AUTOMATIC TEXT COMPLETION USING NLP

Submitted in partial fulfillment of the requirements of the degree of

Bachelor of Engineering

by

AHAD KHAN	**32**
PIYUSH SHENDE	**64**
BINOY BANGERA	**85**

Supervisor:

PROF. UDAY NAYAK

Department of Information Technology
Don Bosco Institute of Technology
University of Mumbai

2020-2021

AUTOMATIC TEXT COMPLETION USING NLP

Submitted in partial fulfillment of the requirements of the degree of

Bachelor of Engineering

by

AHAD KHAN	**32**
PIYUSH SHENDE	**64**
BINOY BANGERA	**85**

Supervisor:

PROF. UDAY NAYAK

Department of Information Technology
DON BOSCO INSTITUTE OF TECHNOLOGY
(Affiliated to the University of Mumbai)
Premier Automobiles Road, Kurla, Mumbai – 400070

2020-2021

Department of Information Technology
DON BOSCO INSTITUTE OF TECHNOLOGY
(Affiliated to the University of Mumbai)
Premier Automobiles Road, Kurla, Mumbai – 400070

CERTIFICATE

This is to certify that the project entitled **"AUTOMATIC TEXT COMPLETION SYSTEM USING NLP"** is a bonafide work of

AHAD KHAN	**32**
PIYUSH SHENDE	**64**
BINOY BANGERA	**85**

submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of **Undergraduate** in **Bachelor of Information Technology**

Date: 21-05-2021

(Prof. Uday Nayak)
Supervisor

(Prof. Janhavi B.)
HOD, IT Department

(Dr. Prasanna Nambiar)
Principal

Department of Information Technology
DON BOSCO INSTITUTE OF TECHNOLOGY
(Affiliated to the University of Mumbai)
Premier Automobiles Road, Kurla, Mumbai – 400070

Project Report Approval for B.E.

This project report entitled **"AUTOMATED TEXT COMPLETION SYSTEM USING NLP"** by **AHAD KHAN, PIYUSH SHENDE & BINOY BANGERA** is approved for the degree of **Bachelor of Engineering inInformation Technology**

(Examiner's Name and Signature)

1. **Dr. Sujata Kohle**

2. ______________________________

(Supervisor's Name and Signature)

1. **Prof. Sunantha Krishnan**

(Chairman)

1. ______________________________

Date: 21-05-2021

Place: Kurla, Mumbai

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea / data / fact / source in my submission. I under- stand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

AHAD KHAN	Signature: ________________
PIYUSH SHENDE	Signature: ________________
BINOY BANGERA	Signature: ________________

Date: 21-05-2021

Place: Kurla, Mumbai

Abstract

Imagine that you were a representative replying to customer online and you are asking more or less the same questions over and over to your customer. Would you like to get automatic suggestions instead of typing the same thing again and again? An autocomplete can be helpful, faster, and convenient and also correct any grammatical / spelling error at the same time. In the jupyter notebook in this project, we select a history of sentences written by the representatives and the customer, format and correct them using a few regex rules and count them so we can estimate their frequency and likeliness to be useful again. After the calculation of a similarity matrix based on the sklearn tfidf tool (frequency and normalization of words), we use this matrix to calculate the similarity between the new few words written by the representative and the history of messages written in the past. The Autocomplete will recognize the closest sentences and rank 3 final proposals:

Keywords: normalization, similarity matrix, regex rules, frequency, tfid,Sklearn

Contents

List of Figures

4.3 List of Tables

Introduction

1.1. Problem Statement

Imagine that you were a representative replying to customer online and you are asking more or less the same questions over and over to your customer. Would you like to get automatic suggestions instead of typing the same thing again and again? An autocomplete can be helpful, faster, and convenient and also correct any grammatical / spelling error at the same time.

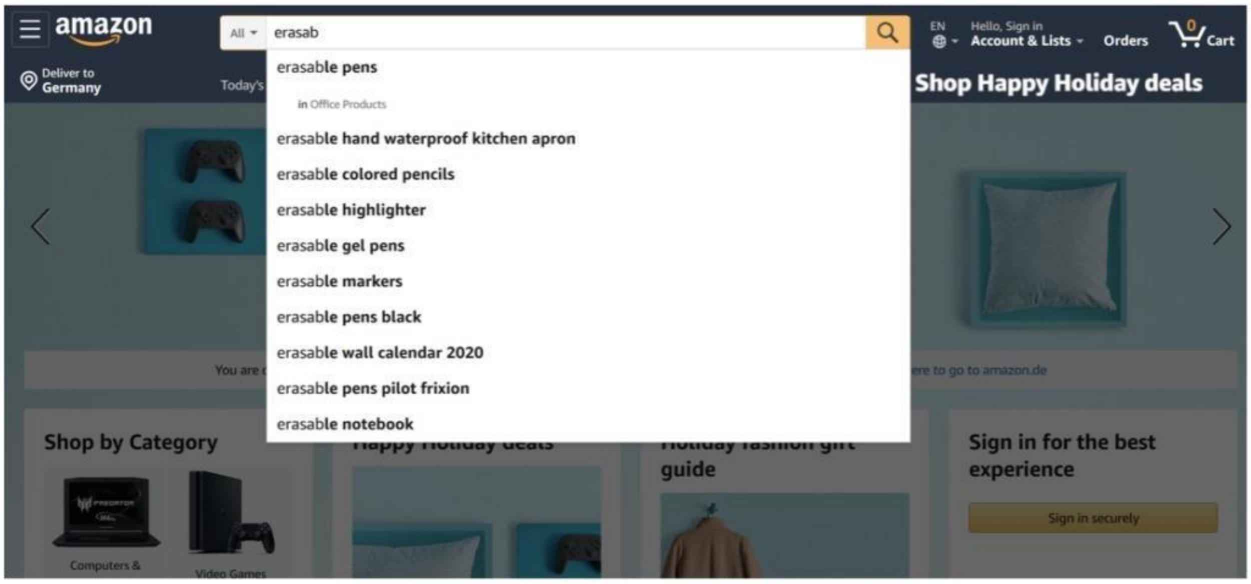
1.2. Scope of the Project

A word completion system that can automatically predict unrestricted word completions for professional use that reduces jargon complexity as well as saves time. This project relates generally to the field of data entry systems and, more particularly, to automated word completion systems for operating with unstructured data files, such as word processing documents and e-mail messages.

1.3. Current Scenario

Autocomplete is a pattern used to display query suggestions.

As you type, the search engine will suggest several predictions of how your query could be completed. For example, if you type in “course” on a n university webpage, it suggests “courses in English”, “courses in geography” or “course catlog”. The beauty of autocomplete is that it also fills up unfinished words, so if you type in “lect”, the engine would suggest “lecture” or “lecturer”.

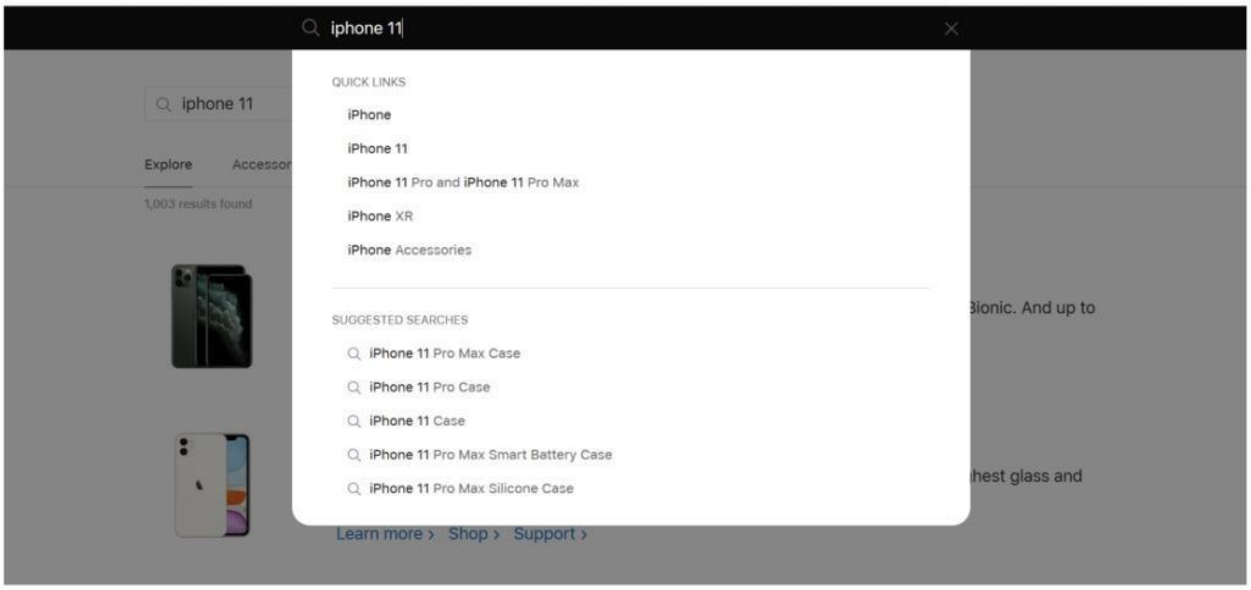


Autocomplete is especially powerful for mobile devices and for e-commerce marketplaces. No matter if your visitors use mobile or desktop, autocomplete is always a great feature because it saves your users time. On average, autocomplete reduces typing about 25%.
Most users don’t want to type as much on their mobile devices. The screen is limited, and it is harder to type than on a desktop. An autocomplete option minimizes the number of characters users have to type which also means that there will be ewer types. When you use autosuggest on your mobile devices it is easier to show suggestions instead of displaying full results already that would take up more spaces.

If you offer an ecommerce site, autocomplete is pretty much a necessity. 82% of the top grossing e-commerce sites offer autocomplete suggestions to their users users as they begin typing their search query. Suggestion help users find the exact words to describe what they are after. Often times they don’t know the exact product name or are not aware of the different variations of one product.

For example, when looking for a new iPhone 11, the user would discover autosuggestions such as “Pro” or Pro Max” which inform them about the available product options.

You can also offer some information about your products via autocomplete. You guide the users through your store and set an expectation of what exists. If a product gets suggested, the users will know its availability.



1.4. Need for the Proposed System

Autocomplete is a handful tool that you have likely stumbled upon in your daily internet usage. When you start typing into google, you can see how it offers you suggestions on what you might be looking for. This feature is called “autocomplete” or “autosuggest” and it’s a great way to guide, educate and promote content and products on your internal site search on your website.

When done well, autocomplete also works as a guide toward your content by providing search suggestions relevant to the site content. It can be a guiding hand for your users to find what they are looking for and offer them help in constructing their search. Autocomplete is especially powerful for mobile devices and for ecommerce marketplaces. No matter if your visitors use mobile or desktop, autocomplete is always a great feature because it saves your user’s time. On average, autocomplete reduces typing by about 25 percent.

Review of Literature

2.1. Summary of the investigation in the published papers

TF-IDF is an abbreviation for Term Frequency Inverse Document Frequency. This is very common algorithm to transform text into a meaningful representation of numbers which is used to fit machine algorithm for prediction. TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents. It has many uses, most importantly in automated text analysis, and is very useful for scoring words in machine learning algorithms for Natural Language Processing (NLP).

TFIDF

For a term i in document j:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

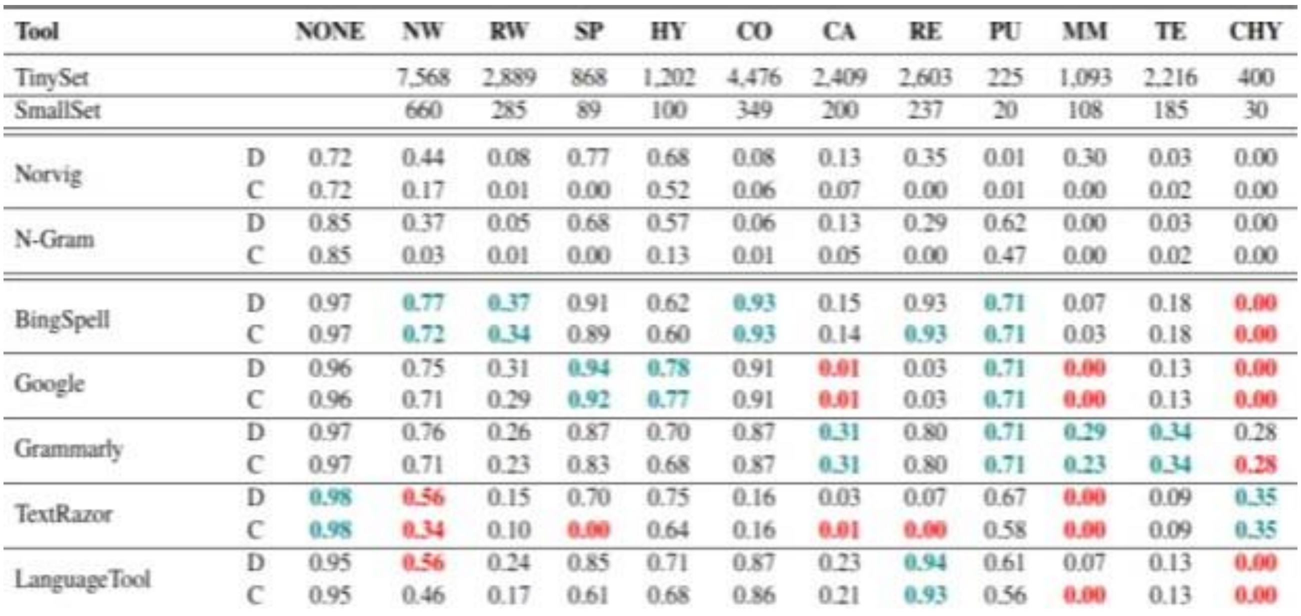
$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

fast. Consequently, it is logical that the best model attested was based on bidirectional LSTM (or Bi-LSTM) architecture; however, being first introduced in 2014, it has accumulated some notable disadvantages. For instance, the forward-propagation of the word makes it prone to being corrupted in the process, creating the problem of vanishing gradients. The other problem arisen was that we could not find any publicly available pre-trained embeddings for Bi-LSTM (although there are some for unidirectional LSTM networks), and here is where Transformer models, namely BERT, have their first advantage over the competition.

BERT is a neural network built by Google Research team, which main feature is combining the Transformer layers architecture and bidirectional text processing. It has shown to perform better on text tasks than the state-of-the-art models on the time of its release [GLUE]. It is equally important that BERT is a publicly available product, released under the Apache 2.0 license.

2.2. Comparison between the tools / methods / algorithms

We have listed few text -aided algorithms that have been used for identifying significant words from a corpus and used for text suggestion/text completion. Apart from all of these we have used TFIDF algorithm that goes sequentially with our domain of customer to user specific interaction system.



Tool		NONE	NW	RW	SP	HY	CO	CA	RE	PU	MM	TE	CHY
TinySet			7,568	2,889	868	1,202	4,476	2,409	2,603	225	1,093	2,216	400
SmallSet			660	285	89	100	349	200	237	20	108	185	30
Norvig	D	0.72	0.44	0.08	0.77	0.68	0.08	0.13	0.35	0.01	0.30	0.03	0.00
	C	0.72	0.17	0.01	0.00	0.52	0.06	0.07	0.00	0.01	0.00	0.02	0.00
N-Gram	D	0.85	0.37	0.05	0.68	0.57	0.06	0.13	0.29	0.62	0.00	0.03	0.00
	C	0.85	0.03	0.01	0.00	0.13	0.01	0.05	0.00	0.47	0.00	0.02	0.00
BingSpell	D	0.97	0.77	0.37	0.91	0.62	0.93	0.15	0.93	0.71	0.07	0.18	0.00
	C	0.97	0.72	0.34	0.89	0.60	0.93	0.14	0.93	0.71	0.03	0.18	0.00
Google	D	0.96	0.75	0.31	0.94	0.78	0.91	0.01	0.03	0.71	0.00	0.13	0.00
	C	0.96	0.71	0.29	0.92	0.77	0.91	0.01	0.03	0.71	0.00	0.13	0.00
Grammarly	D	0.97	0.76	0.26	0.87	0.70	0.87	0.31	0.80	0.71	0.29	0.34	0.28
	C	0.97	0.71	0.23	0.83	0.68	0.87	0.31	0.80	0.71	0.23	0.34	0.28
TextRazor	D	0.98	0.56	0.15	0.70	0.75	0.16	0.03	0.07	0.67	0.00	0.09	0.35
	C	0.98	0.34	0.10	0.00	0.64	0.16	0.01	0.00	0.58	0.00	0.09	0.35
LanguageTool	D	0.95	0.56	0.24	0.85	0.71	0.87	0.23	0.94	0.61	0.07	0.13	0.00
	C	0.95	0.46	0.17	0.61	0.68	0.86	0.21	0.93	0.56	0.00	0.13	0.00

Summary of the investigation in the Published papers

General purpose digital computers are widely used for a large variety of text-based applications, including word processing, e-mail, spreadsheets, personal calendars, etc. To use the computer for one of these purposes, a user typically types on a keyboard to enter text and commands into an active data file, which is open within an application program running on the computer. Other text input devices include a voice recognition interface, a touch-sensitive screen overlaid on top of a graphical image of a keyboard, or a system that detects the motion of a pen in combination with handwriting recognition software. The text and commands are then interpreted and manipulated by the application program in accordance with the syntax and functionality implemented by the application program. For many users, the most time-consuming computer activity is the entry of large amounts of text into various data files, such as word processing files and e-mail files. Regard- less of the input method used, the speed at which the text can be entered into the computer is a major factor governing the user's efficiency. The designers of text-intensive application programs have therefore developed text input aids to assist users in entering text into the computer.

A word prediction system is an example of such a text-input aid. Generally stated, a word prediction system predicts and suggests complete data entries based on partial data entries. This allows the user to type in a partial data entry and then accept a predicted word completion with a single keystroke, thus avoiding the keystrokes that would have been required to type the complete data entry. For example, a word prediction system may be configured to recognize a user's name so that the user's complete name, "Dean Hachamovitch" for instance, may be predicted after the user types the first few letters, "Dea" in this example. Creating word prediction systems that exhibit acceptable memory-use and performance characteristics, and that are not overly disruptive or annoying to the user, is an on-going challenge for software developers. Three techniques have traditionally been used to meet this challenge: organizing the user's document into structured fields; restricting the 50 data space used to predict word completions; and require- ing the user to request a word prediction when desired.

Analysis and Design

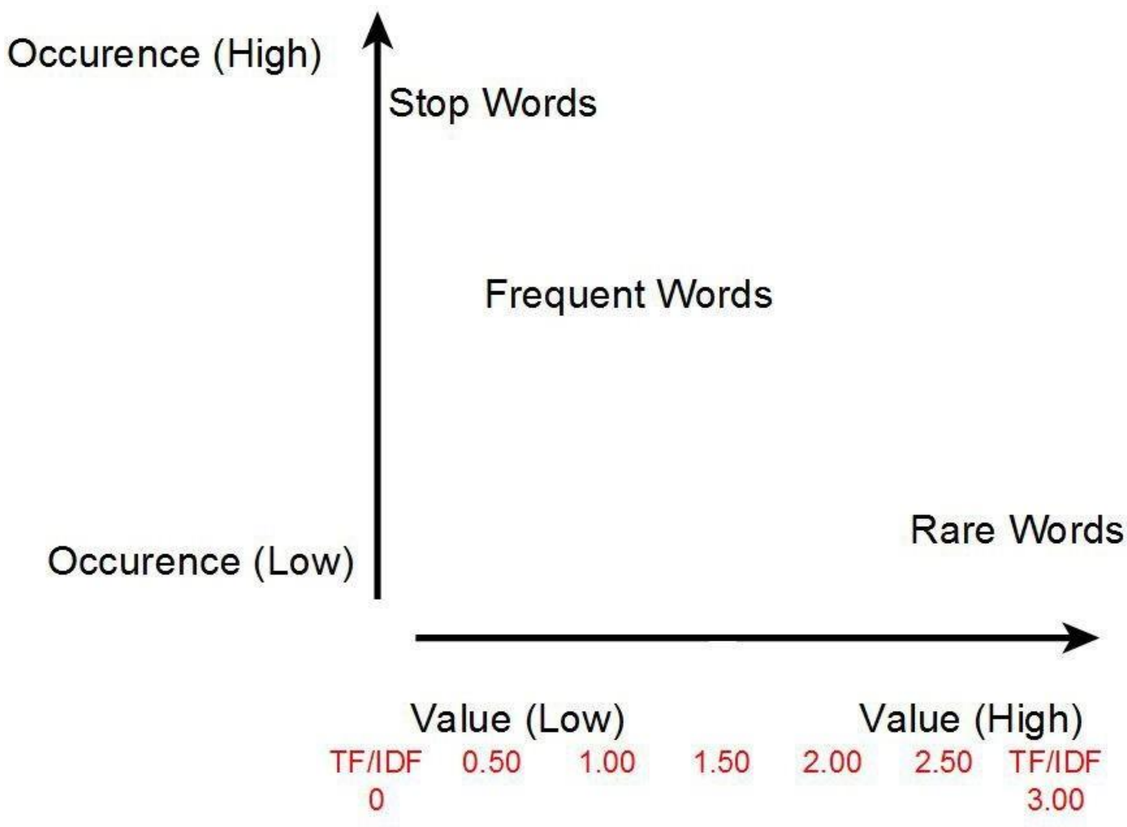
3.1. Procedure

In the jupyter notebook in this project, we select an history of sentences written by the representatives and the customer, format and correct them using a few regex rules and count them so we can estimate their frequency and likeliness to be useful again. After the calculation of a similarity matrix based on the sklearn tfidf tool (frequency and normalization of words), we use this matrix to calculate the similarity between the new few words written by the representative and the history of messages written in the past. The Autocomplete will recognize the closest sentences and rank 3 final proposals:

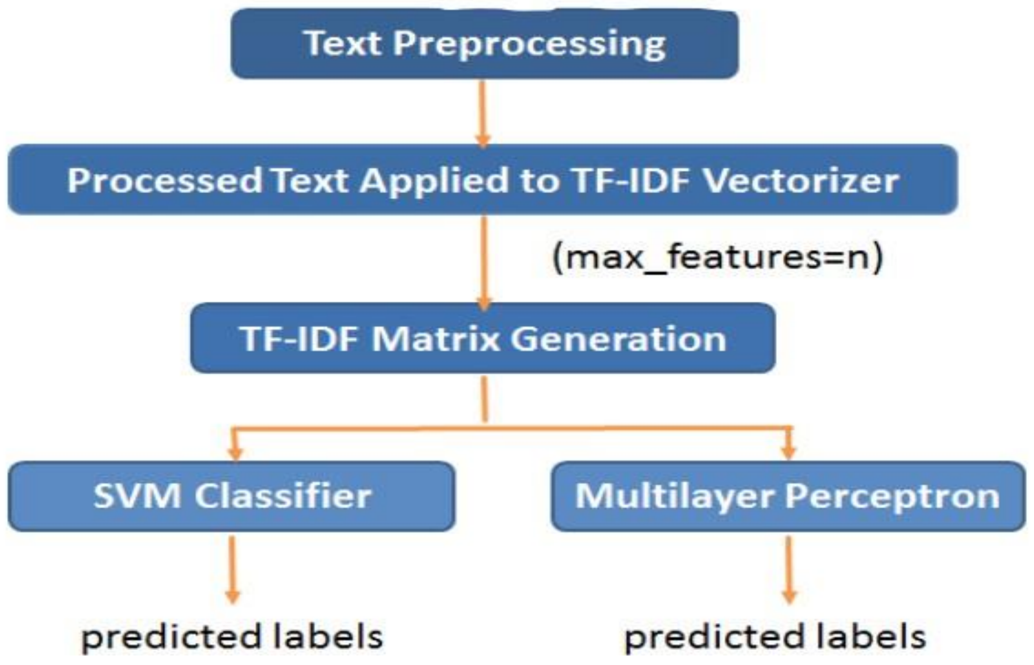
1: import all the important libraries
2: extract important pairs from sklearn to work with
3: load the data from json
4: process and clean the data using certain regex rules
5: define the tfidf model, feed all the data in it
6: use the tfidf vectorizer tool to work with the data
7: classify sentences using the feature weights
8: use a cosine similarity matrix to calculate and predict the next words using the feature weights
9: create a user interface to get user inputs
based on all of the above steps, The Autocomplete will recognize the closest sentences and rank 3 final proposals.

3.2. ANALYSIS

As mentioned before, TF-IDF is a numerical statistic which reflects on how important a word is to a document in the collection or corpus (Salton et al., 1988). The TF-IDF value increases proportionally to the number of times when a word appears in the document, but it is offset by the frequency of the word in the corpus, which helps to control the fact that some words are more common than others. The frequency term means the raw frequency of a term in a document. Moreover, the term regarding inverse document frequency is a measure of whether the term is common or rare across all documents in which can be obtained by dividing the total number of documents by the number of documents containing the term.


Occurence (High)
Stop Words
Frequent Words
Rare Words
Occurence (Low)
Value (Low)
Value (High)
TF/IDF 0
0.50
1.00
1.50
2.00
2.50
TF/IDF 3.00

3.3. System Architecture/Design



Description

1: First and foremost is to load all the required and important libraries to work with, importing libraries such as -The OS module in Python provides functions for interacting with the operating system. importing libraries such as Pandas/NumPy is mainly used for data analysis importing data from various allowing various data manipulation operations such as merging, reshaping, selecting, as well as data cleaning, and data wrangling features. The JSON module is mainly used to convert the python dictionary above into a JSON string that can be written into a file. While the JSON module will convert strings to Python datatypes, normally the JSON functions are used to read and write directly from JSON files.

2: importing all the important pairs to work with such as - mostly the feature extraction and metrics pairwise, we need these feature extraction techniques to convert text into a matrix and vector of features, and we will be using these pairs to work with our model

3: loading our corpus- Our corpus is all about history of sentences written by a customer and a representative in the past. The data is stored in a json file and using certain regex rules we will clean this data and convert this data that can be fed into our model.

4: process and clean our data using certain python functions and regex rules we split the data , split it on punctuations , split text to rows using certain inbuilt python functions such as split text or split text to rows and after processing the data we will count the no of sentences from the representative perspective , remove duplicates and use sentences which have at least 2 words .

5: after processing and cleaning our data, we define our model, and we are using the tfidf model - which stands for term frequency–inverse document frequency, is a numerical model that is intended to reflect how important a word is to a document in a collection or corpus. Typically, the tf-idf model is composed by two terms- Normalized Term Frequency (tf) - no of times term t occurs in a document/total no of terms in a document Inverse Document Frequency (idf) - log (total number of documents)/no of documents with term t in it after calculating both term frequency and inverse document frequency, we merge their results and higher the tfidf score more frequent and signific that word or sentence is to a document/corpus

6: now we will use a tool known as tfidf vectorizer which comes along with the tfidf model that has something also known as the weight feature which works along the tfidf

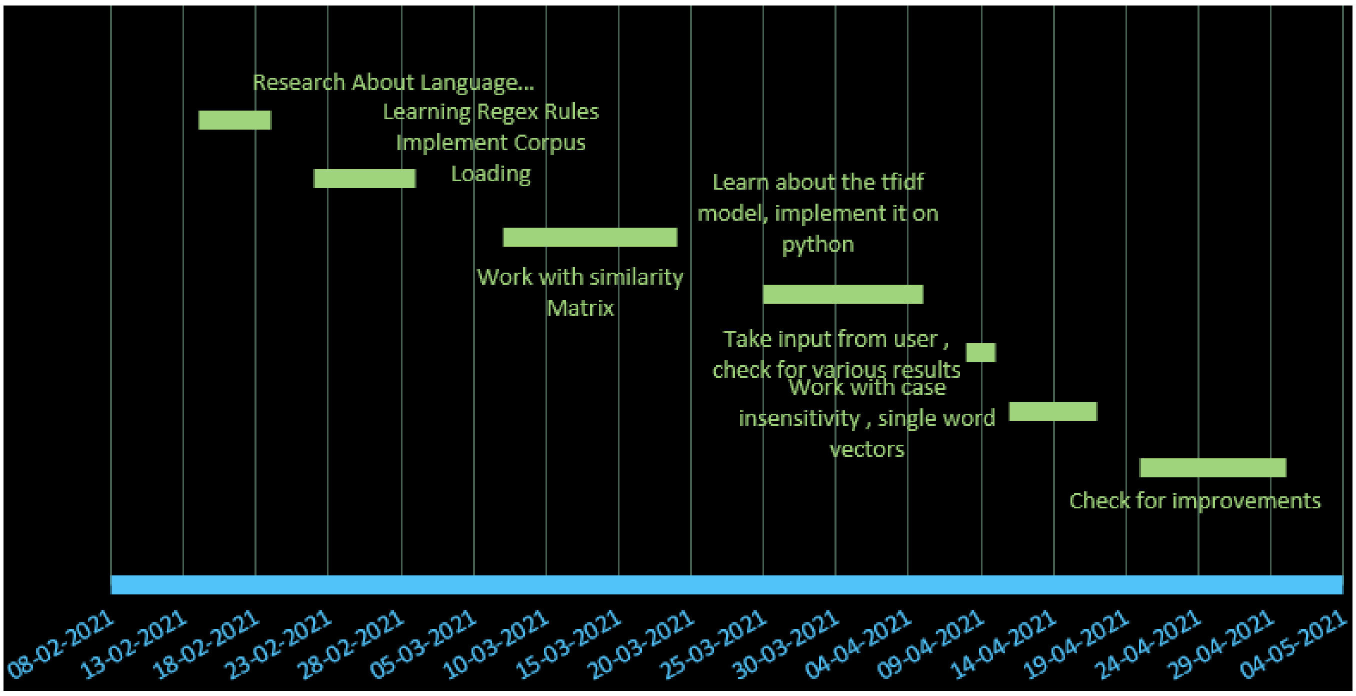
score and gives weight to the most important sentences. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. This weight feature is directionally proportional to the importance/significance of the sentence

7: Last but not the least we feed all of this to a matrix known as cosine similarity matrix that is a metric which is used to determine how similar two entities are irrespective of their size. It generally works with vector weights and After the calculation of a similarity matrix based on the sklearn tfidf tool (frequency and normalization of words), we use this matrix to calculate the similarity between the new few words written by the representative and the history of messages written in the past.

8: Finally, after applying all the 8 steps, we create a user interface to take multiple inputs from the user, the autocomplete is calculating the similarity between the sentence in the data and the prefix of the sentence written by the representative. As a weight feature, we chose to reorder using the frequency of the most common similar sentence. The Autocomplete will recognize the closest sentences and rank 3 final proposals based on the respective inputs.

Implementation

4.1. Implementation Plan



<u>Individual Contribution</u>

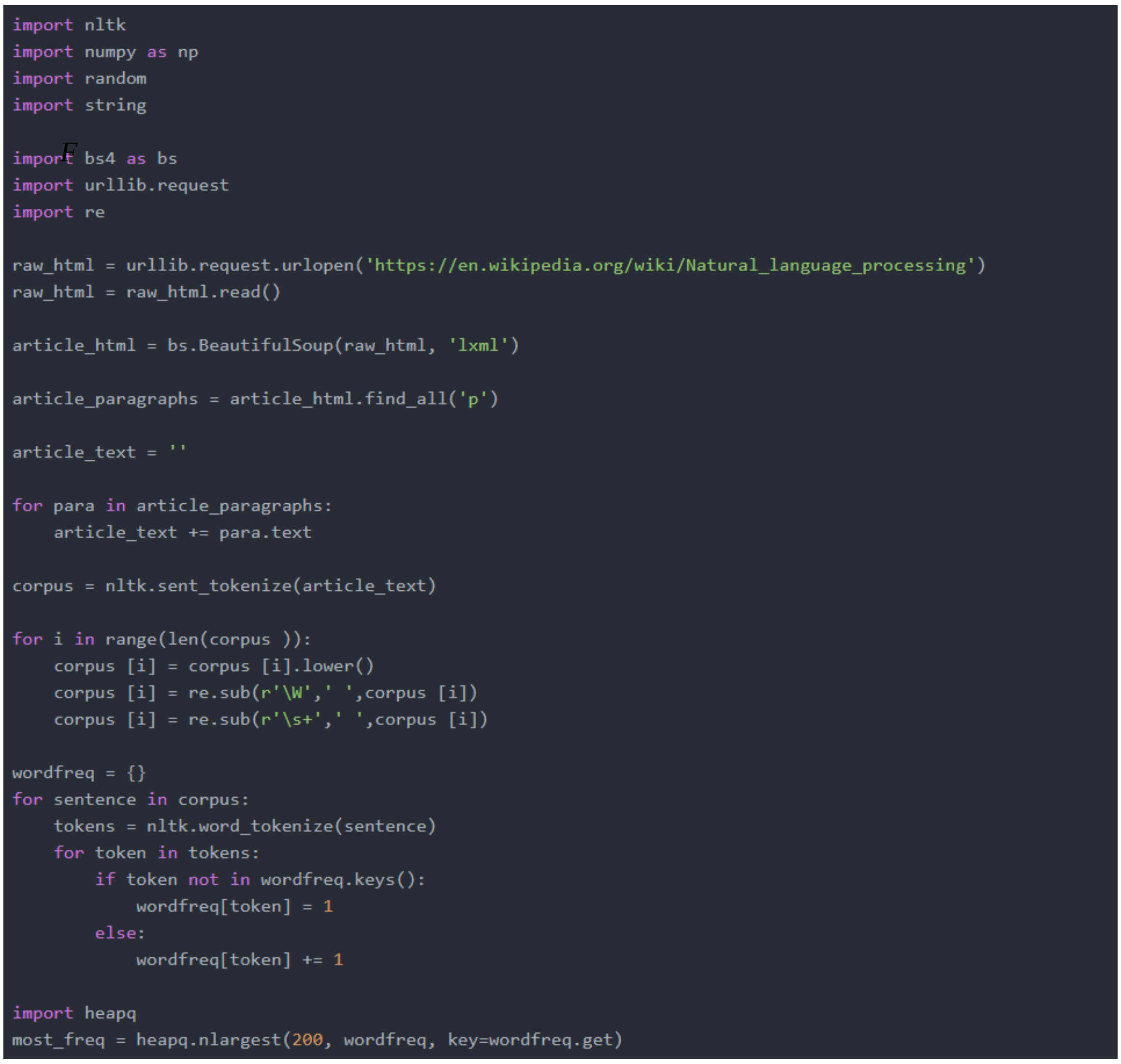
AHAD KHAN - 35% (Coding/System Implementation/Training the model)

PIYUSH SHENDE - 33% (Testing and Model suggestion)

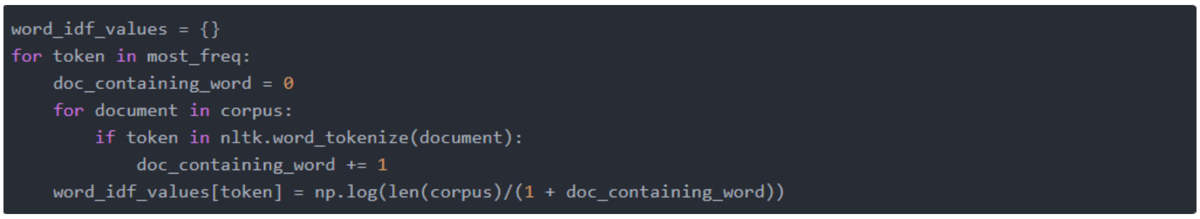
BINOY BANGERA - 32% (Research/error correction/Paperwork’s)

4.2. Coding Standard

<u>Tfidf Model</u>



```python
import nltk
import numpy as np
import random
import string

import bs4 as bs
import urllib.request
import re

raw_html = urllib.request.urlopen('https://en.wikipedia.org/wiki/Natural_language_processing')
raw_html = raw_html.read()

article_html = bs.BeautifulSoup(raw_html, 'lxml')

article_paragraphs = article_html.find_all('p')

article_text = ''

for para in article_paragraphs:
    article_text += para.text

corpus = nltk.sent_tokenize(article_text)

for i in range(len(corpus )):
    corpus [i] = corpus [i].lower()
    corpus [i] = re.sub(r'\W',' ',corpus [i])
    corpus [i] = re.sub(r'\s+',' ',corpus [i])

wordfreq = {}
for sentence in corpus:
    tokens = nltk.word_tokenize(sentence)
    for token in tokens:
        if token not in wordfreq.keys():
            wordfreq[token] = 1
        else:
            wordfreq[token] += 1

import heapq
most_freq = heapq.nlargest(200, wordfreq, key=wordfreq.get)
```

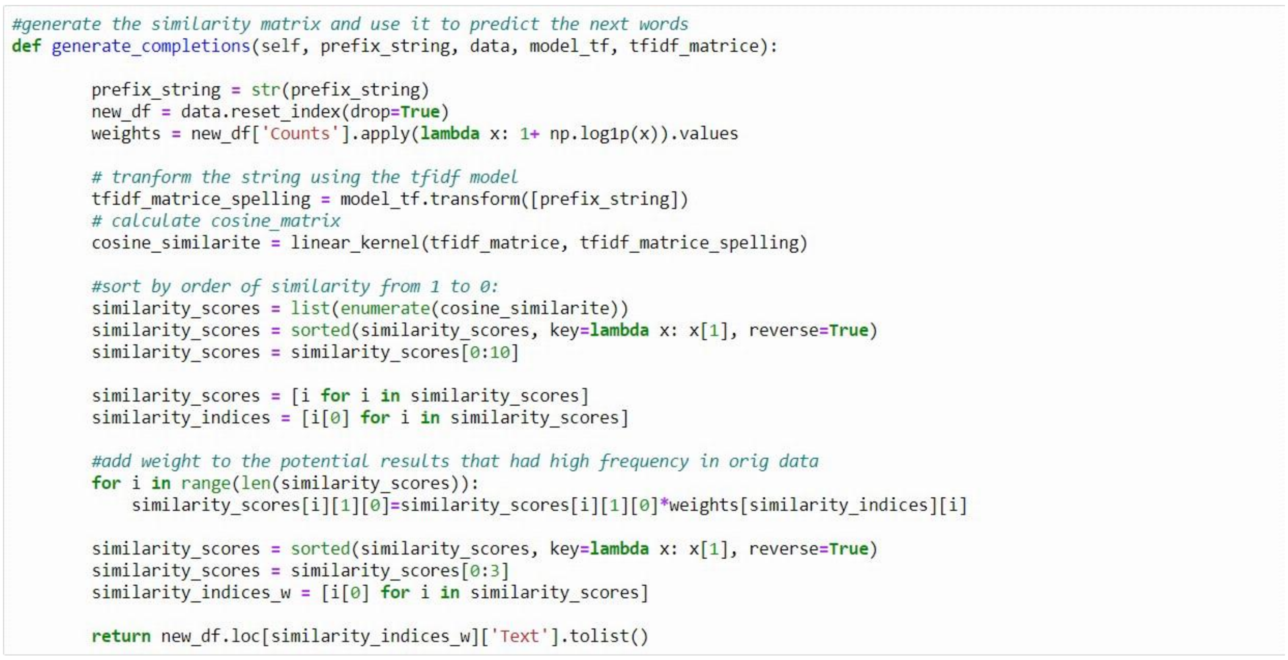


```python
word_idf_values = {}
for token in most_freq:
    doc_containing_word = 0
    for document in corpus:
        if token in nltk.word_tokenize(document):
            doc_containing_word += 1
    word_idf_values[token] = np.log(len(corpus)/(1 + doc_containing_word))
```

In the above script, we first scrape the Wikipedia article on Natural Language Processing. We then pre-process it to remove all the special characters and multiple empty spaces. Finally, we create a dictionary of word frequencies and then filter the top 200 most frequently occurring words. We create an empty dictionary word_idf_values. This dictionary will store most frequently occurring words as keys and their corresponding IDF values as dictionary values. Next, we iterate through the list of most frequently occurring words. During each iteration, we create a variable doc_containing_word. This variable will store the number of documents in which the word appears. Next, we iterate through all the sentences in our corpus. The sentence is tokenized and then we check if the word exists in the sentence or not, if the word exists, we increment the doc_containing_word variable. Finally, to calculate the IDF value we divide the total number of sentences by the total number of documents containing the word.

Tfidf Vectorizer

Scikit-learn's Tfidftransformer and Tfidfvectorizer aim to do the same thing, which is to convert a collection of raw documents to a matrix of TF-IDF features. Here, we can see clearly that Count Vectorizer give number of frequencies with respect to index of vocabulary whereas tf-idf consider overall documents of weight of words.

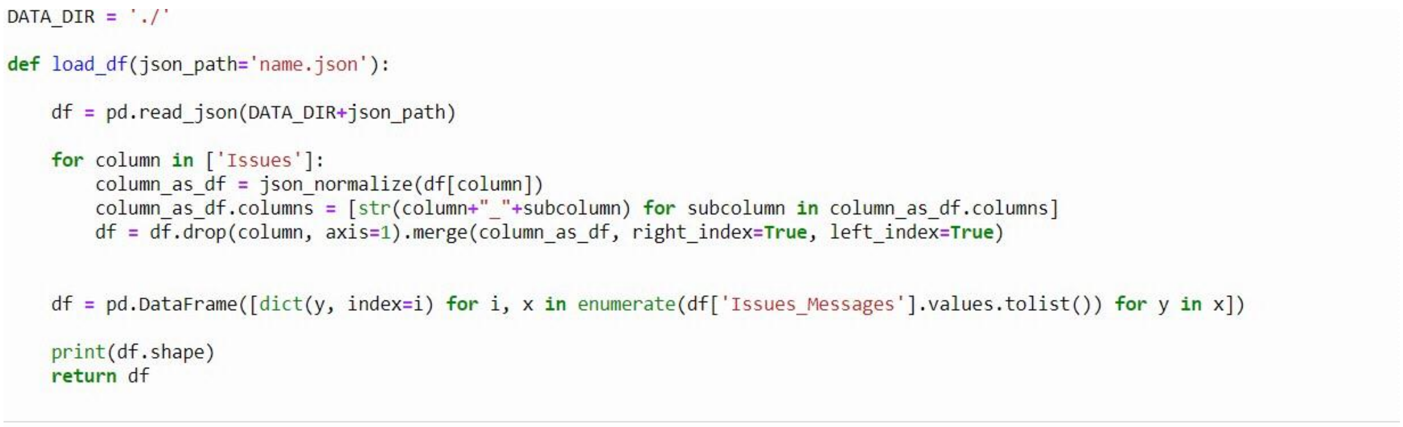


```python
#generate the similarity matrix and use it to predict the next words
def generate_completions(self, prefix_string, data, model_tf, tfidf_matrice):

        prefix_string = str(prefix_string)
        new_df = data.reset_index(drop=True)
        weights = new_df['Counts'].apply(lambda x: 1+ np.log1p(x)).values

        # tranform the string using the tfidf model
        tfidf_matrice_spelling = model_tf.transform([prefix_string])
        # calculate cosine_matrix
        cosine_similarite = linear_kernel(tfidf_matrice, tfidf_matrice_spelling)

        #sort by order of similarity from 1 to 0:
        similarity_scores = list(enumerate(cosine_similarite))
        similarity_scores = sorted(similarity_scores, key=lambda x: x[1], reverse=True)
        similarity_scores = similarity_scores[0:10]

        similarity_scores = [i for i in similarity_scores]
        similarity_indices = [i[0] for i in similarity_scores]

        #add weight to the potential results that had high frequency in orig data
        for i in range(len(similarity_scores)):
            similarity_scores[i][1][0]=similarity_scores[i][1][0]*weights[similarity_indices][i]

        similarity_scores = sorted(similarity_scores, key=lambda x: x[1], reverse=True)
        similarity_scores = similarity_scores[0:3]
        similarity_indices_w = [i[0] for i in similarity_scores]

        return new_df.loc[similarity_indices_w]['Text'].tolist()
```

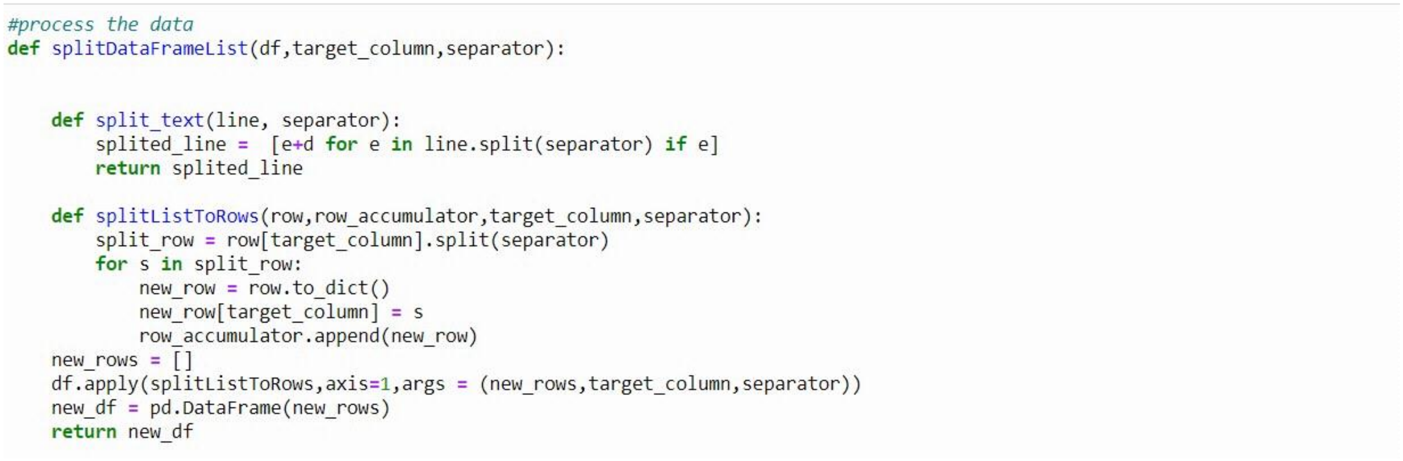
We can convert the test document set into a vector space where each term of vector is indexed as our index vocabulary. Example first term of the vector represents "blue" term of our vocabulary. the second term represents "sun" and so on. Now we are going to use term — frequency which means more than a measure of how many times the terms present in our vocabulary (E(t)). We can define the term-frequency as counting function:

Loading and Processing the Data

We have built this project on the jupyter notebook which is a free, open-source, interactive web tool to combine software and different parts of the code together, and it runs on python3 language. Our system revolves around the domain of Natural language processing your corpus is all about history of sentences written by a customer and a representative in the past. The data is stored in a json file and using certain regex rules we will clean this data and convert this data that can be fed into our model. Short for regular expression, a regex is a string of text that allows you to create patterns that help match, locate, and manage text..... Regular expressions can also be used from the command line and in text editors to find text within a file. Process and clean our data using certain python functions and regex rules we split the data, split it on punctuations, split text to rows using certain inbuilt python functions such as split text or split text to rows and after processing the data we will count the no of sentences from the representative perspective, remove duplicates and use sentences which have at least 2 words.



```
DATA_DIR = './'

def load_df(json_path='name.json'):

    df = pd.read_json(DATA_DIR+json_path)

    for column in ['Issues']:
        column_as_df = json_normalize(df[column])
        column_as_df.columns = [str(column+"_"+subcolumn) for subcolumn in column_as_df.columns]
        df = df.drop(column, axis=1).merge(column_as_df, right_index=True, left_index=True)


    df = pd.DataFrame([dict(y, index=i) for i, x in enumerate(df['Issues_Messages'].values.tolist()) for y in x])

    print(df.shape)
    return df
```



```
#process the data
def splitDataFrameList(df,target_column,separator):


    def split_text(line, separator):
        splited_line =  [e+d for e in line.split(separator) if e]
        return splited_line

    def splitListToRows(row,row_accumulator,target_column,separator):
        split_row = row[target_column].split(separator)
        for s in split_row:
            new_row = row.to_dict()
            new_row[target_column] = s
            row_accumulator.append(new_row)
    new_rows = []
    df.apply(splitListToRows,axis=1,args = (new_rows,target_column,separator))
    new_df = pd.DataFrame(new_rows)
    return new_df
```

Results and Discussion

We have built this project on the jupyter notebook which is a free, open-source, interactive web tool to combine software and different parts of the code together , and it runs on python3 language . In the jupyter notebook in this project, we select an history of sentences written by the representatives and the customer, format and correct them using a few regex rules and count them so we can estimate their frequency and likeliness to be useful again. After the calculation of a similarity matrix based on the sklearn tfidf tool (frequency and normalization of words), we use this matrix to calculate the similarity between the new few words written by the representative and the history of messages written in the past.

Expectations from the system:

1: To get accurate sentence suggestions.

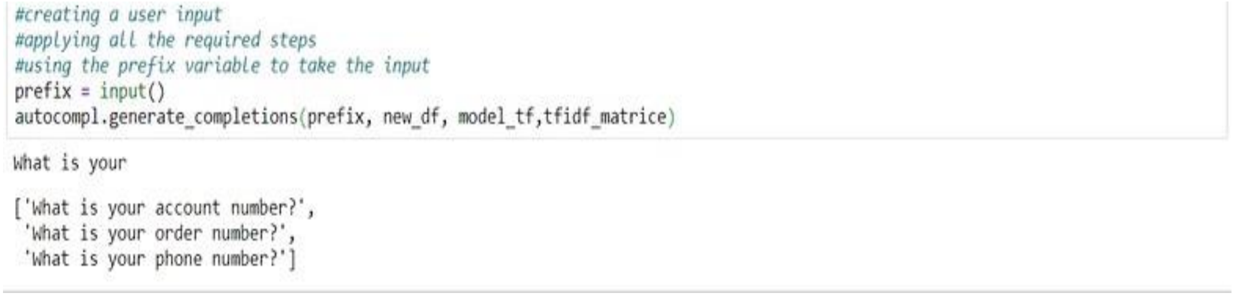
2: To predict grammatically correct and structured sentences.

3: To search for a particular response/question using a particular word.

4: To accept both uppercase and lowercase inputs.

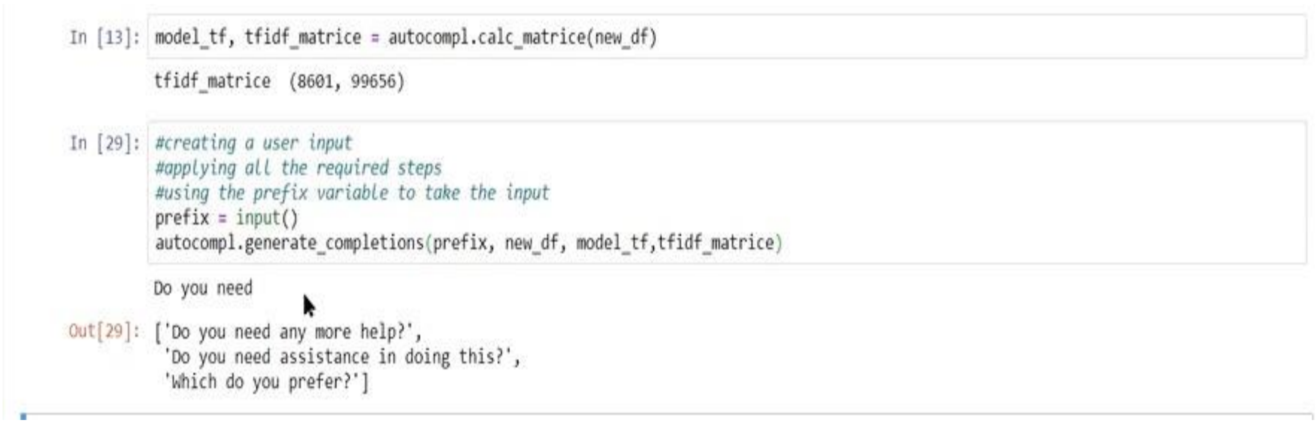
Result1:

If any customer wants to retrieve any information such as his account number/order number or any information from a service provider, so the representative can quickly use this system to interact with him by selecting domain specific questions which are suggested.



```
#creating a user input
#applying all the required steps
#using the prefix variable to take the input
prefix = input()
autocompl.generate_completions(prefix, new_df, model_tf,tfidf_matrice)
```

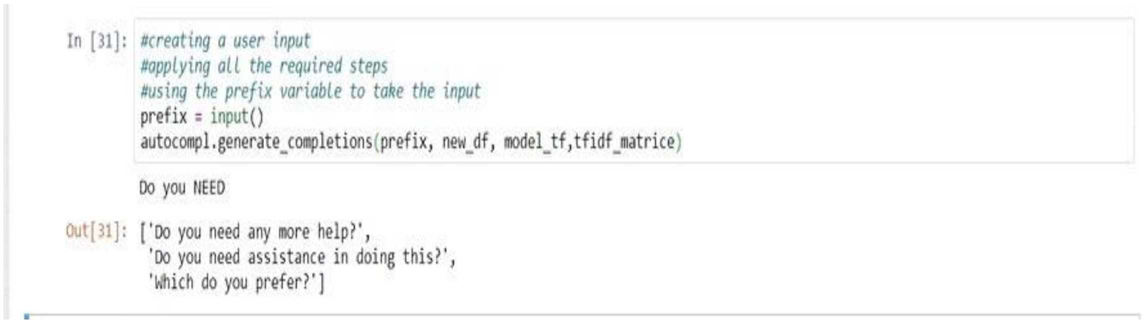
```
What is your

['What is your account number?',
 'What is your order number?',
 'What is your phone number?']
```



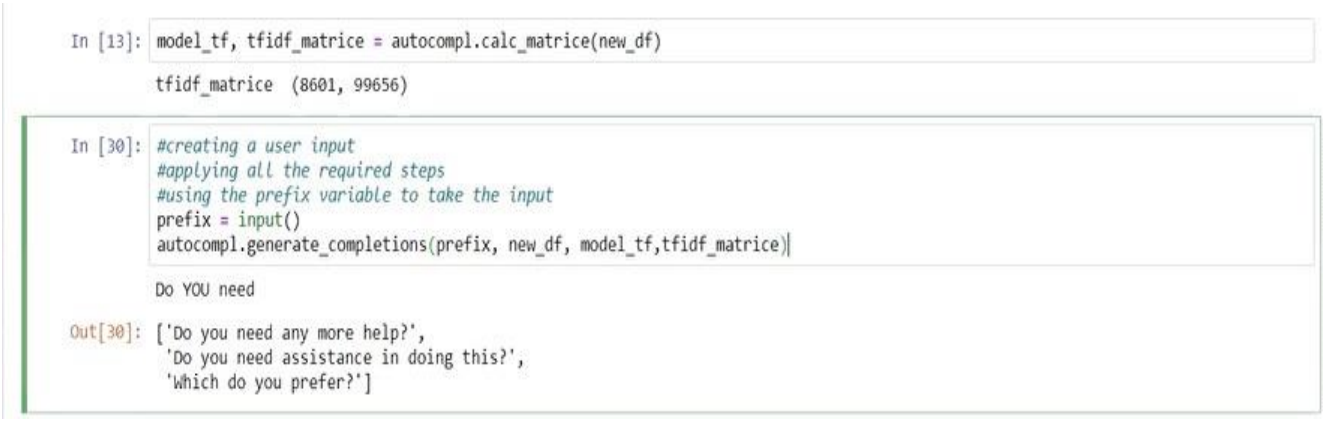
```
In [13]: model_tf, tfidf_matrice = autocompl.calc_matrice(new_df)

         tfidf_matrice  (8601, 99656)

In [29]: #creating a user input
         #applying all the required steps
         #using the prefix variable to take the input
         prefix = input()
         autocompl.generate_completions(prefix, new_df, model_tf,tfidf_matrice)

         Do you need

Out[29]: ['Do you need any more help?',
          'Do you need assistance in doing this?',
          'Which do you prefer?']
```

Result 2:

The system can give accurate suggestions irrespective of the case sensitivity of the system. In the first case "NEED" is all in capital and in the second case, "YOU" is all in capital. Irrespective of the case sensitivity of the input, the system gives the same output.



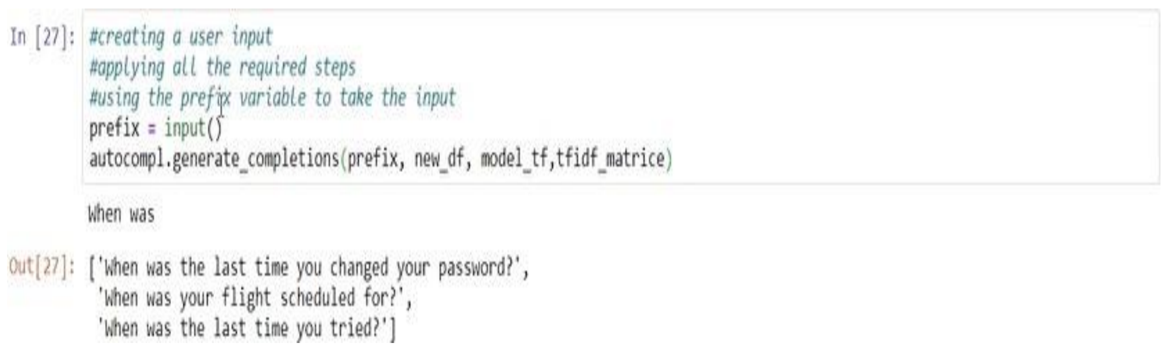
```
In [31]: #creating a user input
         #applying all the required steps
         #using the prefix variable to take the input
         prefix = input()
         autocompl.generate_completions(prefix, new_df, model_tf,tfidf_matrice)

         Do you NEED

Out[31]: ['Do you need any more help?',
          'Do you need assistance in doing this?',
          'Which do you prefer?']
```



```
In [13]: model_tf, tfidf_matrice = autocompl.calc_matrice(new_df)

         tfidf_matrice  (8601, 99656)

In [30]: #creating a user input
         #applying all the required steps
         #using the prefix variable to take the input
         prefix = input()
         autocompl.generate_completions(prefix, new_df, model_tf,tfidf_matrice)

         Do YOU need

Out[30]: ['Do you need any more help?',
          'Do you need assistance in doing this?',
          'Which do you prefer?']
```

Result 3:

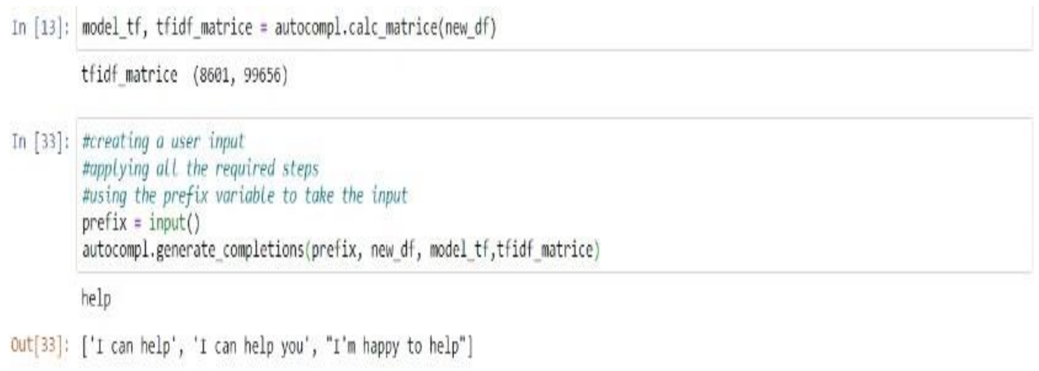
The system is also responsible to give grammatically correct and framed sentences. Instead of typing the whole and same questions again and again the user can use the feature of autocomplete to interact/respond efficiently and quickly to the customer's needs.



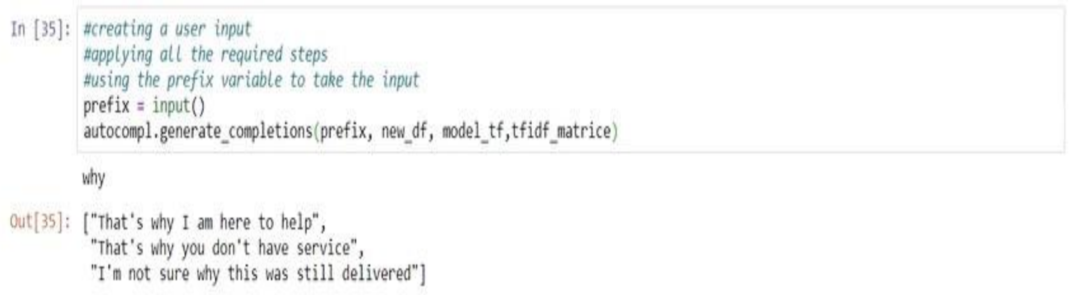
```
In [27]: #creating a user input
         #applying all the required steps
         #using the prefix variable to take the input
         prefix = input()
         autocompl.generate_completions(prefix, new_df, model_tf,tfidf_matrice)

         When was

Out[27]: ['When was the last time you changed your password?',
          'When was your flight scheduled for?',
          'When was the last time you tried?']
```

<u>Result 4:</u>

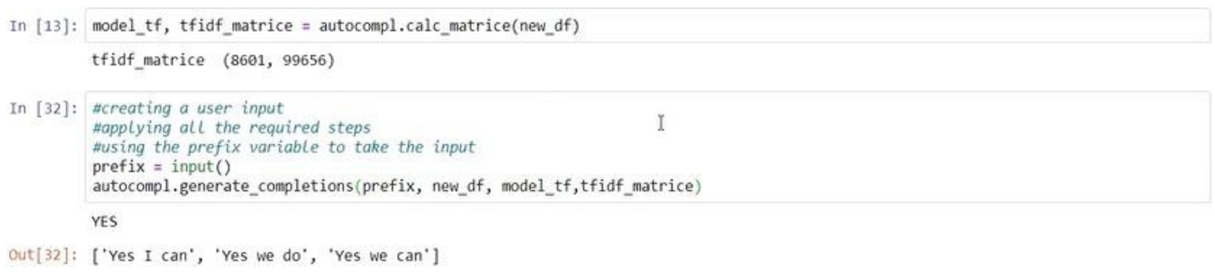
We can also use this system to find/search for sentences using only common words such as – help, need, please, what, why, when, yes, no and many more. Using these common keywords, it can frame accurate sentences which the representative can use it to interact with the user.



```
In [13]: model_tf, tfidf_matrice = autocompl.calc_matrice(new_df)

         tfidf_matrice  (8601, 99656)

In [33]: #creating a user input
         #applying all the required steps
         #using the prefix variable to take the input
         prefix = input()
         autocompl.generate_completions(prefix, new_df, model_tf,tfidf_matrice)

         help

Out[33]: ['I can help', 'I can help you', "I'm happy to help"]
```



```
In [35]: #creating a user input
         #applying all the required steps
         #using the prefix variable to take the input
         prefix = input()
         autocompl.generate_completions(prefix, new_df, model_tf,tfidf_matrice)

         why

Out[35]: ["That's why I am here to help",
          "That's why you don't have service",
          "I'm not sure why this was still delivered"]
```



```
In [13]: model_tf, tfidf_matrice = autocompl.calc_matrice(new_df)

         tfidf_matrice  (8601, 99656)

In [32]: #creating a user input
         #applying all the required steps
         #using the prefix variable to take the input
         prefix = input()
         autocompl.generate_completions(prefix, new_df, model_tf,tfidf_matrice)

         YES

Out[32]: ['Yes I can', 'Yes we do', 'Yes we can']
```

Conclusion & Future Work

Autocomplete, or word completion, is a feature in which an application predicts the rest of a word a user is typing .....Autocomplete speeds up human-computer Interactions when it correctly predicts the word a user intends to enter after only a few characters have been typed into a text input field. A word completion user interface allows the user to customize each suggestion list with user- defined name- completion pairs on an on-going basis. This work proposes a Autocomplete word completion system detection method using NLP. This method of autocomplete of text using ML can be very useful to the employees/representatives to have a interaction with different types of users as well as save their time. The system automatically predicts your best context based on certain training of your neural network

Some of the improvements we can from our system- improvements:

Clean up the “Mr. Smith” and “Ms. Smith” in the dataset Match the letter to the words (spelling match) and the match the words to the representatives’ sentences history.

Build an evaluation of the results:

a. Offline: using unseen conversations between the representative and the customer, input the prefix of the representative in the model and match to see if the actual representative sentences are part of the 3 ranked proposals.

b. Online: count the number of times the representative actually selects a proposal and the count the number of times the representative decides to ignore them. Improve the system by first matching the customer sentences to a topic id context in order to better predict the representative answers.

References

1. MahdiehPootchi, George Thomas, ‘Nlp analysis and machine learning for detecting patterns [Online] Available: https://www.sciencedirect.com/science/article/pii/S193152441 730333X [Accessed 17 May 2020].

2. Dipanjan (Dj) Sarkar, ‘NLP with Deep Learning’, 2019. [Online] Available: https://towardsdatascience.com/detecting-patterns-with- deep- learning-9e45c1e34 Poostchi, M., Silamut, K., Maude, R., Jaeger, S. and Thoma, G., 2020. ‘Text Analysis and Machine Learning.’ [Online] Available at: https://www.Ncbi.nlm.nih.gov/pmc/articles/PMC5840030/ [Accessed 17 May 2020].

3. P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100,000+ questions for machine comprehension of text. EMNLP, 2016.

4. P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100,000+ questions for machine comprehension of text. EMNLP, 2016.

5. P. Ramachandran, P. J. Liu, and Q. V. Le. Unsupervised pretraining for sequence-to-sequence learning. ArXiv preprint arXiv: 1611.02683, 2016.

6. M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun. Efficient learning of sparse representations with an energy-based model. In Advances in neural information processing systems, pages 1137– 1144, 2007.X. Zhu. Semi-supervised learning literature survey. 2005.

APPENDIX-1

Software/System requirements

• **Os**
The OS module in python provides functions for interacting with the operating system. OS, comes under Python's standard utility modules. This module provides a portable way of using operating system dependent functionality. The *os* and *os. path* modules include many functions to interact with the file system. Following are some functions in OS module:
osname: This function gives the name of the operating system dependent module imported

• **NumPy**
NumPy is a python library used for working with arrays.
NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. Moreover, NumPy forms the foundation of the Machine Learning stack.
In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

• **Scikit learn**
Scikit-learn (formerly scikits. learn and also known as sklearn) is a free software machine learning library for the Python programming language.[3] It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. Scikit-learn is largely written in Python, and uses NumPy extensively for high-performance linear algebra and array operations. Furthermore, some core algorithms are written in Cython to improve performance. Support vector machines are implemented by a Cython wrapper around LIBSVM; logistic regression and linear support vector machines by a similar wrapper around LIBLINEAR. In such cases, extending these methods with Python may not be possible. Scikit-learn integrates well with many other Python libraries, such as Matplotlib and plotly for plotting, NumPy for array vectorization, Panda's data frames, SciPy, and many more.

• **Pandas**
Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

• **Metrics**
The sklearn. metrics module implements several losses, score, and utility functions to measure classification performance. Some metrics might require probability estimates of the positive class, confidence values, or binary decisions values.

Hardware/Software Requirements

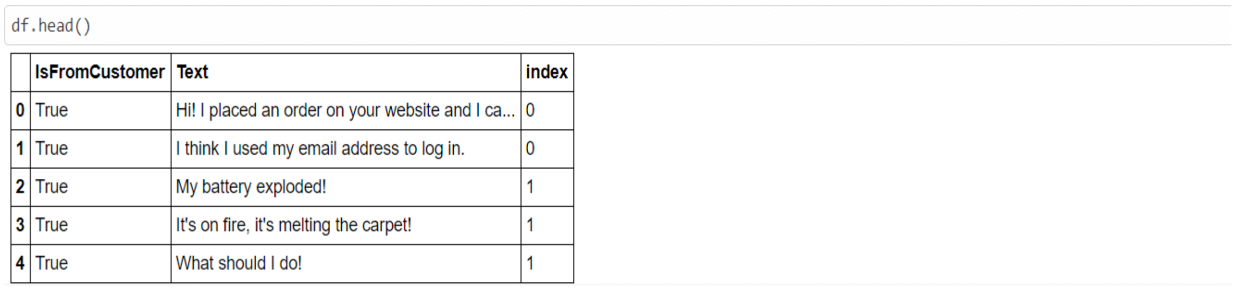
• Python
• Csv Reader
• NumPy/Pandas
NumPy provides the essential multi-dimensional array-oriented computing functionalities designed for high-level mathematical functions and scientific computation. Pandas: Pandas is one of the most widely used python libraries in data science. It provides high-performance, easy to use structures and data analysis tools.

• **Sci-kit Learn**
Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

• Kaggle
Kaggle allows users to find and publish data sets, explore and build models in a web-based data.

Datasets
Our corpus is all about history of sentences written by a customer and a representative in the past. The data is stored in a json file and using certain regex rules we will clean this data and convert this data that can be fed into our model. The data contains information about day to day if interaction between a customer in the form of a common thread. The file contains 22K conversations between a customer and a representative. For the purpose of this project, we are only interested in completing the threads of the representative. The data is going to separate the threads from the customer and the representative, separate the sentences based on the punctuation (we will keep the punctuation), the final text will be cleaned up with some light regex and only the sentence larger than 1 word will be kept.

Finally, since the representative has the tendency to ask the same question over and over again, the autocomplete is extremely useful by suggesting a complete sentence. In our case, we will count the number of occurrences of the same sentence so we can use it as a feature later on and delete the duplicates.



```
df.head()
```

	IsFromCustomer	Text	index
0	True	Hi! I placed an order on your website and I ca...	0
1	True	I think I used my email address to log in.	0
2	True	My battery exploded!	1
3	True	It's on fire, it's melting the carpet!	1
4	True	What should I do!	1

Acknowledgements

We place on record our heartfelt gratitude to our guide Prof. Uday Nayak and Prof. Janhavi Baikerikar, Head of Department of Information Technology Engineering, DBIT, Kurla, for all the mentoring and help extended to us in the successful completion of the project. His inspiration and encouragement all through the project work is remarkable and his thought-provoking support through different critical stages of the project is highly commendable. We also thank all the staff members of our department who have been supportive and extended their timely help for the completion of this project.

Autocomplete, or word completion, is a feature in which an application predicts the rest of a word a user is typing....... Autocomplete speeds up human computer interactions when it correctly predicts the words a user intends to enter after only a few characters have been typed into a text input field. A word completion user interface allows the user to customize each suggestions list with user- defined name- completion pairs on an on- going basis. This work proposes a autocomplete word completion system detection method using NLP. This method of autocomplete of text using ML can be very useful to the employees/ representatives to have a better text completion system.

Name	**Signature**
AHAD KHAN	________________
PIYUSH SHENDE	________________
BINOY BANGERA	________________

Date: 21-05-2021

SmallSEOTools

PLAGIARISM SCAN REPORT

Words	334	Date	May 20,2021
Characters	2112	Excluded URL	

0% Plagiarism	100% Unique	0 Plagiarized Sentences	15 Unique Sentences

Content Checked For Plagiarism

We have built this project on the jupyter notebook which is a free, open-source, interactive web tool to combine software and different parts of the code together , and it runs on python3 language . In the jupyter notebook in this project, we select an history of sentences written by the representatives and the customer, format and correct them using a few regex rules and count them so we can estimate their frequency and likeliness to be useful again. After the calculation of a similarity matrix based on the sklearn tfidf tool (frequency and normalization of words), we use this matrix to calculate the similarity between the new few words written by the representative and the history of messages written in the past.Expectations from the system:
1: To get accurate sentence suggestions.
2: To predict grammatically correct and structured sentences.
3: To search for a particular response/question using a particular word.
4: To accept both uppercase and lowercase inputs.
Result1:
If any customer wants to retrieve any information such as his account number/order number or any information from a service provider, so the representative can quickly use this system to interact with him by selecting domain specific questions which are suggested.The system can give accurate suggestions irrespective of the case sensitivity of the system. In the first case "NEED" is all in capital and in the second case,
"YOU" is all in capital. Irrespective of the case sensitivity of the input, the system gives the same output.
The system is also responsible to give grammatically correct and framed sentences. Instead of typing the whole and same questions again and again the user can use the
feature of autocomplete to interact/respond efficiently and quickly to the customer's needs.We can also use this system to find/search for sentences using only common words such as – help, need, please, what, why, when, yes, no and many more. Using these common keywords, it can frame accurate sentences which the representative can use it to interact with the user.

Sources	Similarity