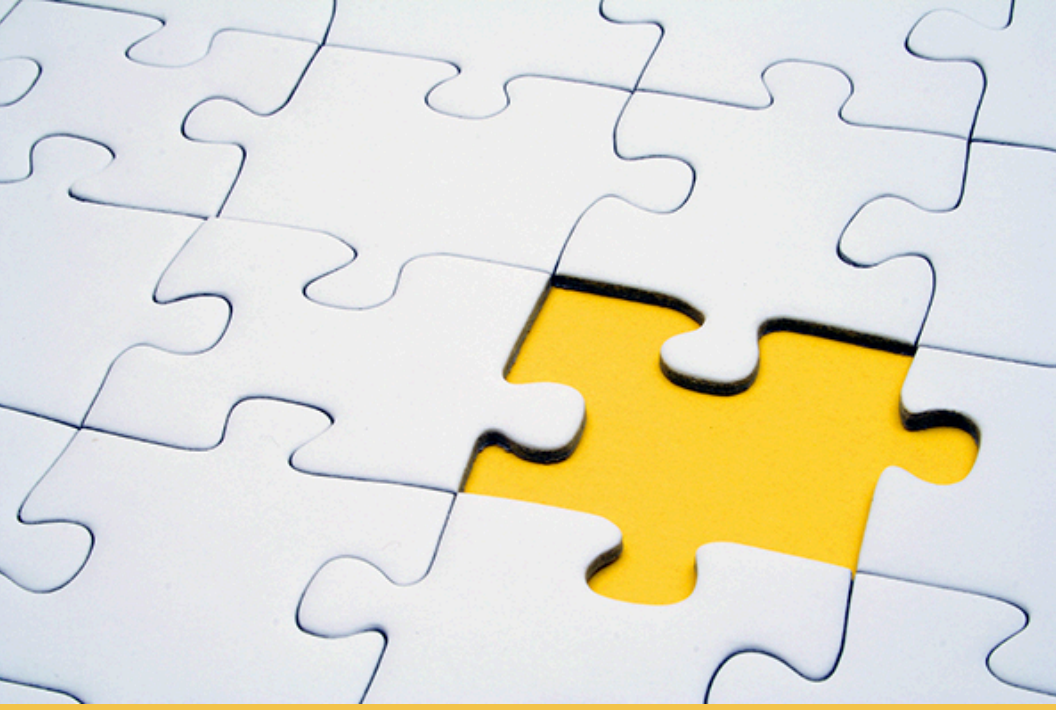


ML PACKAGE

Abstractive Text Summarization using LSTM MODEL

- *Harisaipravin Sv*

Abstractive Text Summarization using LSTM Model

A package which intakes a feedback and returns a summarised easy to understand summary by using neural networks.

ABSTRACT

Text Summarization is the process of extracting the most important information from the source and providing an abridged version to the particular user or an organisation. It can broadly be classified into extractive summarisation and abstractive summarisation. As the growth of information is exponential, at present it is quite challenging to get sense out of it so that it could be leveraged by the stakeholders. One kind of solution could be given using text summarization. The system is evaluated using weighted cross- entropy loss for a sequence of logits. The sequence-to-sequence model is built using bi-directional Recurrent Neural Networks with Long short-term memory. The system shall summarize the text reviews using Natural Language Processing techniques on top of the neural network model so that the information is made effective for the organisation.

Dataset: Amazon Fine Food Reviews dataset collected from the Stanford Network Analysis Project.

Keywords: Natural Language Processing, Summarization, Neural Network, LSTM

TEXT SUMMARIZATION

In the modern Internet age, textual data is ever increasing. Need some way to condense this data while preserving the information and meaning.We need to summarize textual data for that. Text summarization is the process of automatically generating natural language summaries from an input document while retaining the important points. It would help in easy and fast retrieval of information.

There are two prominent types of summarization algorithms

- ExtractiveSummarization
- AbstractiveSummarization

Extractive Summarization

Extractive summarization systems form summaries by copying parts of the source text through some measure of importance and then combine those part/sentences together to render a summary. Importance of sentence is based on linguistic and statistical features.

Abstractive Summarization

Abstractive summarization systems generate new phrases, possibly rephrasing or using words that were not in the original text. Naturally abstractive approaches are harder. For perfect abstract summary, the model has to first truly understand the document and then try to express that understanding in short possibly using new words and phrases. Much harder than extractive. Has complex capabilities like generalization, paraphrasing and incorporating real-world knowledge.

Concepts Used

Natural language processing

Natural Language Processing (NLP) is a field in Computer Science that focuses on the study of the interaction between human languages and computers . Text summarization is in this field because computers are required to understand what humans have written and produce human-readable outputs. NLP can also be seen as a study of Artificial

Intelligence (AI). Therefore many existing AI algorithms and methods, including neural network models, are also used for solving NLP related problems. With the existing research, researchers generally rely on two types of approaches for text summarization: extractive summarization and abstractive summarization

Text Abstraction

Compared to extractive summarization, abstractive summarization is closer to what humans usually expect from text summarization. The process is to understand the original document and rephrase the document to a shorter text while capturing the key points. Text abstraction is primarily done using the concept of artificial neural networks. This section introduces the key concepts needed to understand the models developed for text abstraction.

Artificial Neural Networks

Artificial neural networks are computing systems inspired by biological neural networks. Such systems learn tasks by considering examples and usually without any prior knowledge. For example, in an email spam detector, each email in the dataset is manually labeled as “spam” or “not spam”. By processing this dataset, the artificial neural networks evolve their own set of relevant characteristics between the emails and whether a new email is spam. To expand more, artificial neural networks are composed of artificial neurons called units usually arranged in a series of

layers. Network model below is the most common architecture of a neural network model. It contains three types of layers: the input layer contains units which receive inputs normally in the format of numbers; the output layer contains units that “respond to the input information about how it is learned any task”; the hidden layer contains units between input layer and output layer, and its job is to transform the inputs to something that output layer can use.

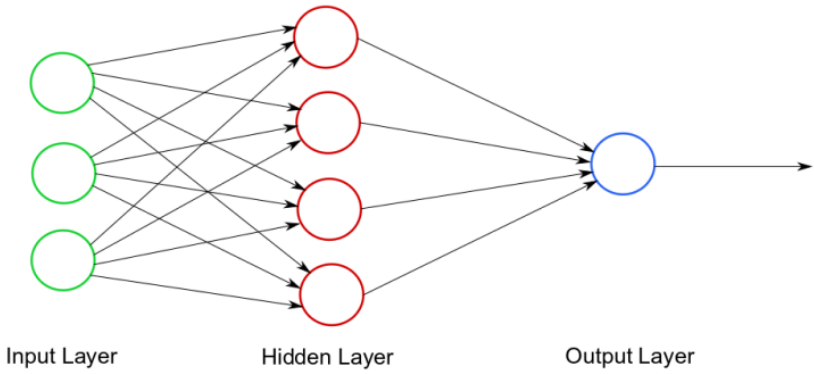


Fig 1:- Neural Network Model

RNN & LSTM

Traditional neural networks do not recall any previous work when building the understanding of the task from the given examples. However, for tasks like text summarization, the sequence of words in input documents is critical. In this case, we want the model to remember the previous words when it processes the next one.

To be able to achieve that, we have to use recurrent neural networks because they are networks with loops in them where information can persist in the model (Christopher, 2015). Figure 2 shows how a Recurrent neural network (RNN) looks like if it is unrolled. For the symbols in the figure, “ht” represents the output units value after each timestamp (if the input is a list of strings, each timestamp can be the processing of one word), “x” represents the input units for each timestamp, and A means a chunk of the neural network. Figure 2 shows that the result from the previous timestamp is passed to the next step for part of the calculation that happens in a chunk of the neural network. Therefore, the information gets captured from the previous timestamp. However, in practice, traditional RNNs often do not memorize information efficiently with the increasing distance between the connected information. Since each activation function is nonlinear, it is hard to trace back to hundreds or thousands of operations to get the information.

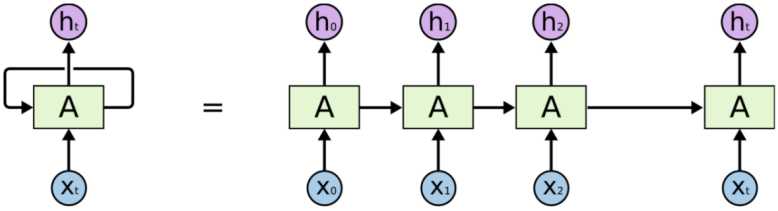


Figure 2 :- Recurrent Neural Network

Fortunately, Long Short-Term Memory (LSTM) networks can convey information in the long term. Different from the traditional RNN, inside each LSTM cell, there are several simple linear operations which allow data to be conveyed without doing the complex computation. As shown in , the previous cell state containing all the information so far smoothly goes through an LSTM cell by doing some linear operations. Inside, each LTSM cell makes decisions about what information to keep, and when to allow reads, writes and erasures of information via three gates that open and close. As shown in Figure 4, the first gate is called the "forget gate layer", which takes the previous output units value ht-1 and the current

Figure 1.3

input xt, and outputs a number between 0 and 1 to indicate the ratio of passing information. 0 means do not let any information pass, while 1 means let all information pass. To decide what information needs to be updated, LSTM contains the "input gate layer". It also takes in the previous output units value ht-1 and the current input xt and outputs a number to indicate inside which cells the information should be updated. Then, the previous cell state Ct-1 is updated to the new state Ct. The last gate is "output gate layer", which decides what the output should be. Figure 6 shows that in the output layer, the cell state is going through a tanh function, and then it is multiplied by the weighted output of the sigmoid function. So, the output units value ht is passed to the next LSTM cell. Simple linear operators connect the three gate layers. The vast LSTM neural network consists of many LSTM cells, and all

information is passed through all the cells while the critical information is kept to the end, no matter how many cells the network has.

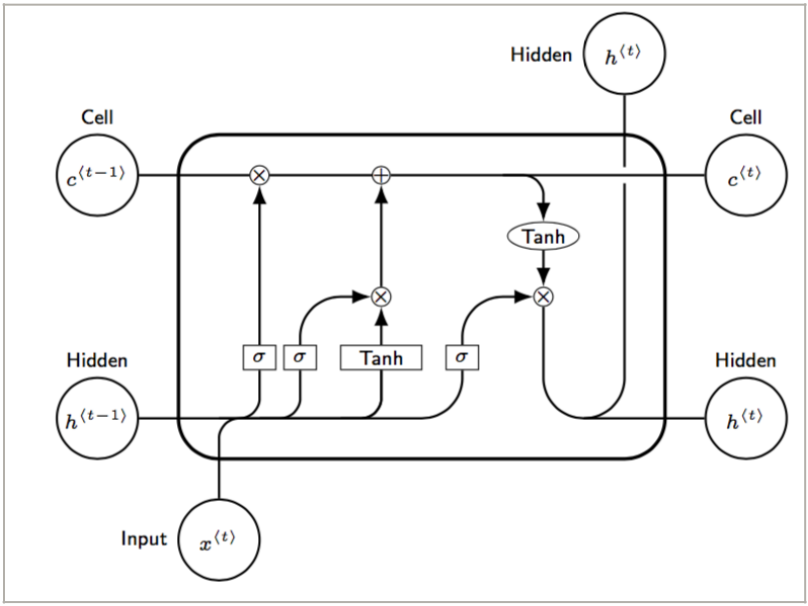


Fig 3:- LSTM

Word Embeddings

Word embedding is a set of feature learning techniques in NLP where words are mapped to vectors of real numbers. It allows similar words that have similar representation, so it builds a relationship between words and allows calculations among them. A typical example is that after representing words to vectors, the function “king - men + women” would ideally give the vector representation for the word “queen”. The benefit of using word embedding is that it captures more meaning of the word.

DATASET

Amazon Fine Food Reviews from Stanford Network Analysis Project. This dataset consists of reviews of fine foods from amazon. The data span a period of more than 10 years, including all ~500,000 reviews up to October 2012. Reviews include product and user information, ratings, and a plain text review. It also includes reviews from all other Amazon categories.
Data includes:

- - Reviews from Oct 1999 - Oct 2012
- - 568,454 reviews
- - 256,059 users
- - 74,258 products
- - 260 users with > 50 reviews

Libraries

- Keras
- NLTK
- Scikit-Learn
- Pandas

METHODOLOGY

The overall process of this project:

Figure 4:- Overall Process

Data Collection

The dataset is collected from Stanford Network Analysis Project hosted in Kaggle site.

Data Preprocessing

This section includes the methods of data preprocessing

Tokenization

Tokenization is a step which splits longer strings of text into smaller pieces, or tokens. Larger chunks of text can be tokenized into sentences, sentences can be tokenized into words, etc. Further processing is generally performed after a piece of text has been appropriately tokenized.
Tokenization is also referred to as text segmentation or lexical analysis. Sometimes segmentation is used to refer to the breakdown of a large chunk of text into pieces larger than words (e.g. paragraphs or sentences), while tokenization is reserved for the breakdown process which results exclusively in words.

Normalization

Before further processing, text needs to be normalized. Normalization generally refers to a series of related tasks meant to put all text on a level playing field: converting all text to the same case (upper or lower), removing punctuation, converting numbers to their word equivalents, and so on. Normalization puts all words on equal footing, and allows processing to proceed uniformly.

Stemming
Stemming is the process of eliminating affixes (suffixed, prefixes, infixes, circumfixes) from a word in order to obtain a word stem.

Lemmatization
Lemmatization is related to stemming, differing in that lemmatization is able to capture canonical forms based on a word's lemma.

Noise Removal
Noise removal continues the substitution tasks of the framework. While the first 2 major steps of our framework (tokenization and normalization) were generally applicable as-is to nearly any text chunk or project (barring the decision of which exact implementation was to be employed, or skipping certain optional steps, such as sparse term removal, which simply does not apply to every project), noise removal is a much more task-specific section of the framework.

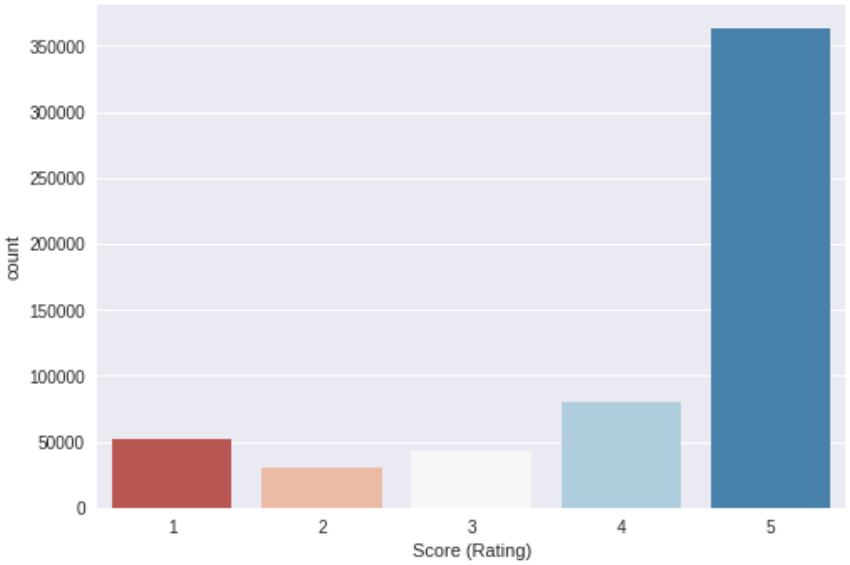
Data Exploration

Data Fields Explanation

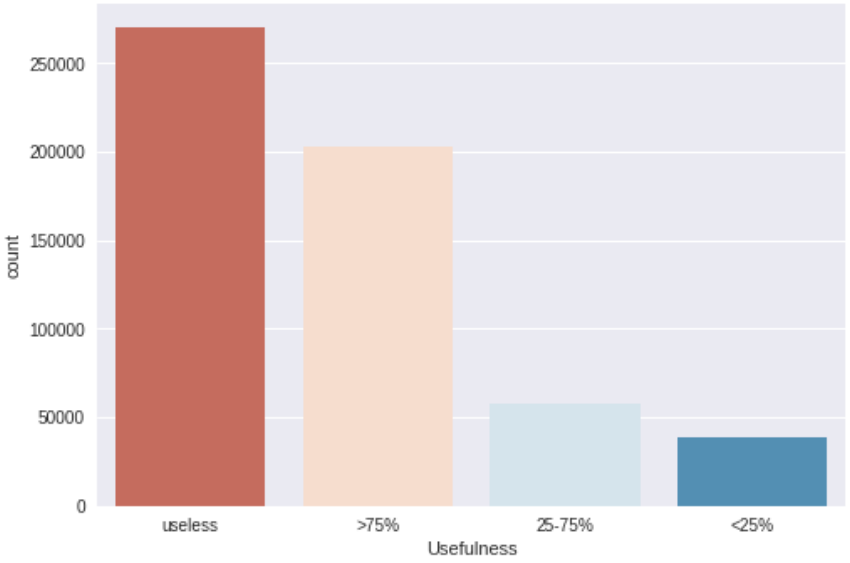
- Id - Unique row number
- ProductId - unique identifier for the product
- UserId - unique identifier for the user name
- HelpfulnessNumerator - number of users who found the review helpful
- HelpfulnessDenominator - number of users who indicated whether they found the review helpful
- Score - rating between 1 and 5
- Time - timestamp for the review
- Summary - brief summary of the review
- Text - text of the review

Rating Analysis

Till now we saw that 5-star reviews constitute a large proportion (64%) of all reviews. The next most prevalent rating is 4-stars(14%), followed by 1-star (9%), 3-star (8%), and finally 2-star reviews (5%).



Exploration of Helpfulness

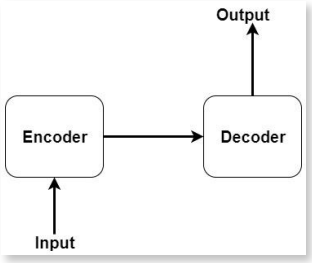


Model Building

This section consists of information related to the model and its architecture.

Sequence-to-Sequence (Seq2Seq) Modeling

Seq2Seq is a method of encoder-decoder based machine translation that maps an input sequence to an output of sequence with a tag and attention value. The idea is to use 2 RNN that will work together with a special token and trying to predict the next state sequence from the previous sequence.



Encoder

The encoder simply takes the input data, and train on it then it passes the last state of its recurrent layer as an initial state to the first recurrent layer of the decoder part.

Decoder

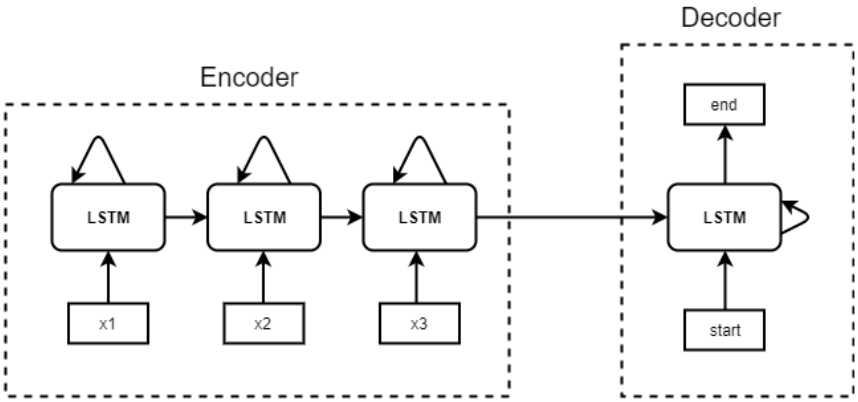
The decoder takes the last state of the encoder's last recurrent layer and uses it as an initial state to its first

recurrent layer , the input of the decoder is the sequences that we want to get

Many-to-Many Seq2Seq

Many-to-Many Seq2Seq architecture can handle applications where the input and output sequence lengths are not the same using the encoder/decoder setup shown in

he bottom right of the diagram above. Many-to-many models are commonly referred to as sequence-to-sequence models in the literature.



Many-to-Many Seq2Seq Architecture

The encoder is made up of :

1. Input Layer : Takes the sentence and pass it to the embedding layer.

2. Embedding Layer : Takes the sentence and convert each word to fixed size
vector

3. First LSTM Layer : Every time step, it takes a vector that represents a word
and pass its output to the next layer.

4. Second LSTM Layer : It does the same thing as the previous layer, but
instead of passing its output, it passes its states to the next layer.

5. Third LSTM Layer: It does the same thing as the previous layer, but instead of passing its output, it passes its states to the first LSTM layer of the decoder.

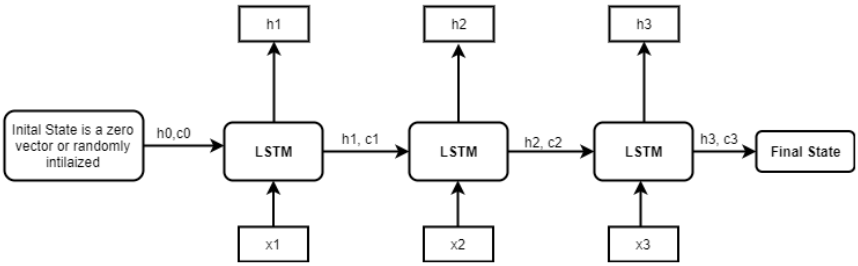


Fig 8:- Encoder

The decoder is made up of :

1. Input Layer : Takes the input sentence and pass it to the embedding layer.

2. Embedding Layer : Takes the input sentence and convert each word to fixed
size vector

3. LSTM Layer : Every time step, it takes a vector that represents a word and
pass its output to the next layer, but here in the decoder, we initialize the state of this layer to be the last state of the last LSTM layer from the decoder . Processing the output from the previous layer. Takes the output from the previous layer and outputs a one hot vector representing the summarized review.

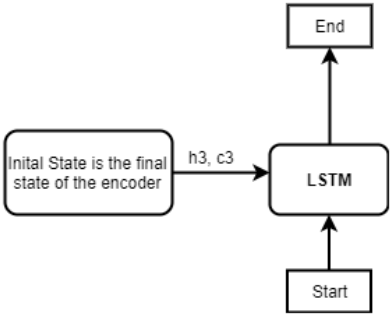


Fig 9:- Decoder

Limitations of Encoder - Decoder Model:

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

1. 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

2. It is difficult for the encoder to memorize long sequences into a fixed length vector.

To overcome this limitation, we use the concept of attention mechanism comes into the picture. It aims to predict a word by looking at a few specific parts of the sequence only, rather than the entire sequence.

Attention Mechanism :

Attention is proposed as a solution to the limitation of the Encoder-Decoder model encoding the input sequence to one fixed length vector from which to decode each output time step. This issue is believed to be more of a problem when decoding long sequences. Attention is proposed as a method to both align and translate. Alignment is the problem in machine translation that identifies which parts of the input sequence are relevant to each word in the output, whereas translation is the process of using the relevant information to select the appropriate output.

Instead of encoding the input sequence into a single fixed context vector, the attention model develops a context vector that is filtered specifically for each output time step.

IMPLEMENTATION

Importing Libraries

```
[ ] from nltk.corpus import stopwords # frequently used words like a, an , the , etc
    from tensorflow.keras.callbacks import EarlyStopping # To stop when Model converges
    from tensorflow.keras.models import Model # To build a Model
    from tensorflow.keras.layers import Input, LSTM, Embedding, Dense, Concatenate, TimeDistributed # LSTM Support using keras
    from bs4 import BeautifulSoup # Remoeving html tags
    from keras.preprocessing.text import Tokenizer  # vectorize a text
    from keras.preprocessing.sequence import pad_sequences # ensure that all sequences in a list have the same size
    import warnings # To omit warnings
    import numpy as np # NumPy is used for working with arrays
    import pandas as pd # Manupulating data with dataframe
    import re #regular expressions to be removed
```

Preprocessing

- - Convert everything to lowercase
- - Remove HTML tags
- - Contraction mapping
- - Remove (’s)
- - Remove any text inside the parenthesis ( )
- - Eliminate punctuations and special characters
- - Remove stopwords
- - Remove short words

Text Cleaning

```
def clean_data(text,num):
    buffer = text.lower()
    buffer = BeautifulSoup(buffer, "lxml").text # Handling XML and HTML related datas by converting to text
    buffer = re.sub(r'\([^)]*\)', '', buffer) #removing unnecessary datas
    buffer = re.sub('"','', buffer) # clearing double quotes
    buffer = ' '.join([spams[t] if t in spams else t for t in buffer.split(" ")]) # removing spam words
    buffer = re.sub(r"'s\b","",buffer) # plurals and empty string
    buffer = re.sub("[^a-zA-Z]", " ", buffer) #special characters removed
    buffer = re.sub('[m]{2,}', 'mm', buffer) # remove repeating words

    if(num==0):
        tokens = [w for w in buffer.split() if not w in word_s] #first cleanup removing stowords
    else:
        tokens=buffer.split() #second cleanup seperating them

    long_words=[]
    for i in tokens:
        if len(i)>1:
            long_words.append(i) #records having more than 1 words still
    return (" ".join(long_words)).strip()
```

Preparing the tokenizer

```
tokens_x = Tokenizer(num_words=tot_cnt-cnt)  # tokenizer initiallization
tokens_x.fit_on_texts(list(train_x)) # Updates internal vocabulary based on a list of texts

value_x_seq = tokens_x.texts_to_sequences(value_x) # Transforms each text in texts to a sequence of integers from word_index dictionary
value_x = pad_sequences(value_x_seq, maxlen=MAX_T_L, padding='post') # checks if all records in the list are of same size or it apends 0 post the given rec

train_x_seq = tokens_x.texts_to_sequences(train_x) # Transforms each text in texts to a sequence of integers from word_index dictionary
train_x = pad_sequences(train_x_seq, maxlen=MAX_T_L, padding='post') # checks if all records in the list are of same size or it apends 0 post the given rec

final_x = tokens_x.num_words + 1 # Total words in the record +1 for NULL
```

Model Building

- 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.

```
# stack 1 LSTM
answer_a=LSTM(facet_a,
              dropout=0.4, # Attention paid to all the data elements during this iteartion
              return_sequences=True, # LSTM produces the hidden state and cell state for every timestep
              recurrent_dropout=0.4, # Attention paid to input data
              return_state=True)
r1,tell1,tell2=answer_a(coll_d)

# stack 2 LSTM
answer_b=LSTM(facet_a,
              dropout=0.4, # Attention paid to all the data elements during this iteartion
              return_sequences=True,  # LSTM produces the hidden state and cell state for every timestep
              recurrent_dropout=0.4, # Attention paid to the before result iteration
              return_state=True)
r2,m2,m3=answer_b(r1)

# stack 3 LSTM
answer_c=LSTM(facet_a,
              dropout=0.4, # Attention paid to all the data elements during this iteartion
              return_sequences=True,  # LSTM produces the hidden state and cell state for every timestep
              recurrent_dropout=0.4, # Attention paid to the before result iteration
              return_state=True)
eye_e,item_a,item_b=answer_c(r2)
# All information is passed through all the cells while the critical information is kept to the end, no matter how many cells the network has.


ins_datum=Input(shape=(None,))
# builds a relationship between words and allows calculations among them
# Eg:- "king - men + women" = "queen"
stack_e=Embedding(final_y, facet_e,trainable=True)
podium_d=stack_e(ins_datum)

#Final LSTM Decoding for biderectional check
dmodule=LSTM(facet_a,
              dropout=0.4, # Attention paid to all the data elements during this iteartion
              return_sequences=True,  # LSTM produces the hidden state and cell state for every timestep
              recurrent_dropout=0.2, # Attention paid to the before result iteration
              return_state=True)
```

Result

```
Layer (type)                 Output Shape              Param #   Connected to
==================================================================================================
input_1 (InputLayer)         [(None, 30)]              0
__________________________________________________________________________________________________
embedding (Embedding)        (None, 30, 100)           844000    input_1[0][0]
__________________________________________________________________________________________________
lstm (LSTM)                  [(None, 30, 300), (N 481200    embedding[0][0]
__________________________________________________________________________________________________
input_2 (InputLayer)         [(None, None)]            0
__________________________________________________________________________________________________
lstm_1 (LSTM)                [(None, 30, 300), (N 721200    lstm[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)      (None, None, 100)         198900    input_2[0][0]
__________________________________________________________________________________________________
lstm_2 (LSTM)                [(None, 30, 300), (N 721200    lstm_1[0][0]
__________________________________________________________________________________________________
lstm_3 (LSTM)                [(None, None, 300),  481200    embedding_1[0][0]
                                                            lstm_2[0][1]
                                                            lstm_2[0][2]
__________________________________________________________________________________________________
Algorithm_attention (Algo_atten ((None, None, 300),  180300    lstm_2[0][0]
                                                            lstm_3[0][0]
__________________________________________________________________________________________________
concat_layer (Concatenate)   (None, None, 600)    0         lstm_3[0][0]
                                                            Algorithm_attention[0][0]
__________________________________________________________________________________________________
time_distributed (TimeDistribut (None, None, 1989)   1195389   concat_layer[0][0]
==================================================================================================
Total params: 4,823,389
Trainable params: 4,823,389
Non-trainable params: 0
__________________________________________________________________________________________________
```

OUTPUT :-

(1)

Feedback: gave caffeine shakes heart anxiety attack plus tastes unbelievably bad stick coffee tea soda thanks

Real Result: hour

Predicted Result: great tea

(2)

Feedback: got great course good belgian chocolates better

Real Result: would like to give it stars but

Predicted Result: great taste

(3)

Feedback: one best flavored coffees tried usually like flavored coffees one great serve company love

Real Result: delicious

Predicted Result: great coffee

(4)

Feedback: salt separate area pain makes hard regulate salt putting like salt go ahead get product

Real Result: tastes ok packaging

Predicted Result: great salt

(5)

Feedback: really like product super easy order online delivered much cheaper buying gas station stocking good long drives

Real Result: turkey jerky is great

Predicted Result: great flavor

(6)

Feedback: best salad dressing delivered promptly quantities last vidalia onion dressing compares made oak hill farms sometimes find costco order front door want even orders cut shipping costs

Real Result: my favorite salad dressing

Predicted Result: great product

(7)

Feedback: think sitting around warehouse long time took long time send got tea tasted like cardboard red rasberry leaf tea know supposed taste like

Real Result: stale

Predicted Result: not so good

(8)

Feedback: year old cat special diet digestive problems also diabetes stopped eating usual special formula food tried different kinds catfood one liked easy digestion diabetes thank newman

Real Result: wonderful

Predicted Result: great for cats

(9)

Feedback: always perfect snack dog loves knows exactly starts ask time evening gets greenie snack thank excellent product fast delivery

Real Result: greenies buddy treat

Predicted Result: great for training

(10)

Feedback: dog loves tiny treats keep one car one house

Real Result: dog loves them

Predicted Result: great treat