SWIFTKEY TEXT PREDICTION

Submitted by

Group E

Satish Gupta (02)

Vishal kushwaha (06)

Neha Yadav (18)

Manisha Singh (25)

Batch Details

MSc. Part-II

Course Details

MSc-DSAI

Institute Details

Ramniranjan Jhunjhunwala College

Name of the Guide: Prof. Cyrus Lentin

PROJECT SYNOPSIS

Name of Group	**Group E**	
Name of Students	Gupta Satish 02 Kushwaha Vishal 06 Yadav Neha 18 Singh Manisha 25	
Program	Batch Year	Academic Year
MSc DSAI	2020-2022	2021

Name of Guide	Cyrus Lentin
Title of the Project	SwiftKey Text Prediction
Reason for this topic	To develop the understanding of predictive text model which are used by SwiftKey
Project Details	
Objectives / Goals	The Capstone project involves using the HC Corpora Dataset. The Capstone project is done in collaboration with SwiftKey. The goal of this project is to deploy a web application with text prediction capabilities.
Please mention key Literature which you plan to Study	-Building a Predictive text Model -Studying the various NLP Libraries. Performing Exploratory Data Analysis/Validation discriminant analysis to extract useful insights. -Model selection and prediction accuracy measurements -Webapp developments and deployment
Expected Outcomes which you expect after Data Analysis	-Successful deployment of the Web Application -Webapp predicts the further words for the upcoming sentence. -Webapp Recommends words based on the input through keyboard

	-It Autocorrects -Webapp offers correctly spelled predictions based on the word you're typing. -Webapp autocorrects the word typed for grammatical errors.

	Guide's Signature

Index

Literature Review

This section describes the methodology adopted for the literature review. This paper represents an exploration of the contributions that have already been made in the academic field.

[1] multi-window convolution(MRNN) algorithm is implemented, also they have created residual-connected minimal gated unit(MGU) which is short version of LSTM in this CNN try to skip few layers while training result in less training time and they have good accuracy by far using multiple layers of neural networks can cause latency for predicting n numbers of words.

[2] This paper used RNN algorithm and also they have used GRU another form of RNN for code completion problem as RNN help to predict next code syntax for users. Authors claim that their method is more accurate compare to existing methods. They have separated next word prediction in two components: within-vocabulary words and identifier prediction. They have used LSTM neural language model to predict within vocabulary words. A pointer network model is proposed to identifier prediction.

[3] Authors worked on Bangla Language. They have proposed a novel method for word prediction and word completion. They have proposed N-gram based language model which predicts set of words. They have achieved satisfactory results.

[4] In this paper they created an auto-next-keyword for Bengali language which was challenging and it was found out that it is hard to get good accuracy by using RNN algorithm as due to its vanishing gradients and heavy recurrent NN take more time to train and test.

[5] They unseeded predicting next character highlighter(PNCH) for Indian language it was more of text correction and less about next word prediction but was quite good to understand. The Method called hit and miss but accuracy is less and the model was not efficient for this kind of problem statement.

[6]This was first approach to tackle this kind of problem, the paper discusses about LM and perplexity algorithm which is base of natural language processing, This help us to make a 3D input data for our model.

Data Dictionary (Includes Data Source)

Data Overview

For solving this problem, we need corpus of text data. This data is provided by **SwiftKey**

This zip file contains 4 folders each containing data from 4 different languages namely English, Russian, Finnish and German.

The HC Corpora dataset is comp raised of the output of crawls of news sites, blogs and twitter

We will be using the data from English folder. It contains the following files.

- **en_US.blogs.txt**: this text file contains all English language data from news websites and channels

  https://d rive .google .com/ file/d/0B95yMv4YhoSOSzd IeDJNSDVONEE/vie w?u sp= sha ring&resou rcekey=0-EZ0 rawoyG1dbTFJT--uqAw

- **en_US.news.txt:** this text file contains all English language data from all blog sites

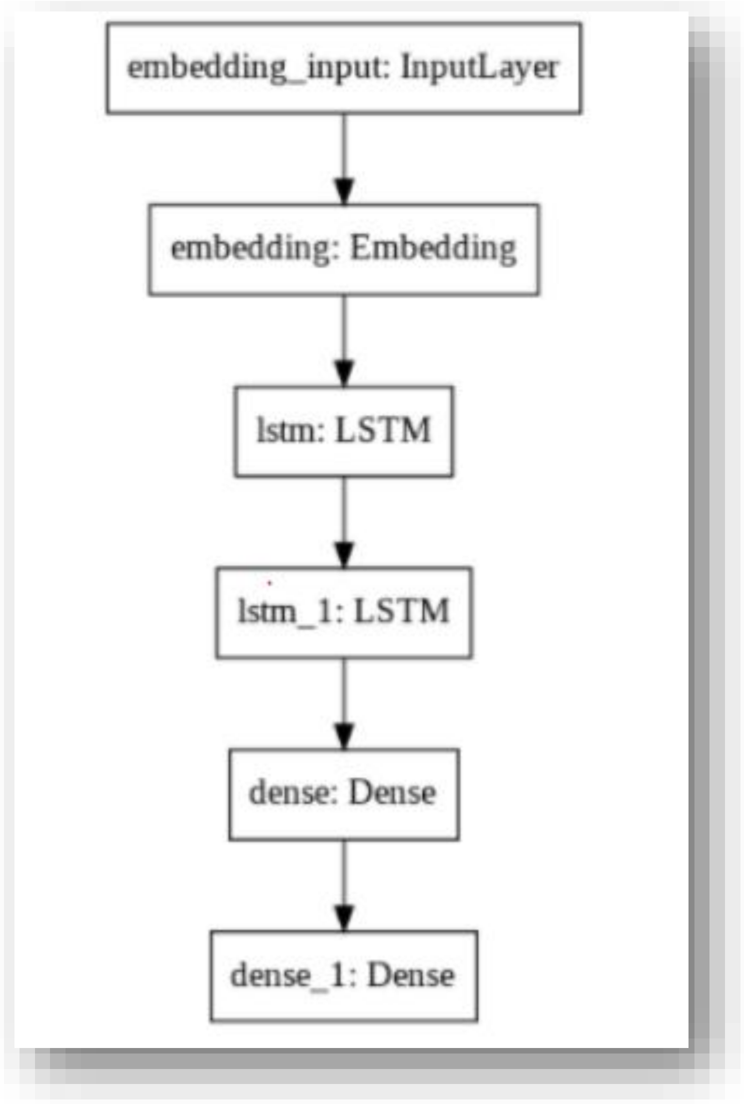
  https://d rive .google .com/ file/d/0B95yMv4YhoSOLVkwbVhaaFBtMFk/vie w?u sp= sha ring&resou rcekey=0-wLDqU jn9P9JuW IxxpoTbWg

- **en_US .twitter.txt**: this text file contains all English language data from twitter.

  https://d rive .google .com/ file/d/0B95yMv4YhoSONXBhX2l4NWVLZFE /vie w?u sp= sha ring&resou rcekey=0-fP4 rFM jTLva zUO593oWnAA

The data sets may be referred to as "Twitter", "Blogs" and "News" for purpose of the project.

Flow Chart


embedding_input: InputLayer
embedding: Embedding
lstm: LSTM
lstm_1: LSTM
dense: Dense
dense_1: Dense

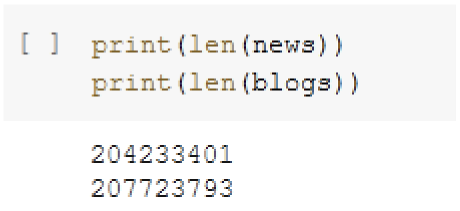
Exploratory Data Analysis

Exploratory Data Analysis (EDA) is an approach/philosophy for data analysis that employs a variety of techniques (mostly graphical) to

1. maximize insight into a data set;
2. uncover underlying structure;
3. extract important variables;
4. detect outliers and anomalies;
5. test underlying assumptions;
6. develop parsimonious models; and
7. Determine optimal factor settings.

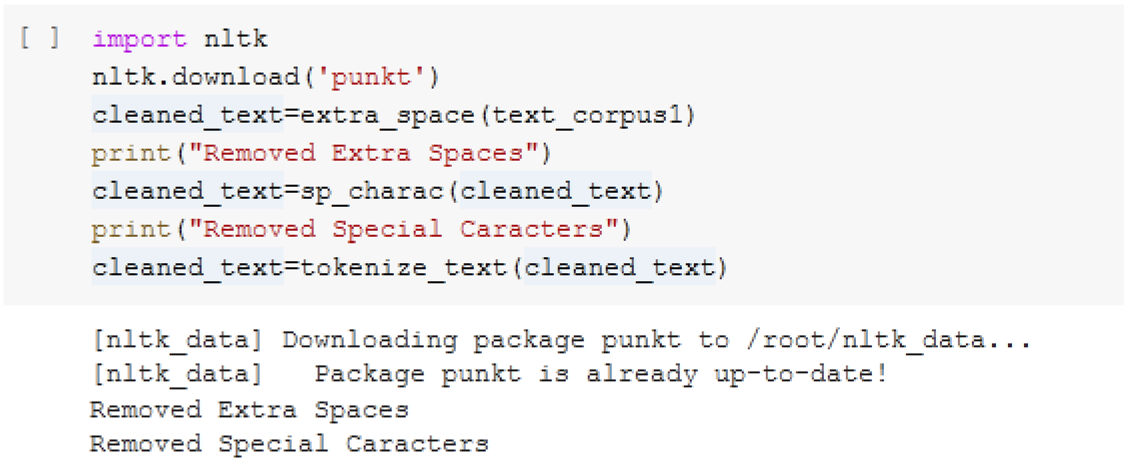
We have used 2 Datasets i.e.US news, Us Blog.

- Length of the data set.



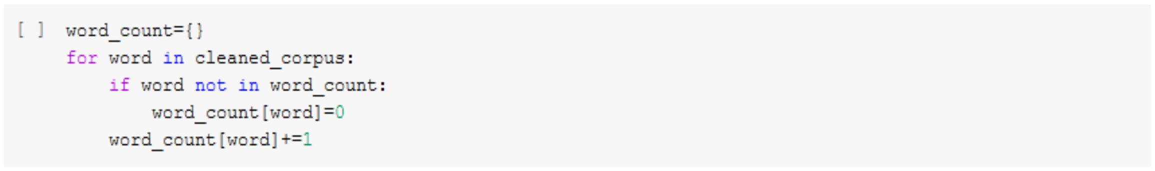
```
[ ]  print(len(news))
     print(len(blogs))

     204233401
     207723793
```

- Removing extra Spaces and Special Character.

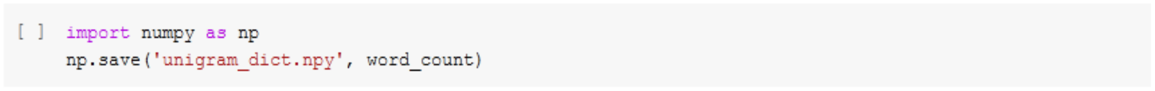


```
[ ]  import nltk
     nltk.download('punkt')
     cleaned_text=extra_space(text_corpus1)
     print("Removed Extra Spaces")
     cleaned_text=sp_charac(cleaned_text)
     print("Removed Special Caracters")
     cleaned_text=tokenize_text(cleaned_text)

     [nltk_data] Downloading package punkt to /root/nltk_data...
     [nltk_data]   Package punkt is already up-to-date!
     Removed Extra Spaces
     Removed Special Caracters
```

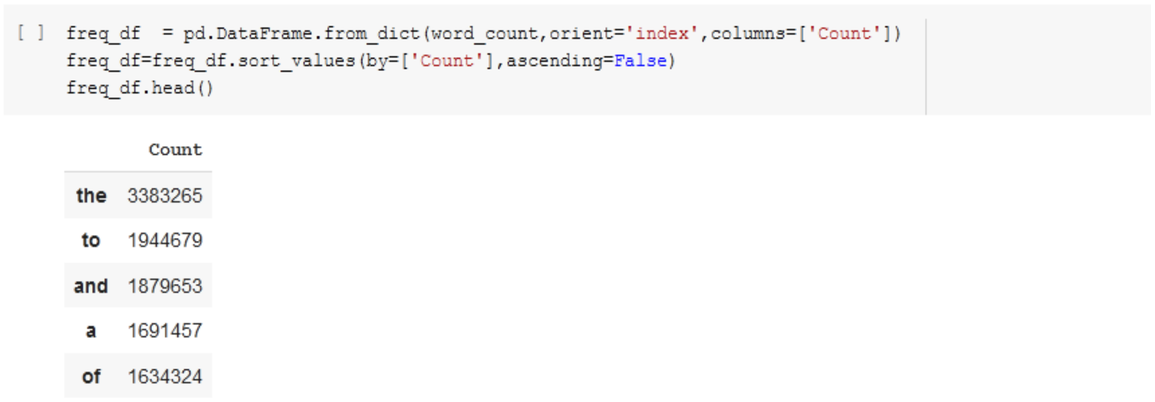
- Creating dictionary of unigrams with stop words



```
[ ] word_count={}
    for word in cleaned_corpus:
        if word not in word_count:
            word_count[word]=0
        word_count[word]+=1
```



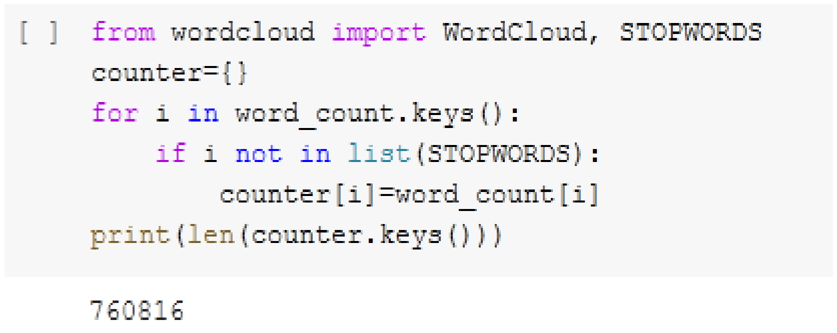
```
[ ] import numpy as np
    np.save('unigram_dict.npy', word_count)
```



```
[ ] freq_df  = pd.DataFrame.from_dict(word_count,orient='index',columns=['Count'])
    freq_df=freq_df.sort_values(by=['Count'],ascending=False)
    freq_df.head()
```

	Count
the	3383265
to	1944679
and	1879653
a	1691457
of	1634324

- Creating dictionary of unigrams without stopwords



```
[ ] from wordcloud import WordCloud, STOPWORDS
    counter={}
    for i in word_count.keys():
        if i not in list(STOPWORDS):
            counter[i]=word_count[i]
    print(len(counter.keys()))

    760816
```

Visual Data Analysis

Data Visualization

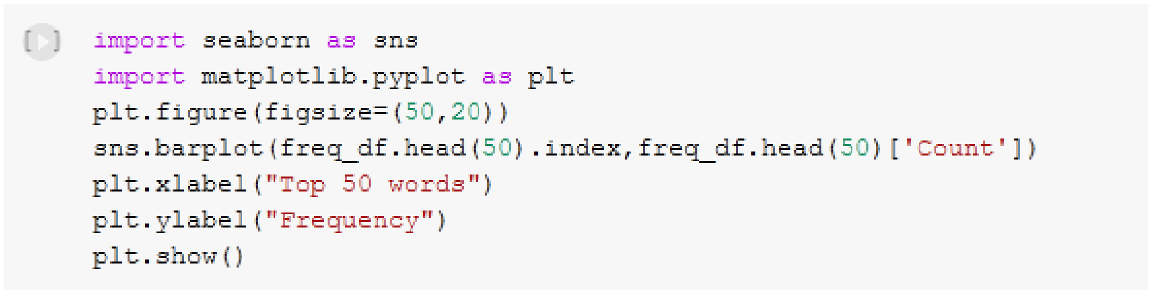
To get an idea about the corpus, we carried out certain visualizations after cleaning the data

Data Cleaning

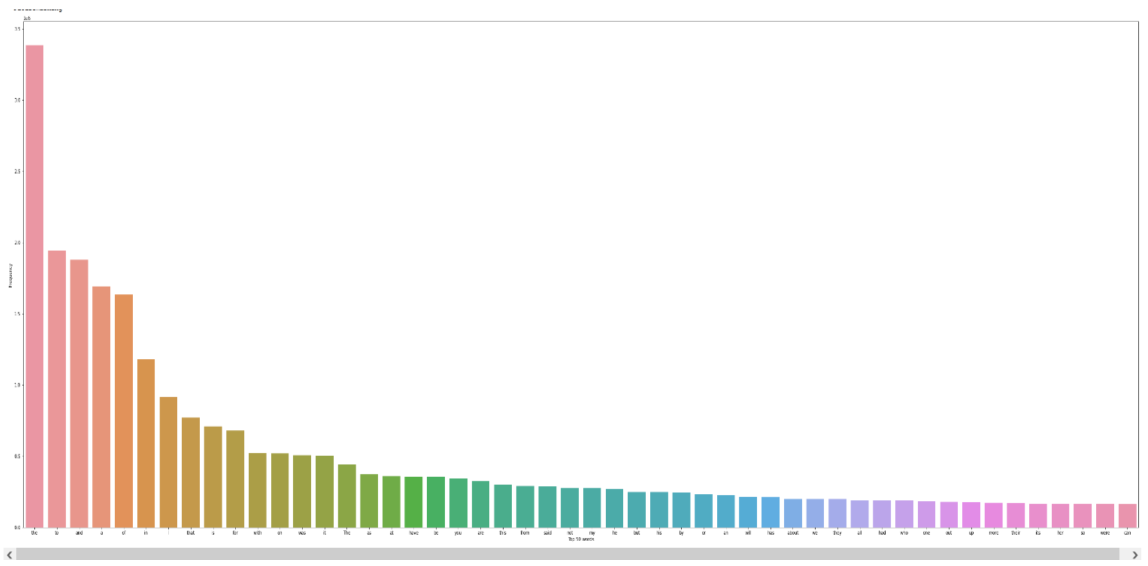
For cleaning, we have carried out the following steps:

- **Removing Extra space**: All the extra space between the words are removed.
- **Removing Special Characters**: Any form special character including punctuations are removed.
- **Tokenizing text data**: News and Blogs data are tokenized using ***nltk Tokenizer***.

Code Sample:

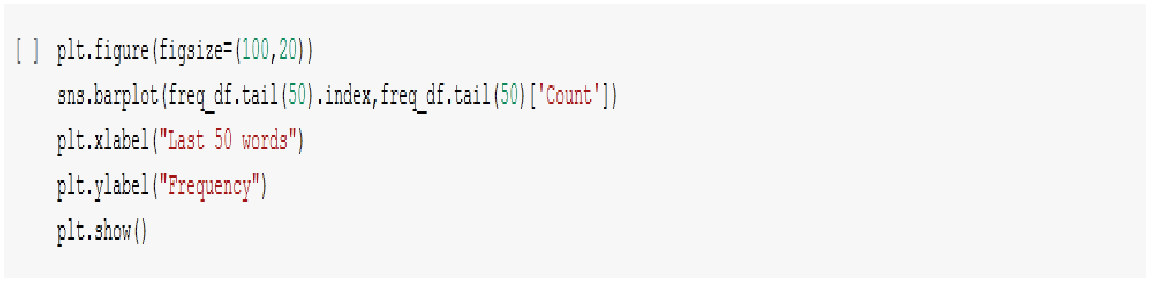


```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(50,20))
sns.barplot(freq_df.head(50).index,freq_df.head(50)['Count'])
plt.xlabel("Top 50 words")
plt.ylabel("Frequency")
plt.show()
```

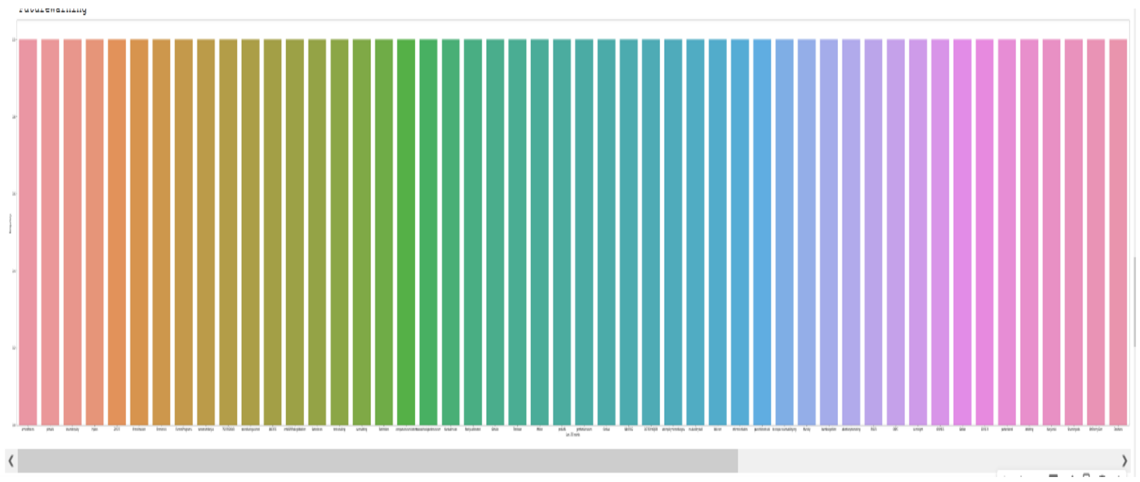


- Visualizing the top 50 words based on their frequency of occurrence
- Most of them are stop words as they are used frequently.

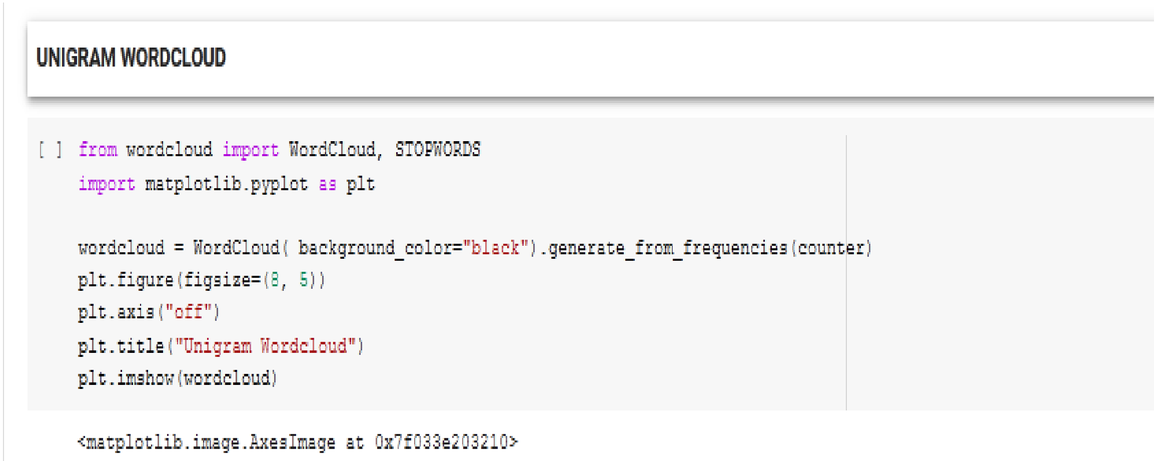
Visualising the top 50 words based on their frequency of occurrence Most of them are stopwords as they are used frequently .



```
[ ] plt.figure(figsize=(100,20))
    sns.barplot(freq_df.tail(50).index,freq_df.tail(50)['Count'])
    plt.xlabel("Last 50 words")
    plt.ylabel("Frequency")
    plt.show()
```



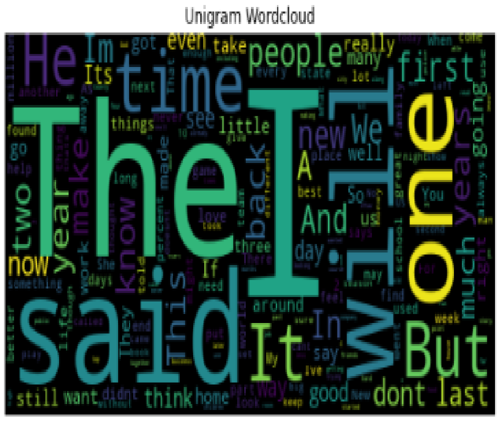
- Visualizing the last 50 words based on their frequency
- All of them occur once in the entire corpus



UNIGRAM WORDCLOUD

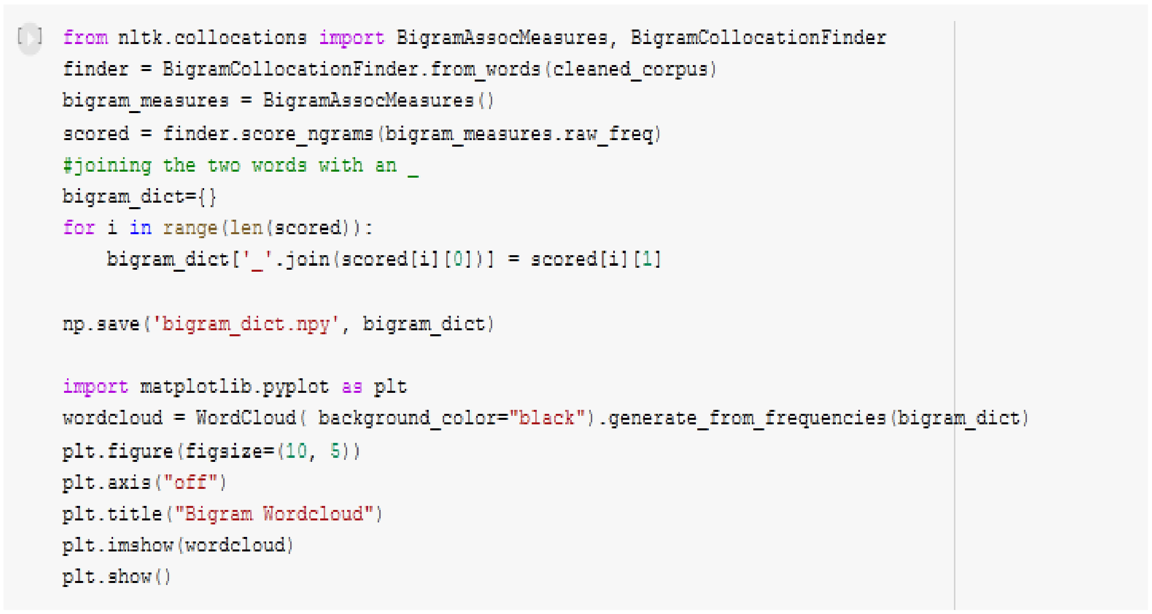
```
[ ]  from wordcloud import WordCloud, STOPWORDS
     import matplotlib.pyplot as plt

     wordcloud = WordCloud( background_color="black").generate_from_frequencies(counter)
     plt.figure(figsize=(8, 5))
     plt.axis("off")
     plt.title("Unigram Wordcloud")
     plt.imshow(wordcloud)
```

```
<matplotlib.image.AxesImage at 0x7f033e203210>
```



- Generated the word cloud with unigram words.
- There are no stop words here as I was trying to visualize most commonly used non-stop words.

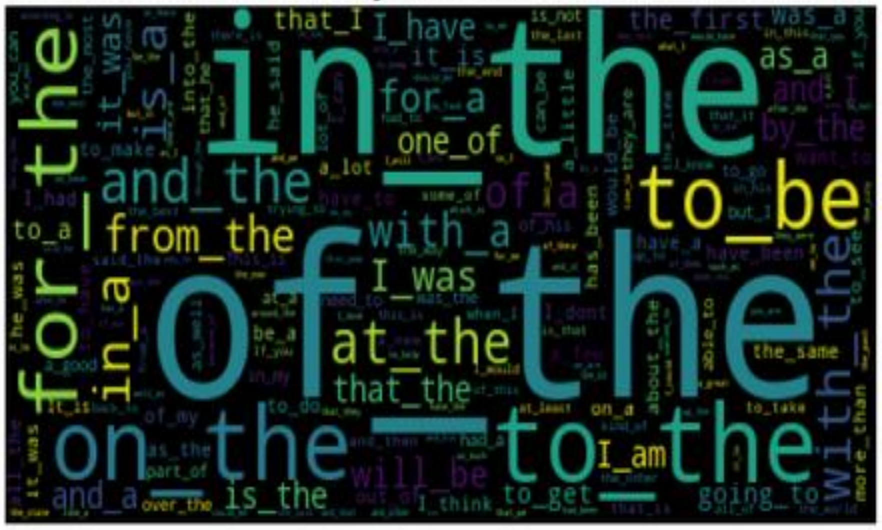
BIGRAM WORDCLOUD



```
from nltk.collocations import BigramAssocMeasures, BigramCollocationFinder
finder = BigramCollocationFinder.from_words(cleaned_corpus)
bigram_measures = BigramAssocMeasures()
scored = finder.score_ngrams(bigram_measures.raw_freq)
#joining the two words with an _
bigram_dict={}
for i in range(len(scored)):
    bigram_dict['_'.join(scored[i][0])] = scored[i][1]

np.save('bigram_dict.npy', bigram_dict)

import matplotlib.pyplot as plt
wordcloud = WordCloud( background_color="black").generate_from_frequencies(bigram_dict)
plt.figure(figsize=(10, 5))
plt.axis("off")
plt.title("Bigram Wordcloud")
plt.imshow(wordcloud)
plt.show()
```

- Here I have included the stop words otherwise the meaning of the phrases will change.
- There is a lot of preposition use as observed from the word cloud

Bigram Wordcloud



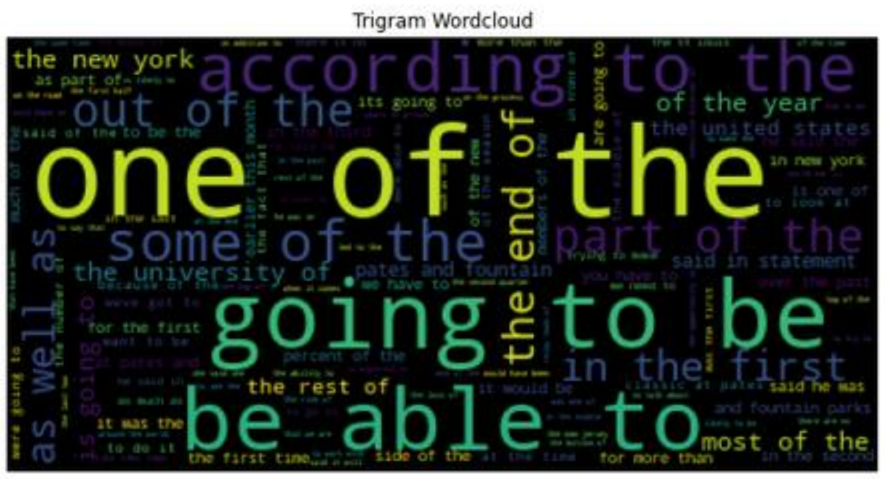
There is a lost of preposition use as observed from the word cloud

```
[ ] data=" ".join(cleaned_corpus[:100000])

    from sklearn.feature_extraction.text import CountVectorizer
    vectorizer = CountVectorizer(ngram_range=(3,3),stop_words=None)
    X = vectorizer.fit_transform([data])
    vocab = vectorizer.vocabulary_
    print("Vectorized the data : ")
    count_values = X.toarray().sum(axis=0)
    print("Creating Trigram Dictionary")
    trigram_dict={}
    for ng_count, ng_text in sorted([(count_values[i],k) for k,i in vocab.items()], reverse=True):
        trigram_dict[ng_text]=ng_count
```

```
Vectorized the data :
Creating Trigram Dictionary
```

```
[ ] np.save('trigram_dict.npy', trigram_dict)
```

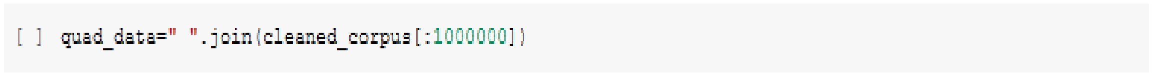
```
[ ] import matplotlib.pyplot as plt
    from wordcloud import WordCloud, STOPWORDS
    wordcloud = WordCloud( background_color="black").generate_from_frequencies(trigram_dict)
    plt.figure(figsize=(10, 5))
    plt.axis("off")
    plt.title("Trigram Wordcloud")
    plt.imshow(wordcloud)
    plt.show()
```



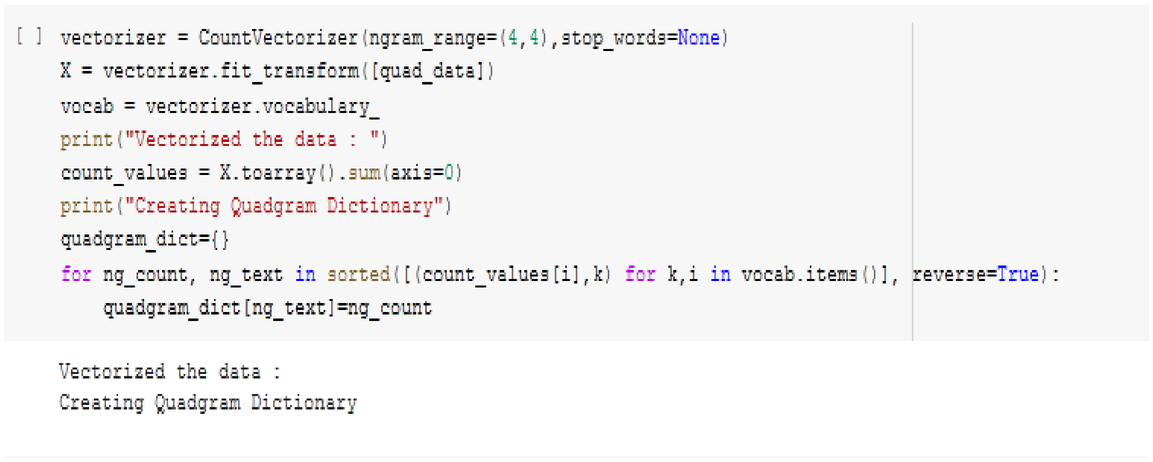
Most commonly used phrases

- ‘one of the’
- ‘out of the’
- ‘be able to’
- ‘some of the’
- ‘going to be’

QUADGRAM WORDCLOUD

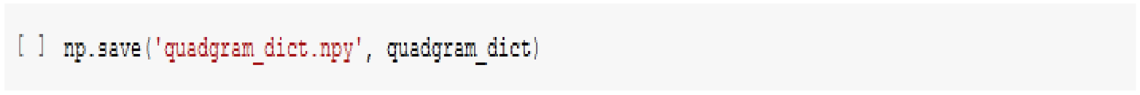


```
[ ] quad_data=" ".join(cleaned_corpus[:1000000])
```

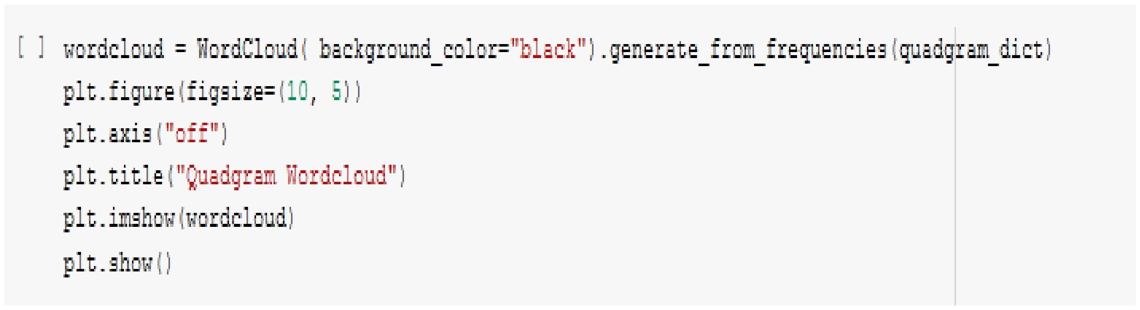


```
[ ] vectorizer = CountVectorizer(ngram_range=(4,4),stop_words=None)
    X = vectorizer.fit_transform([quad_data])
    vocab = vectorizer.vocabulary_
    print("Vectorized the data : ")
    count_values = X.toarray().sum(axis=0)
    print("Creating Quadgram Dictionary")
    quadgram_dict={}
    for ng_count, ng_text in sorted([(count_values[i],k) for k,i in vocab.items()], reverse=True):
        quadgram_dict[ng_text]=ng_count
```

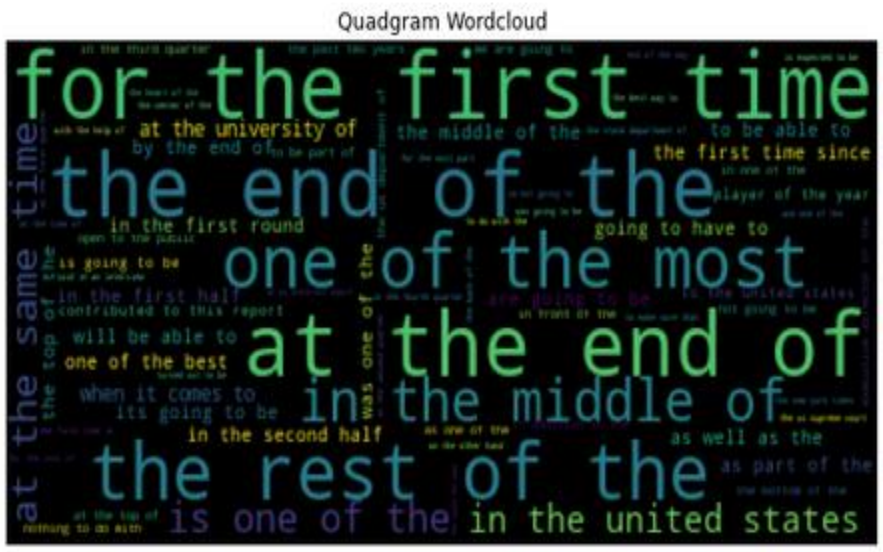
```
Vectorized the data :
Creating Quadgram Dictionary
```



```
[ ] np.save('quadgram_dict.npy', quadgram_dict)
```



```
[ ] wordcloud = WordCloud( background_color="black").generate_from_frequencies(quadgram_dict)
    plt.figure(figsize=(10, 5))
    plt.axis("off")
    plt.title("Quadgram Wordcloud")
    plt.imshow(wordcloud)
    plt.show()
```



Most commonly used phrases

- ‘in the middle of’
- ‘one of the most’
- ‘the rest of the’
- ‘in the middle of’
- ‘at the end of’

Data Analysis Findings

Probabilistic Approach

In probabilistic approach, next word is predicted based on the probability of its occurrence given a set of input words.

Under this section, we have 2 models.

Markov Model-

Key Idea:

1. The sequence of words (history) is taken whose next word has to be predicted.
2. If length of history = 1, then we look for it in unigram dictionary keys. The word having highest probability corresponding to the history key in the dictionary is the next word. If it is not found in the keys then I have generated a random next word.
3. If length of history = 2 , then we look for it in bigram dictionary keys .The word having highest probability corresponding to the history key in the dictionary is the next word .If it is not found in the keys then I take history = history[-1:]and repeat step 2 .
4. If length of history = 3 , then we look for it in trigram dictionary keys .The word having highest probability corresponding to the history key in the dictionary is the next word .If it is not found in the keys then I take history = history[-2:]and repeat step 3 .
5. If length of history > 3, then I take history = history [-3:] and repeat step 4.

GPT- 1 Model**- Next Word Prediction**

There are 3 different ways for prediction of the next word.

1. **Greedy Search** : chooses the best possible next word based on highest probability from 1 hypothesis
2. **Beam Search** : chooses the high probability next word from n hypothesis
3. **Random Sampling**: chooses random next word from possible hypothesis, however as the temperature is set high, it will ignore low probability words.

1. Classification Approach

In classification approach, next word is treated as a class to be predicted. Thus the problem turns into a Multi-Class Classification Problem.

Under this section, we have 2 model.

Stacked LSTM with and without Attention Mechanism-

Key Idea:

1. The sequence of words (history) is taken as input whose next word has to be predicted.
2. If length of history = 1, then we pass it to the model corresponding to sequence length 2 and predict the class with highest probability.
3. If length of history < 4, then we pass it to the model corresponding to sequence length 4 and predict the class with highest probability.
4. If length of history < 7, then we pass it to the model corresponding to sequence length 7 and predict the class with highest probability.
5. If length of history ≥ 7, then I take history = history [-6:] and repeat step 4.

ALBERT Model- There is a certain format in which BERT models take input. It takes in 3 inputs.

- **Encoded Data:** In this type of input, the sequence of words sent as input are tokenized and encoded using ALBERT tokenizer. Also as it is a classification based approach, a [CLS] should be added at the beginning of each sequence.
- **Masked Input data**: When we give data to any version of BERT model, we need to specify max_length of the input. If length of the sequence of words is less than the max_length then we have to pad it. The mask allows the model to cleanly differentiate between the content and the padding. The mask has the same shape as the ***encoded data***, and contains a 1 anywhere the ***encoded data*** is not padding.
- **Input Type**: Generally BERT is used for next sentence prediction where 2 sentences are given as input. So in this input data, the non-padded region contains a 0 or a 1 indicating which sentence the token is a part of. But here there is only sentence, so it will have all 0s.

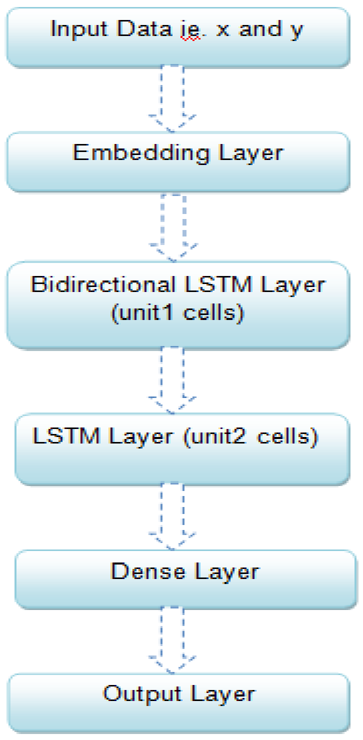
Although the test loss for Al-BERT is lowest, it produces very bad predictions.
So we decided to consider LSTM as the final model for predictions.

Model Selection

Long-short-term memory models or LSTMs are used to solve the problem of short term memory by using gates that regulate the flow of information. These models have mechanisms that decide whether or not to keep information, thereby being able to retain important information over a long time. Due to their ability to learn long-term dependencies, they are extensively used in machine translation, speech recognition, handwriting recognition and generation, language modeling and translation, speech synthesis, and many other deep learning tasks.

Stacked LSTM Model

It is a model developed by stacking a Bidirectional LSTM layer over a LSTM layer .Following is the architecture for the LSTM Model.



Model Architecture

The output layer has a softmax activation, so I have got probability distribution of the output classes.

N-Gram Language Model

In language models , either probabilities are assigned to a series of words or probabilities are assigned to the next word given some preceding words.

Here 2 scenarios are discussed:

1st Scenario- In case of a sentence (w1 w2 w3 w4 w5) ,the probability for the sentence is given by P(w1)*P(w2)*P(w3)*P(w4)*P(w5) where P(wn) is the probability of

wn.P(wn)= (Number of occurrences of wn in the corpus) / (Total number of words in the corpus)

2nd Scenario - In case of a sentence (w1 w2 w3 w4 w5) ,the probability for the sentence is given by

P(w1)*P(w2|w1)*P(w3|w1w2)*P(w4|w1w2w3)*P(w5|w1w2w3w4)

where P(wn|wn-1) is theprobability of wn given the preceding word is

wn-1.P(wn|wn-1)= P(wn-1 wn)/P(wn-1) that is Probability of occurrence of (wn-1 wn) in the corpus divided by Probability of occurrence of wn-1

.N- Gram models can be of different types. Those are unigram, bigram, trigram and so on.

Unigram Model - In this model, the probability of each word in the sentence is given by P(wi) = Count of wi in the text corpus / total words .

Bigram Model - This is similar to the 2nd scenario discussed above. But here we take into account only the previous word.

In the case of a sentence (w1 w2 w3 w4 w5)

the probability for the sentence is given by

P(w1)*P(w2|w1)*P(w3|w2)*P(w4|w3)*P(w5|w4)

where P(wn|wn-1) is the probability of wn given the preceding word is wn-1

.

Trigram Model - This is similar to the bigram model. But here we take intoaccount 2 previous words.Thus the expression for n-gram model is

P(wi|w1w2...wi-1)=P(wi| w(i-n+1)….wi-1) .

This is also called Markov assumption.

Now to deal with 0 probability, Back off Algorithm is used.

If higher order n-gram results in 0, then we back off to lower order n-gram.

It is given by the following formula.

S(wi|wi-k+1i-1)= f(wii−k+1)/ f(wi−1i−k+1) if f(wii−k+1) > 0 αS(wi|wi−1i−k+2)

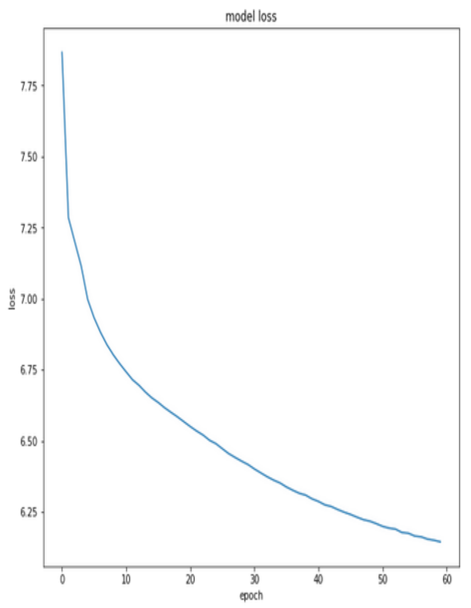
otherwise

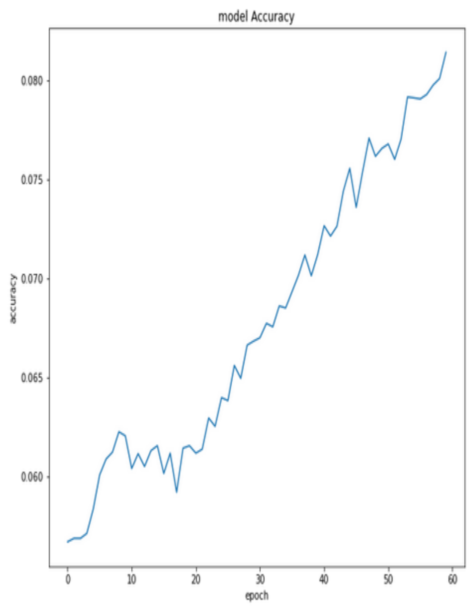
For example: Let us consider a sequence of words “this is a very beautiful” .Here we have to find probability of “beautiful” given “this is a very”. Let's say "beautiful" never occurred in the context "this is a very" so for the 4-grams model "beautiful" has probability 0 . So we backoff to 3-gram model and find the probability as α*P(“beautiful”|”is a very”).

Model Evaluation

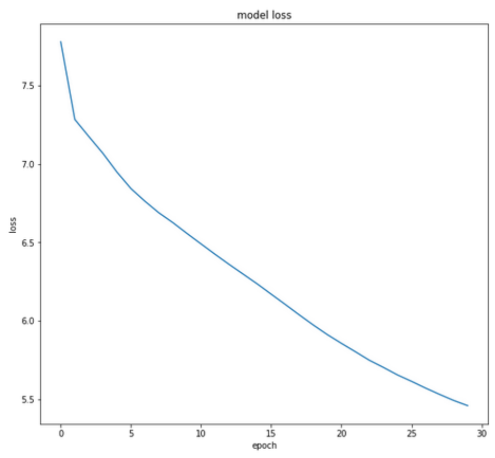
Train Results

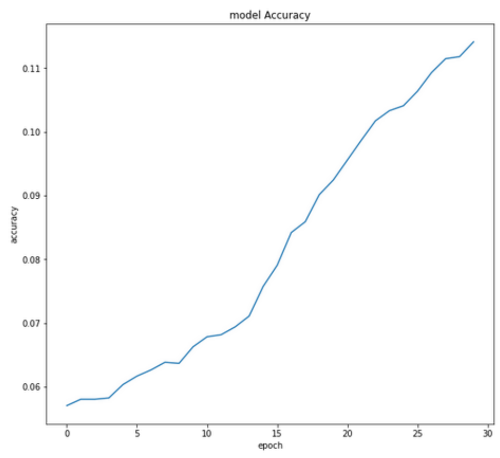
For Sequence of length 2:



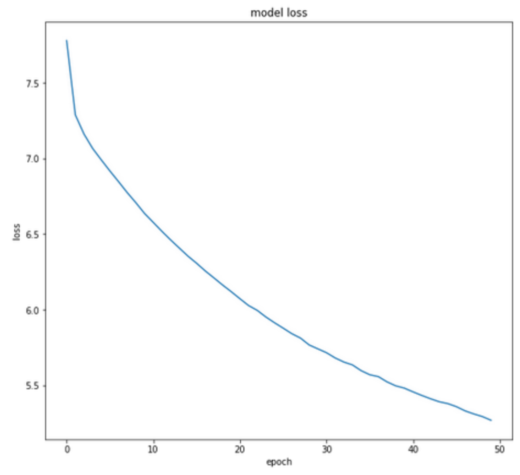


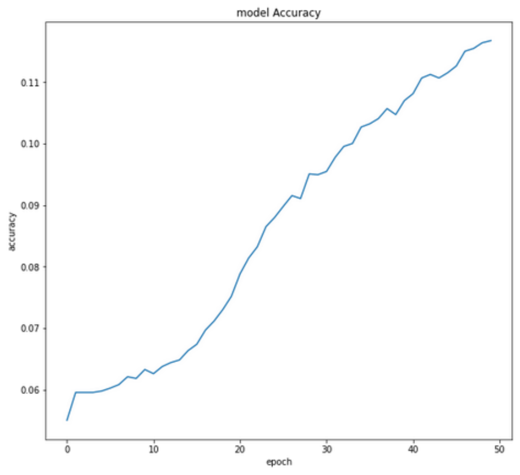
For Sequence of length 4:





For Sequence of length 7:

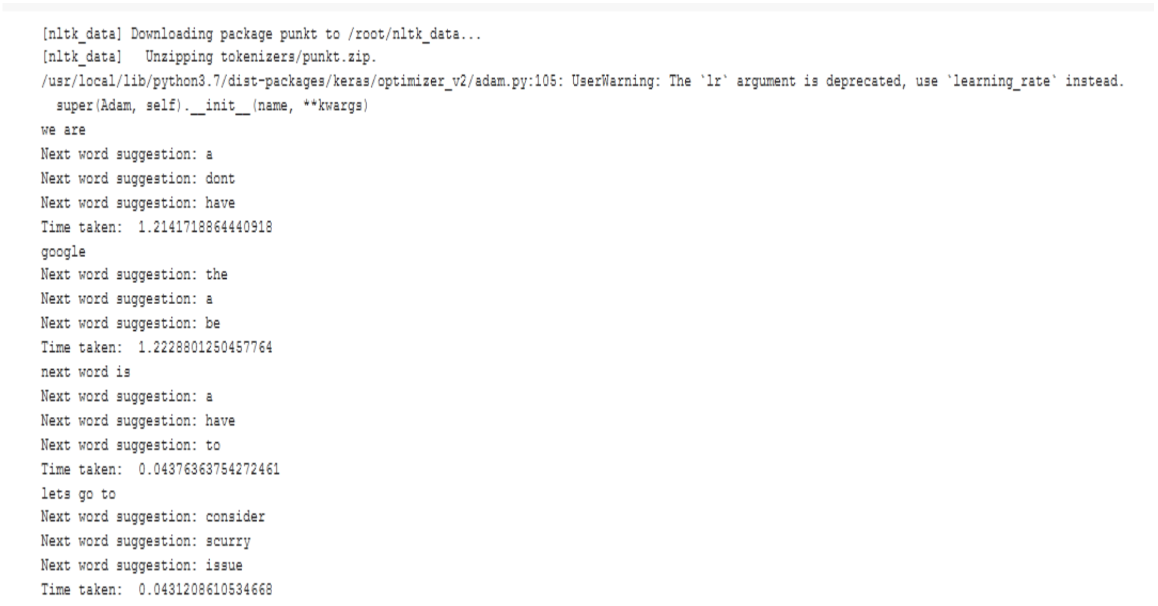




Test Results

- Sequence 2- Loss : 9.2959
- Sequence 4- Loss : 8.7960
- Sequence 7- Loss : 10.7551

Some prediction Example.



```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
we are
Next word suggestion: a
Next word suggestion: dont
Next word suggestion: have
Time taken:  1.2141718864440918
google
Next word suggestion: the
Next word suggestion: a
Next word suggestion: be
Time taken:  1.2228801250457764
next word is
Next word suggestion: a
Next word suggestion: have
Next word suggestion: to
Time taken:  0.04376363754272461
lets go to
Next word suggestion: consider
Next word suggestion: scurry
Next word suggestion: issue
Time taken:  0.0431208610534668
```

Overall the model predicting is pretty well, but there is scope for improvement as the losses are high.

Model Performance on New Data

As we have a trained model, we can use it to get the next word predictions. To do this, first we have to create a word map from the predictions to the actual word using which we can find the word corresponding to a certain prediction. We then write a function to return the next *n* words greedily.

Project Notes

Notes when you build your final notebook:
1. you should not train any model either it can be a ML model or DL model or Count Vectorizer or even simple Standard Scalar
2. You should not read train data files
3. The function1 takes only one argument “X” (a single data points i.e. 1*d feature) and the inside the function you will preprocess data point similar to the process you did while you featurize your train data

a. Ex: consider you are doing taxi demand prediction case study (problem definition: given a time and location predict the number of pickups that can happen)
b. so in your final notebook, you need to pass only those two values
c. def. final(X):

4. You must save the trained model for future references.

5. While creating the model we must consider the memory constraints.
Preprocess data i.e. data cleaning, filling missing values etc.
compute features based on this X

Conclusion

- Loading and processing the dataset is time consuming. Analysis was done on a subset of the real data.
- Stop words are very important, they are fundamental part of language. We have to test and maybe include these words to our prediction algorithm.
- Larger the dataset greater the accuracy of prediction as the model exposes to wide range of words.
- SwiftKey Data also contains corpus of many different languages, we can work on them.
- LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series.
- LSTM has wide range of application such as:

  ⇒ Robot control
  ⇒ Time series prediction
  ⇒ Speech recognition
  ⇒ Rhythm learning
  ⇒ Music composition
  ⇒ Grammar learning
  ⇒ Handwriting recognition
  ⇒ Human action recognition
  ⇒ Sign language translation

References

- https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-language-model-nlp-python-code/
- https://thecleverprogrammer.com/2020/07/20/next-word-prediction-model/
- https://www.youtube.com/watch?v=BAN3NB_SNHY
- https://thinkinfi.com/fasttext-word-embeddings-python-implementation/
- https://www.youtube.com/watch?v=gHC9tRyVSNE
- https://medium.com/swlh/language-modelling-with-nltk-20eac7e70853
- https://stackoverflow.com/questions/54978443/predicting-missing-words-in-a-sentence-natural-language-processing-model
- https://huggingface.co/blog/how-to-generate

Appendix (Code & Code Output with Proper Documentation)

Index.html

```
<!DOCTYPE html>
<html >
<!--From https://codepen.io/frytyler/pen/EGdtg-->
<head>
<meta charset="UTF-8">
<title>ML API</title>
<link href='https://fonts.googleapis.com/css?family=Pacifico' rel='stylesheet'
type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Arimo' rel='stylesheet'
type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Hind:300' rel='stylesheet'
type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300'
rel='stylesheet' type='text/css'>
<link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">

</head>

<body>
<div class="login">
<h2>SwiftKey Text Prediction</h2>

<!-- Main Input For Receiving Query to our ML -->
<form action="{{ url_for('predict')}}"method="post">
<input type="text" name="data" placeholder="enter text here" attribute />

<button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
</form>

<br>
<br>
{{ prediction_text }}

</div>
</body>
</html>
```

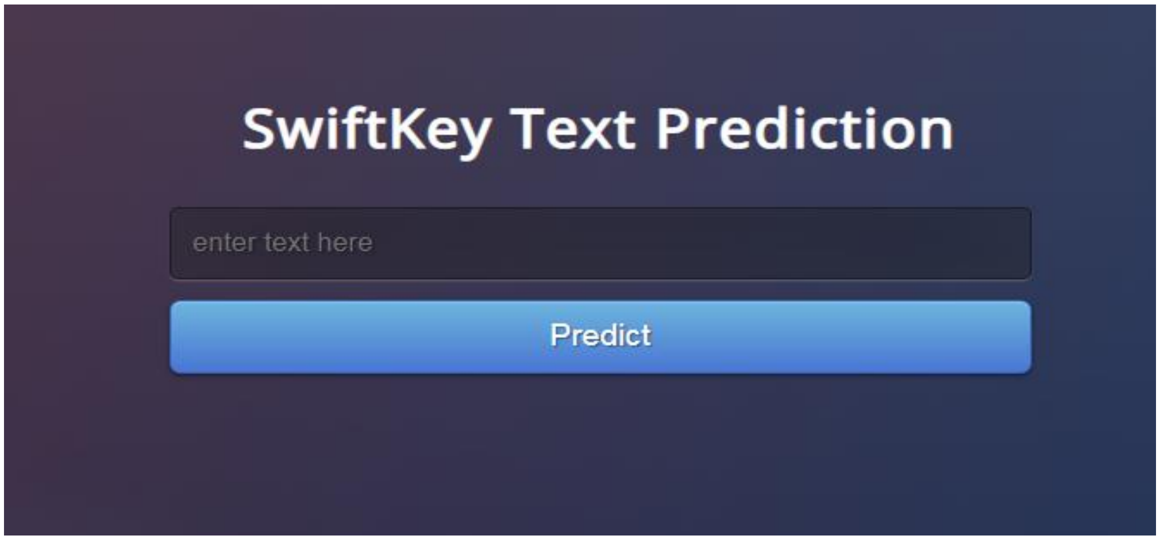
App.py

```
import numpy as np
from flask import Flask, request, jsonify, render_template

app = Flask(__name__)

import re
from nltk.tokenize import word_tokenize
def extra_space(text):
    new_text= re.sub("\s+"," ",text)
    return new_text
def sp_charac(text):
    new_text=re.sub("[^0-9A-Za-z ]", "" , text)
    return new_text
def tokenize_text(text):
    new_text=word_tokenize(text)
    return new_text

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict',methods=['POST'])
def predict():
    '''
    For rendering results on HTML GUI
    '''
    data = request.form.get("data")
    prediction = predict_next(data)
    output = prediction

    return render_template('index.html', prediction_text='Next word suggestions are:
{}'.format(output))

@app.route('/predict_next',methods=['POST'])
def predict_next(data):

    import tensorflow as tf
    tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
    from tensorflow.keras.preprocessing.sequence import pad_sequences
    from tensorflow.keras.models import load_model
    from tensorflow.keras.optimizers import Adam
    import pickle
    import time

    with open('tokenizer1_len7.pickle', 'rb') as handle:
        tokenizer_len7 = pickle.load(handle)

    with open('tokenizer1_len4.pickle', 'rb') as handle:
        tokenizer_len4 = pickle.load(handle)

    with open('tokenizer1_len2.pickle', 'rb') as handle:
        tokenizer_len2 = pickle.load(handle)

    file="lstm_len7.hdf5"
    model_len7 = load_model(file)
    model_len7.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.001),
metrics=['accuracy'])

    file="lstm_len4.hdf5"
    model_len4 = load_model(file)
    model_len4.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.001),
metrics=['accuracy'])

    file="lstm_len2.hdf5"
    model_len2 = load_model(file)
    model_len2.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.001),
metrics=['accuracy'])

    text=data
    start= time.time()
    cleaned_text=extra_space(text)
    cleaned_text=sp_charac(cleaned_text)
    tokenized=tokenize_text(cleaned_text)

    line = ' '.join(tokenized)

    if len(tokenized)==1:

        encoded_text = tokenizer_len2.texts_to_sequences([line])
        pad_encoded = pad_sequences(encoded_text, maxlen=1, truncating='pre')
        final1=''
        for i in (model_len2.predict(pad_encoded)[0]).argsort()[-3:][::-1]:

            pred_word1 = tokenizer_len2.index_word[i]
            print("Next word suggestion:",pred_word1)
            final1=pred_word1+" ,"+final1
        return  final1

    elif len(tokenized)<4:
        encoded_text = tokenizer_len4.texts_to_sequences([line])
        pad_encoded = pad_sequences(encoded_text, maxlen=3, truncating='pre')
        final2=''
        for i in (model_len4.predict(pad_encoded)[0]).argsort()[-4:][::-1]:


            pred_word2 = tokenizer_len4.index_word[i]
            print("Next word suggestion:",pred_word2)
            final2=pred_word2+" ,"+final2

        return final2

    else:
        encoded_text = tokenizer_len7.texts_to_sequences([line])
        pad_encoded = pad_sequences(encoded_text, maxlen=6, truncating='pre')
        final3=''
        for i in (model_len7.predict(pad_encoded)[0]).argsort()[-5:][::-1]:

            pred_word3 = tokenizer_len7.index_word[i]
            print("Next word suggestion:",pred_word3)
            final3=pred_word3+" ,"+final3

        return final3
    print('Time taken: ',time.time()-start)

@app.route('/predict_api',methods=['POST'])
def predict_api():
    '''
    For direct API calls trought request
    '''
    data = request.get_json(force=True)
    prediction = predict_next(data)

    output = prediction
    return jsonify(output)

if __name__ == "__main__":
    app.run(debug=True)
```

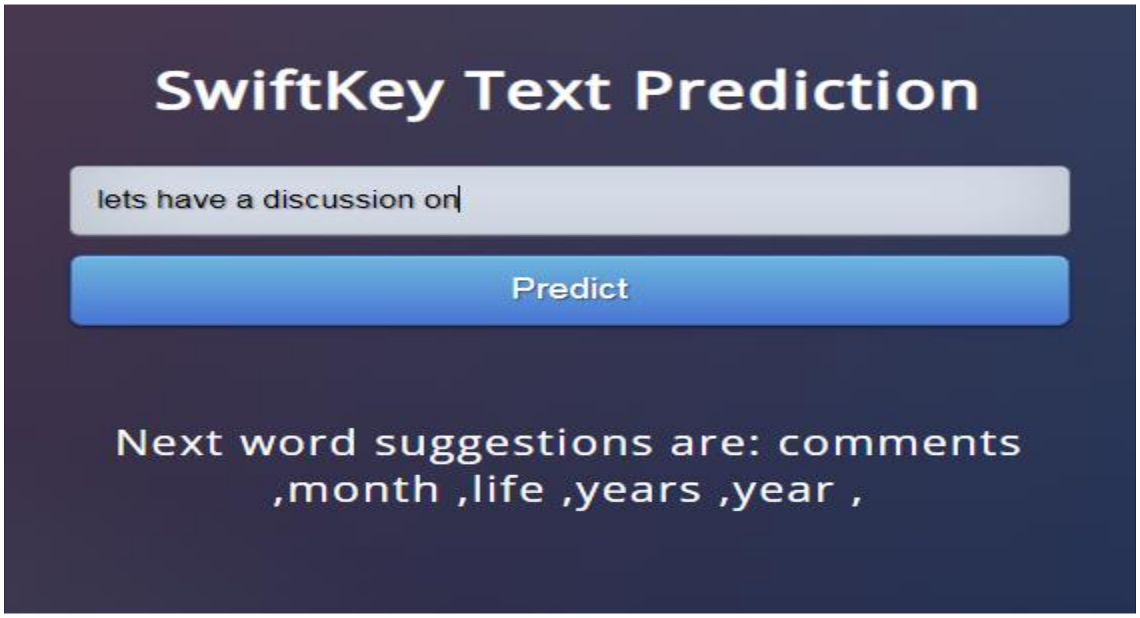
request.py:

```
import requests

url = 'http://localhost:5000/predict_api'
r = requests.post(url,json={'experience':2, 'test_score':9, 'interview_score':6})

print(r.json())
```

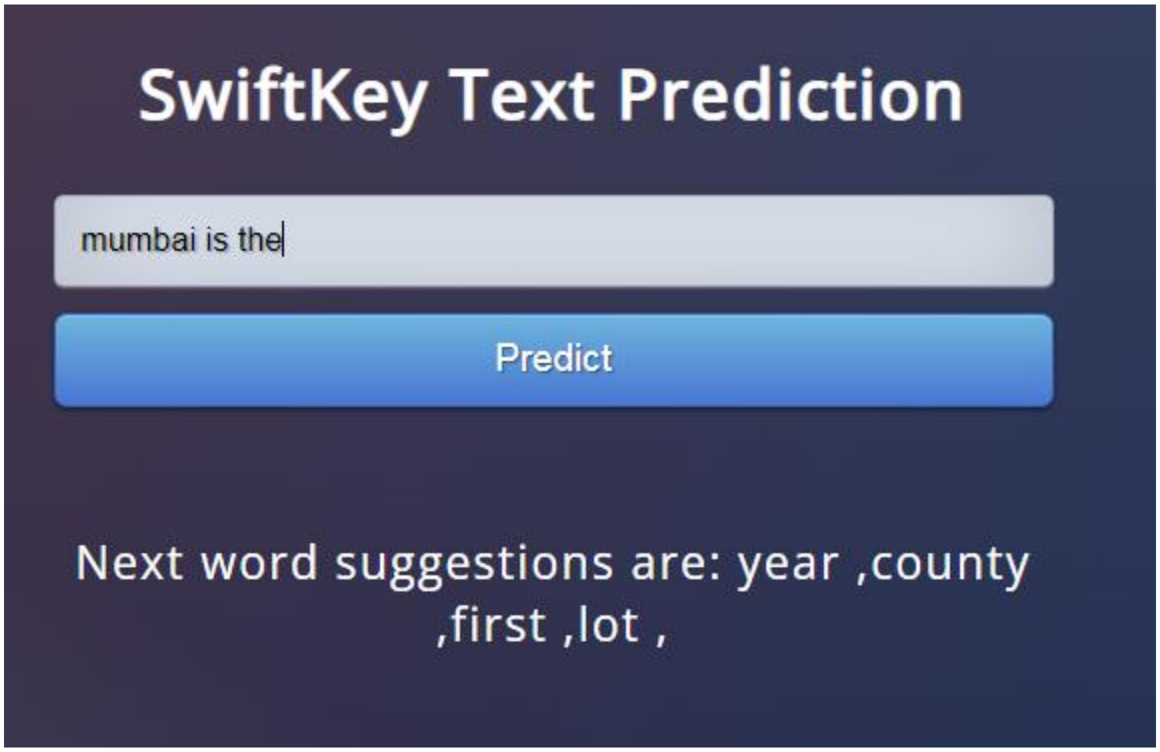
UI Design:



Word prediction based on input:



Next word suggestions are: comments ,month ,life ,years ,year ,



SwiftKey Text Prediction

mumbai is the

Next word suggestions are: year ,county ,first ,lot ,