NATURAL LANGUAGE PROCESSING

Team Members:

Mansi Saxena (18BCE0307)
Raksha Kannu (18BCE0341)
Yogeswari Sahu (18BCE0928)
Samuel Sujith (18BCE0344)

Report submitted for
the Final Review of the Project

Information retrieval for COVID19 for Common End Users

Course Code: CSE4022 – NLP
Slot: E2 + TE2

Professor: Rajeshkannan R

Deadline: November 2020

INDEX

1. ABSTRACT

The task of Question-Answering has always been a challenging task for any machine. This is because it requires the ability to understand and comprehend natural human language, along with the knowledge about the world we live in. This is a major problem in the field of research on natural language processing and artificial intelligence. Question-Answering systems are a suitable benchmark by which the ability of machines to understand complex human reasoning can be measured with decent accuracy. Many Question-Answering systems largely consist of text retrieval models made in the past. In the current world, deep-learning sequential models such as Recurrent Neural Networks are used to implement the Question-Answering systems, thus broadening and deepening their cases. Some examples of these models are Bag of Words (BOW), Word to Vector (Word2Vec), GloVe, TF-IDF, and BERT embeddings. These models use vector representation of words to understand the relationship between various words and how they are placed together. These models have shown good performance and accuracy, having good natural language understanding ability. Thus, they can easily be applied to other Natural Language Processing problems using Transfer Learning. The idea that we have implemented in our project is to build a Question-Answer Retrieval system for the common end-users on COVID-19.

2. INTRODUCTION

Information retrieval is the activity of obtaining data that is relevant to the information provided from a collection of those resources. Searches can be based on full-text or other content-based indexing. Information retrieval is the science of searching for information in a document, searching for documents themselves, and also searching for the metadata that describes data, and for databases of texts.

Due to this pandemic, there is panic all over the world, there has been a desperate need for correct information and updates for common people. Most of the data is already available on the internet on various websites in the FAQ format. The main target is to get this data from different sources into a single searchable format. To be able to make this process easier for the users, the information retrieval method can help search over the large chunks of raw data imported from various sources. The data collected from the sources was organized manually into a supported file and pre-processed.

To be able to achieve this, we are using various NLP models like Bag of Words (BOW), Word to Vector (Word2Vec), GloVe, Term Frequency -Inverse Document Frequency(TF-IDF), and BERT to build an effective solution that can be made viable for common users. With the combination of these models, innovative bidirectional text sequence search words can be possible and can deliver results exceptionally quickly. At first, all the words are added to the dictionary and each word is allotted a number, when a user inputs a question the results are filtered based on the most number of matching words in the organized data. But this won't help you get the best answer from the data, here word2vec and GloVe models come into play to filter the better results out of the earlier ones. To get the best result out of this word embedding

comes in the final stage. We use BERT (Bidirectional Encoder Representations from Transformers) Model as a result to get better accuracy. Bert's key technical innovation is applying the bidirectional training of Transformer, a popular attention model, to language modelling. This is in contrast to previous efforts that looked at a text sequence either from left to right or combined left-to-right and right-to-left training.

Various algorithms are used in this process to make the results more accurate and to efficiently retrieve the best output. This model turns out to be an effective way to help the common man get the right answers for their queries and this helps people not make false assumptions but to get the right answers quickly from reliable sources.

3. PROBLEM STATEMENT

EXPLANATION:

The necessity of a Question-Answering system has become prevalent due to people's increasing queries and a major factor to be taken into consideration is that by time, these queries are going on increasing. In this situation, the approach is to get the most accurate answers by exploring various models. Question Answering (QA) system is taking an important role in the current search engine optimization concept. The natural language processing technique is mostly implemented in the QA system for asking the user's questions and several steps are also followed for the conversion of questions to query form for getting an exact answer. In order to get the highest accuracies, we need to explore and implement various models like Bag of Words, Word2Vec, Glove, TF-IDF, and BERT.

3.1 ARCHITECTURE DIAGRAM

Bag of Words

In this model, all words in the input text or sequence (except proper nouns) are converted to lowercase and all non-alphanumeric keywords, punctuations and stopwords are removed. Then, a list of unique words is obtained from this cleaned and processed input text. The frequency of occurrence of each of these words in the cleaned input text is calculated as a histogram. This list of frequencies can be stored as a dictionary and sorted in descending order. Now, each unique word is made an input feature. The frequency of each word in each sentence in the cleaned input text is calculated as vectors. These vectors are the independent features of the model from which the output dependent feature is calculated.

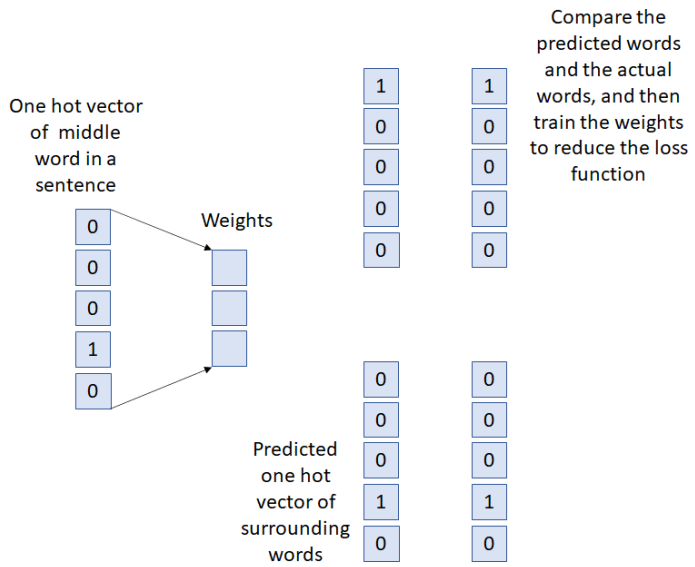
TF-IDF

This model is called "Term Frequency - Inverse Document Frequency". In this model, the Term Frequencies (or, TF) is calculated for each word. TF is defined as the ratio between the numbers of repetitions of a word in a sentence to the total number of words in a sentence. Then Inverse Document Frequency (or, IDF) is calculated for each word. IDF is defined as the

logarithm of the ratio of number of sentences to the number of sentences that contain the particular word. Now, the TF and IDF values for each word in each sentence are multiplied. This gives us the vectors for each word with respect to each sentence, which are treated as the output vectors, or our dependent feature.

Word2vec

In this model, input text is first cleaned as explained above in the BOW Model. Then word embeddings of the words in this cleaned text is generated using a simple 2 layer Neural Network. This embedding preserves the relationship between two words and it can be done using Continuous Bag of Words algorithm (CBOW) or Skip Gram Model. A sliding window of size 3, or any other length, is used to analyse each word. In CBOW, the middle word in this window is predicted at each iteration depending on the surrounding words. Then, the difference in the actual center word and the predicted center word is computed as the loss function, and the neural network is trained by minimising this loss function.

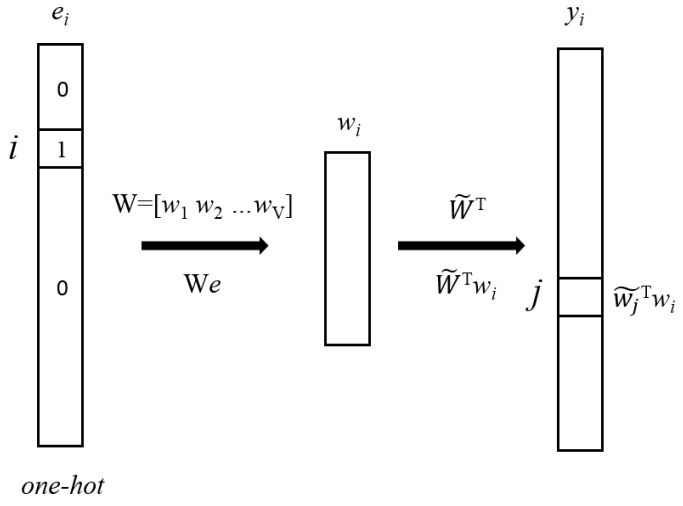


Skip Gram

In the Skip Gram Model, the surrounding words in the window are predicted based on the center word. Again, the difference in the actual words and the predicted words is computed as the loss function, used later for training the weights. This matrix of trained weights is then multiplied with the one-hot encoding of the words with respect to the dictionary to give the word embeddings. In our project, we have used the Skip Gram model.

GloVe Model

GloVe is an alternate approach to build word embeddings using matrix factorization techniques on the word-word co-occurrence matrix. In this model, a Count Matrix is constructed which has dimensions n*n (where n is length of the dictionary). This matrix contains values **Xij**, which is the number of times word j appears in the surrounding (or context) of word i. It is a sparse, symmetric matrix. Due to long-tail distribution, the non-zero values will be very large. Thus, logarithm scale is used, or **Log Xij** (After adding 1 to every value before taking log to



avoid NaN values)**.** Next, every (i,j)th entry is weighted. A function is computed:

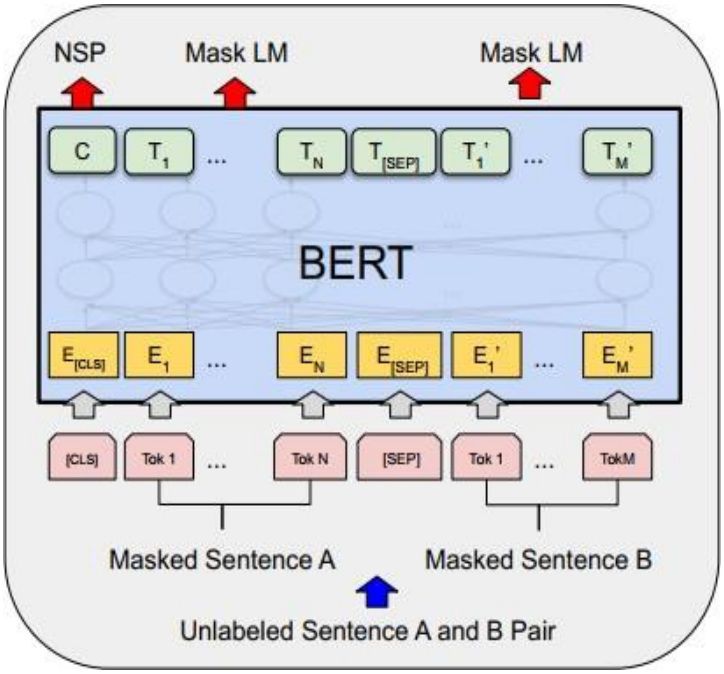
f(X)=(X/Xmax)^alpha if X < Xmax else 1

Cost function is computed by this value, using which the weights of the neural network are trained.

BERT

Transformer is an NLP model that performs better than the popularly known LSTM model and is also faster. It has two main components - the Encoder and the Decoder. The BERT model is essentially a series of stacked up encoders.

It is trained in 2 stages - Pre-Training, and Fine Tuning. Pretraining helps the model learn and understand the language and the context of the input sequence. It trains on two unsupervised tasks simultaneously - Masked Language Model(MLM) and Next Sentence Prediction (NSP). In MLM, sentences are given to BERT with masks on some words, and BERT learns to predict the missing words, similar to filling in the blanks. In NSP, two sentences are given as input and BERT learns to predict if the second sentence follows the first, like a binary classification problem.



In practice, Pre-Training on both the problems is done simultaneously. Two masked sentences A and B are given as input as shown in the figure above. Some words in these sentences are hidden, or "masked". Each word is converted to tokens, which is then converted to Embeddings. This embedding is essentially the summation of three vectors -

1. Token Embedding (or the Pre-Trained embeddings)
2. Segment Embedding (or the sentence number encoded into a vector)
3. Position Embeddings (or the position of a word within a sentence that is encoded into a vector)

This is done to understand the temporal ordering of the sentences, as they are all processed simultaneously.

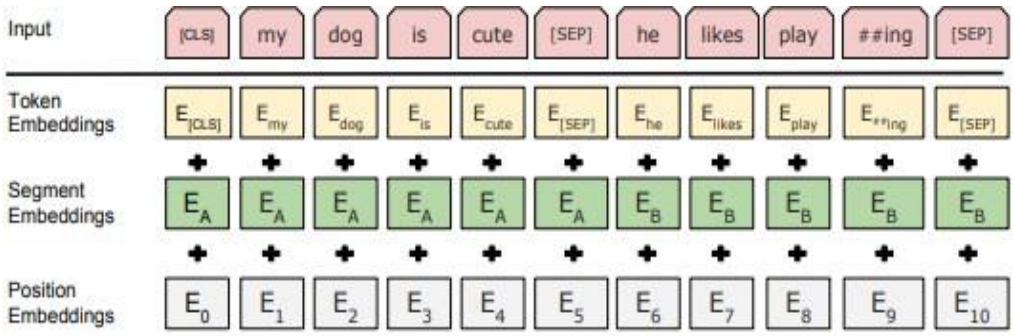
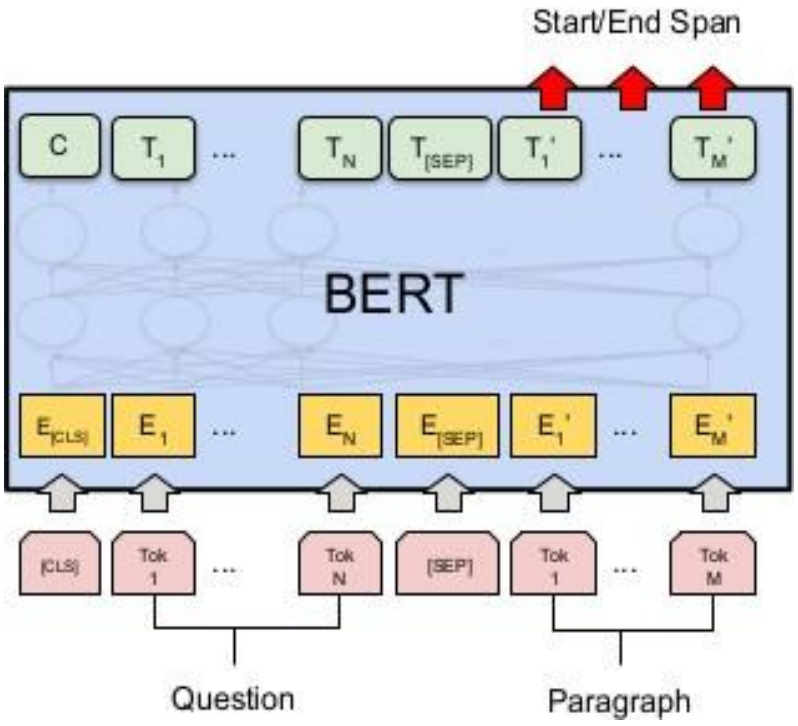


Figure 2: BERT input representation. The input embeddings is the sum of the token embeddings, the segmentation embeddings and the position embeddings.

After the embeddings are generated, they are given as input to BERT, or the stacked up Encoders. The output word embeddings are generated simultaneously for each word. The output c holds value 1 if sentence B follows sentence A, else 0. These output embeddings are one by one given as input to a fully connected, dense neural network layer with its number of units equal to the number of words in our vocabulary. A softmax activation layer is applied after this to get a probability distribution of which word comes next in our output sequence. The loss function is defined as the difference between this probability distribution given by the softmax layer, and the actual one-hot encoding of the next word. It only considers the masked words, ignoring the other output words. This loss is termed as "Cross-Entropy Loss".

Thus, this training will generate optimized parameters for the initial layers of the model. We have downloaded these parameters from the official documentation website of BERT.



The second stage is Fine-tuning BERT for a specific task. For our problem, we will Fine Tune BERT to the Question and Answer problem. The fully connected output layer is replaced with a fresh set of output layers. Now, Question Answering can be performed using a dataset similar to a supervised learning problem. Sentence A and B that are given as

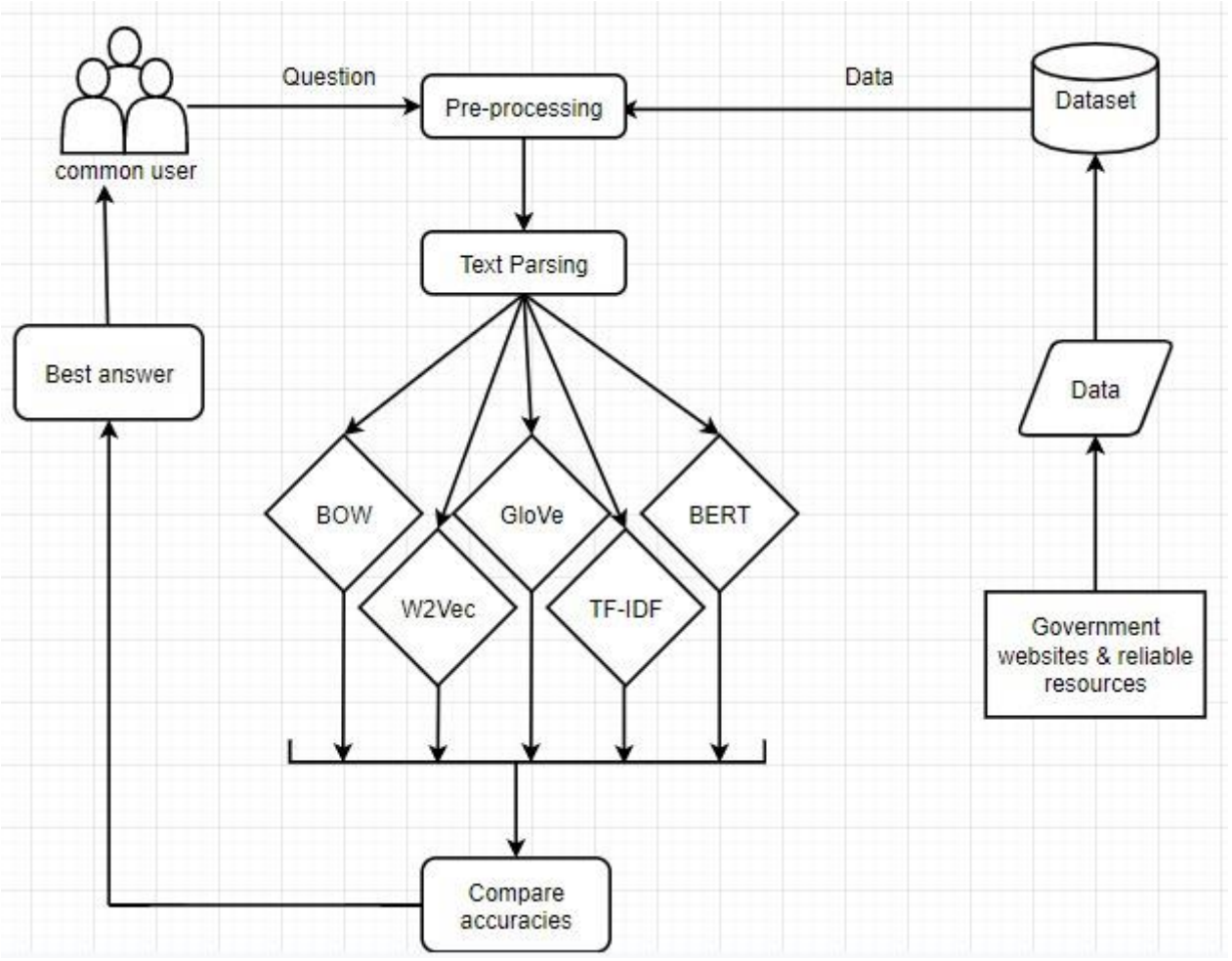
input during Pre-Training are now the questions and answers from our dataset.

These sentences are trained in the same way as in the Pre-Training phase. The output has the start and ends words that encapsulate the answer. Here, only the parameters in the last few layers are trained, while the parameters of the earlier layers are only fine- tuned to better fit our purpose. Thus, this training occurs faster. This technique is also referred to as Transfer Learning.

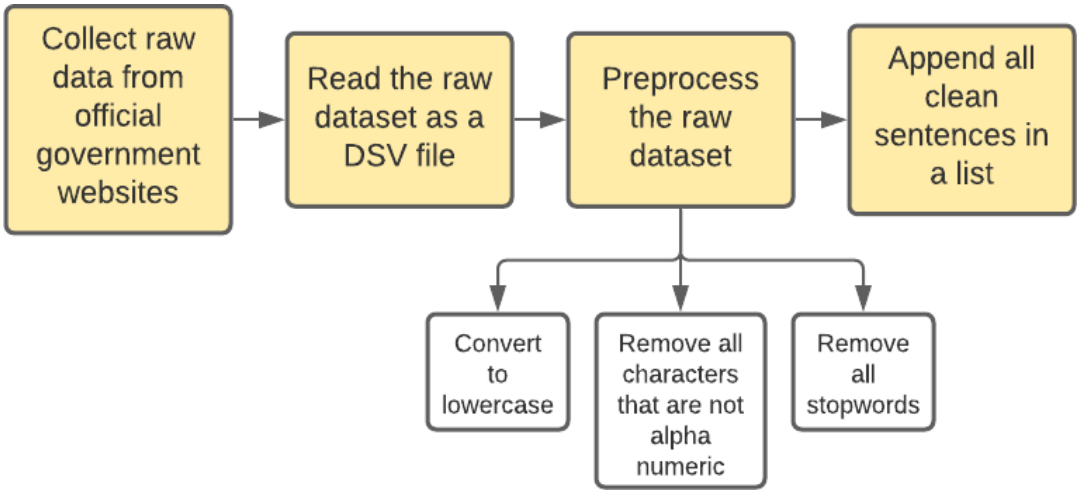
Cosine Similarity is calculated in this way:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

3.2 FLOW DIAGRAM



PREPROCESSING



3.3 PSEUDOCODE

```
START
Import Libraries
Import Dataset
Add header names to dataframe
Drop duplicates
FUNCTION clean_sentence
        Convert input sentence to lowercase
        Remove all characters that are not alphanumeric
        IF the "stopwords" input parameter is True THEN
                Remove stop words
        END IF
END FUNCTION

Function my_tokenizer
        Tokenize the input using the function work_tokenize()
        Tag the part of speeches of each word using the function pos_tag()
        IF the word is not an adjective, verb, noun, or adverb
                Append it to the list "lemmas"
        END IF
END FUNCTION

FUNCTION getWordVec
                Initialise an array of zeros with the same dimension as the word embedding
                Assign vector to the word embeddings
END FUNCTION

FUNCTION getPhraseEmbedding
        Initialise an array of zeros with the same dimension as the word embedding of the input
model
        FOR each word in the input phrase,
                Call function getWordVec() with the input model.
                Add the embeddings of each word to the initialized vector
                Return this vector.
        END FOR
END FUNCTION
```

```
//Bag of Words Model
Import Libraries: Corpora from Gensim, Pprint
Split “cleaned_sentences_with_stopwords” by white space into a list of words
Store as list “sentence_words”
Call Dictionary() function to “sentence_words”
FOR each sentence in “sentence_words”
        Call doc2bow() to get BOW matrix
END FOR
FUNCTION ask_question_bow() with input as question
        Apply clean_sentence() with stopwords=False on question.
        FOR each word
                Apply doc2bow() and store as “question_embedding”
        END FOR
        Use cosine similarity to find the question in the dataset that is closest to our input
        question. Initialise max_sim and index_sim as -1
        Apply cosine_similarity between each sentence’s vector embedding in dataframe’s
        questions to the input query questions embedding
        Store the maximum cosine similarity index by updating max_sim and index_sim
accordingly.
END FUNCTION

//TF-IDF Model
Import Libraries: TfidfVectorizer and cosine_similarity from sklearn
Create an instance “tfidf_vectorizer” of the TfidfVectorizer() with tokenizer as “my_tokenizer”.
Apply fit_transform() under the instance “tfidf_vectorizer” on the questions in our data frame
Store as “tfidf_similarity”.
FUNCTION ask_question_tfidf() with question as input
        Apply transform() on the question. Store as “query_vect”.
        Calculate cosine similarity using cosine_similarity() function on “query_vect” and
        “tfidf_matrix”
        Store as “similarity”
        Calculate “max_similarity” by applying argmax() on “similarity”.
END FUNCTION

//Word2Vec Model
        Import Libraries: Word2Vec from gensim.models, gensim.downloader
        Initialise “v2w_model” as None
        Load the "./w2vecmodel.mod" or “word2vec-google-news-300”
        Save the model
        Store dimension of vector embedding as “w2vec_embedding_size”
        FUNCTION ask_question_v2w() with input as question
                FOR each sentence in “cleaned_sentences_with_stopwords”
                        Call getPhraseEmbedding() with w2v model
                        Append to list “sent_embeddings”
                END FOR
                Apply getPhraseEmbedding() to the input question
                Use cosine similarity to find the question in the dataset that is closest to our input
                question
                Initialize max_sim and index_sim as -1
                Apply cosine_similarity between each sentence’s vector embedding in data
                frames questions to the input query questions embedding
                Store the maximum cosine similarity index by updating max_sim and index_sim
                accordingly
END FUNCTION

//GloVe Model
Initialise “glove_model” as None
Load and save the model.
Store dimension of vector embedding as “glove_embedding_size”
FUNCTION ask_question_glove() with input as question
        FOR each sentence in “cleaned_sentences_with_stopwords”,
```

```
            Call function getPhraseEmbedding() with w2v model
            Append to list "sent_embeddings"
        END FOR
        Call function getPhraseEmbedding() to the input question
        Use cosine similarity to find the question in the dataset that is closest to our input
        question. Initialize max_sim and index_sim as -1
        Apply cosine_similarity between each sentence's vector embedding in data frames
        questions to the input query questions embedding
        Store the maximum cosine similarity index by updating max_sim and index_sim
        accordingly
END FUNCTION

//BERT Model
Import Libraries: BertClient from bert_serving.client
Create an instance of BertClient() with parameter check_length=False
FUNCTION ask_question_bert() with input as question
        FOR each sentence in "cleaned_sentences_with_stopwords"
            Call encode() and append to list "sent_bertphrase_embeddings"
        END FOR
        Call encode() to the input question
        Store as "question_embedding"
        Use cosine similarity to find the question in the dataset that is closest to our input
        question.
        Initialize max_sim and index_sim as -1
        Apply cosine_similarity between each sentence's vector embedding in data frames
        questions to the input query questions embedding
        Store the maximum cosine similarity index by updating max_sim and index_sim
        accordingly
END FUNCTION

//Comparing Accuracy of all models
FUNCTION ask_question() with question as input
        Call function ask_question_bow() with question as parameter
        Call function ask_question_tfidf() with question as parameter
        Call function ask_question_glove() with question as parameter
        Call function ask_question_w2v() with question as parameter
        Call function ask_question_bert() with question as parameter
END FUNCTION

END
```

1. **Set up, Preparing and Preprocessing the Dataset**
    - 1.1 Import Libraries: Numpy, Pandas, String, Nltk, Cosine Similarity from Sklearn
    - 1.2 Read Dataset as 'df'
    - 1.3 Add header names to data frame.
    - 1.4 Drop duplicate questions and null values.
2. **Define functions to Clean the Dataset.**
    - 2.1 Import Libraries: Regular Expression, Gensim
    - 2.2 Define function clean_sentence()
        - 2.2.1 Convert input sentence to lowercase.
        - 2.2.2 Remove all characters that are not alphanumeric.
        - 2.2.3 Remove stop words only if the "stopwords" input parameter is True, else do not remove stop words.
    - 2.3 Define function get_cleaned_sentences()

2.2.1 For each “question” in dataset, apply clean_sentences() function

2.2.2 Append Result of 2.2.1 in a list “cleaned_sentences”

2.3 Call get_cleaned_sentences() on df with stopwords=True. Store as “cleaned_sentences_without_stopwords”

2.4 Call get_cleaned_sentences() on “df” with stopwords=False. Store as “cleaned_sentences_with_stopwords”

3. **Define stopwords_list as the stopwords in English and Call an instance of the WordNetLemmatizer() class**
4. **Define function my_tokenizer() to Tokenize the Dataset.**

4.1 Tokenize the input document using the function work_tokenize()

4.2 Tag the part of speeches of each word using the function pos_tag()

4.3 Remove stopwords and punctuation from the document

4.4 If the word is not an adjective, verb, noun, or adverb, append it to the list “lemmas”. Return lemmas.

5. **Define functions to get phrase embeddings.**

5.1 Define the function getWordVec() with inputs as word and model

5.1.1 Initialise an array of zeros with the same dimension as the word embedding of the input model.

5.1.2 Assign vector to the word embeddings of the given model. Return this vector.

5.2 Define the function getPhraseEmbedding() with inputs phrase and model.

5.2.1 Initialise an array of zeros with the same dimension as the word embedding of the input model

5.2.2 For each word in the input phrase, apply to getWordVec() with the input model. Add the embeddings of each word to the initialized vector. Return this vector.

6. **Bag of Words Model**

6.1 Import Libraries: Corpora from Gensim, Pprint

6.2 Split “cleaned_sentences_with_stopwords” by white space into a list of words. Store as list “sentence_words”

6.3 Apply Dictionary() function to “sentence_words”

6.4 Import print

6.5 Apply doc2bow() to each sentence in “sentence_words” to get BOW matrix

6.6 Define function ask_question_bow() with input as question

6.6.1 Apply clean_sentence() with stopwords=False on question. Apply doc2bow() on each word and store as “question_embedding”

6.6.2 Use cosine similarity to find the question in the dataset that is closest to our input question. Initialise max_sim and index_sim as -1. Apply cosine_similarity between each sentence’s vector embedding in dataframe’s questions to the input query questions embedding. Store the maximum cosine similarity index by updating max_sim and index_sim accordingly.

7. **TF-IDF Model**

7.1 Import Libraries: TfidfVectorizer and cosine_similarity from sklearn

7.2 Create an instance “tfidf_vectorizer” of the TfidfVectorizer() with tokenizer

as “my_tokenizer”.

7.3 Apply fit_transform() under the instance “tfidf_vectorizer” on the questions in our data frame. Store as “tfidf_similarity”.

7.4 Define function ask_question_tfidf() with question as input

7.4.1 Apply transform() on the question. Store as “query_vect”.

7.4.2 Calculate cosine similarity using cosine_similarity() function on “query_vect” and “tfidf_matrix”. Store as “similarity”

7.4.3 Calculate “max_similarity” by applying argmax() on “similarity”.

8. Word2Vec Model

8.1 Import Libraries: Word2Vec from gensim.models, gensim.downloader

8.2 Initialise “v2w_model” as None

8.3 Load the "./w2vecmodel.mod" or “word2vec-google-news-300” and save the model.

8.4 Store dimension of vector embedding as “w2vec_embedding_size”

8.5 Define function ask_question_v2w() with input as question

8.5.1 For each sentence in “cleaned_sentences_with_stopwords”, apply getPhraseEmbedding() with w2v model and append to list “sent_embeddings”.

8.5.2 Apply getPhraseEmbedding() to the input question

8.5.3 Use cosine similarity to find the question in the dataset that is closest to our input question. Initialize max_sim and index_sim as -1. Apply cosine_similarity between each sentence’s vector embedding in data frames questions to the input query questions embedding. Store the maximum cosine similarity index by updating max_sim and index_sim accordingly.

9. GloVe Model

9.1 Initialise “glove_model” as None

9.2 Load the "./glovemodel.mod" or “glove-twitter-25” and save the model.

9.3 Store dimension of vector embedding as “glove_embedding_size”

9.4 Define function ask_question_glove() with input as question

9.4.1 For each sentence in “cleaned_sentences_with_stopwords”, apply getPhraseEmbedding() with w2v model and append to list “sent_embeddings”.

9.4.2 Apply getPhraseEmbedding() to the input question

9.4.3 Use cosine similarity to find the question in the dataset that is closest to our input question. Initialize max_sim and index_sim as -1. Apply cosine_similarity between each sentence’s vector embedding in data frames questions to the input query questions embedding. Store the maximum cosine similarity index by updating max_sim and index_sim accordingly.

10. BERT Model

10.1 Import Libraries: BertClient from bert_serving.client

10.2 Create an instance of BertClient() with parameter check_length=False

10.3 Define function ask_question_bert() with input as question

10.3.1 For each sentence in “cleaned_sentences_with_stopwords”, apply encode() and append to list “sent_bertphrase_embeddings”.

10.3.2 Apply encode() to the input question. Store as “question_embedding”.

10.3.3 Use cosine similarity to find the question in the dataset that is closest to our input question. Initialize max_sim and index_sim as -1. Apply cosine_similarity between each sentence's vector embedding in data frames questions to the input query questions embedding. Store the maximum cosine similarity index by updating max_sim and index_sim accordingly.

11. Comparing Accuracy of all models. Define function ask_question() with question as input

11.1 Call ask_question_bow() with question as parameter.
11.2 Call ask_question_tfidf() with question as parameter.
11.3 Call ask_question_glove() with question as parameter.
11.4 .Call ask_question_w2v() with question as parameter.
11.5 Call ask_question_bert() with question as parameter.

Explanation:

To achieve our aim of developing a model, we first prepare the Dataset using Web Scraping. We then preprocess the data obtained by cleaning all sentences and making them free of any stopwords. We then implement various models like the Bag of Words model, Word2Vec Model, and Glove Models, TF-IDF and BERT, and compare the accuracies of the answers retrieved from them.

4. EXPERIMENT AND RESULTS

STEP 1: Imported the libraries and the dataset.

The following libraries were installed and imported.

- Pandas
- Numpy
- String
- Sklearn - cosine_similarity
- Nltk - word_tokenize, WordNetLemmatizer, stopwords, wordnet, pos_tag, TfidfVectorizer, cosine_similarity

```
import numpy as np import pandas as pd
import string

from nltk.tokenize import word_tokenize from
nltk.stem import WordNetLemmatizer from
nltk.corpus import stopwords
from nltk.corpus import wordnet from nltk
import pos_tag

from sklearn.metrics.pairwise import
```

cosine_similarity

```
df=pd.read_csv("NLP DATASET.csv");
df.columns=["questions","answers"];
df.head(10)
```

OUTPUT:

	questions	answers
0	What is a novel coronavirus?	A novel coronavirus is a new coronavirus that ...
1	Why is the disease being called coronavirus di...	On February 11, 2020 the World Health Organiza...
2	How does the virus spread?	The virus that causes COVID-19 is thought to s...
3	Can I get COVID-19 from food (including restau...	Currently there is no evidence that people can...
4	Will warm weather stop the outbreak of COVID-19?	It is not yet known whether weather and temper...
5	What is community spread?	Community spread means people have been infect...
6	Can mosquitoes or ticks spread the virus that ...	At this time, CDC has no data to suggest that ...
7	Does CDC recommend the use of masks to prevent...	Wear masks in public settings when around peop...
8	Is it safe to get care for my other medical co...	It is important to continue taking care of you...
9	Am I at risk for COVID-19 from mail, packages,...	There is still a lot that is unknown about COV...

Our dataset has 2 columns, questions and answers. We have scraped around 380 questions from the internet from all the reliable resources.

STEP 2: Pre-process the Dataset.

For this task we are performing the following pre-processing:

1. Drop the duplicate questions.
2. Drop the null values.

Our dataset now comprises 349 rows. Its dimensions are (349,2)

```
df = df.drop_duplicates(subset='questions')
df = df.dropna()
df.shape
```

OUTPUT:

```
In [47]: 1 df.shape
Out[47]: (380, 2)
```

STEP 3: Visualising the Dataset.

Some Data Visualisation libraries are imported such as: Matplotlib, Seaborn, TextBlob and WorkCloud.

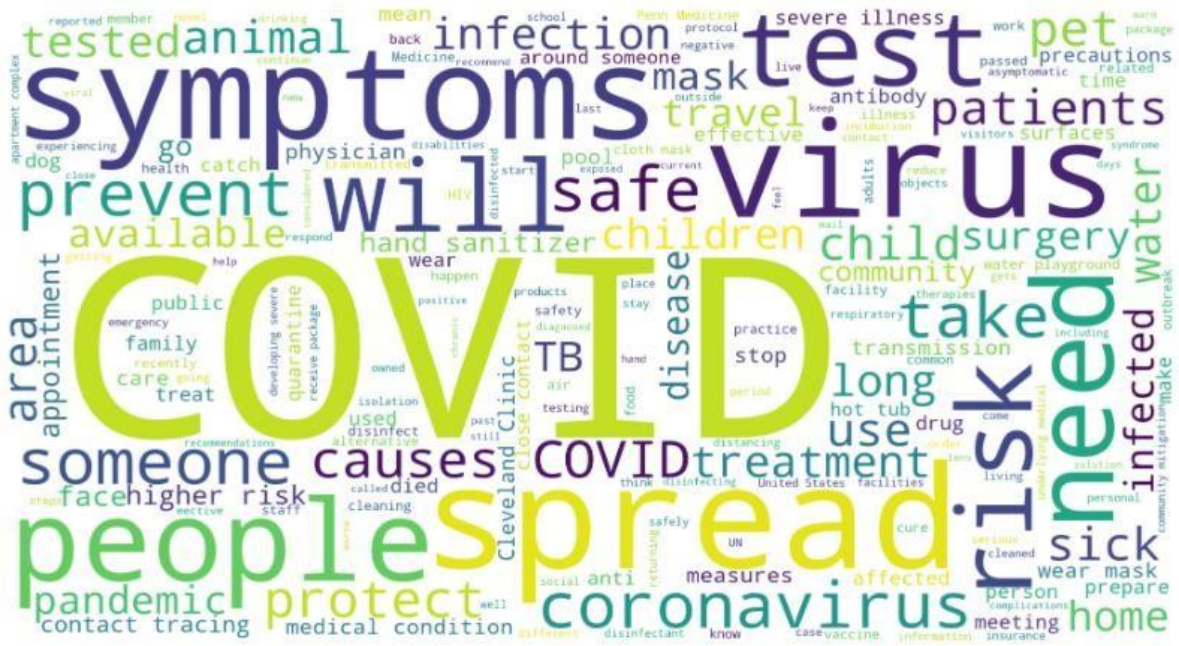
```
import matplotlib.pyplot as plt
import seaborn as sns
%pylab inline
from textblob import TextBlob
from wordcloud import WordCloud
```

```
import sklearn
```

Word Clouds of the questions are made. Here the more the frequency of a term in the input text, larger is the size of the word in the Word Cloud.

```
text = ' '.join(df.questions)
cloud = WordCloud(background_color='white', width=1920,
height=1080).generate(text)
plt.figure(figsize=(32, 18))
plt.axis('off')
plt.imshow(cloud)
plt.savefig('questions_wordcloud.png')
```

OUTPUT:



Word Clouds of the answers are also made.

```
text = ' '.join(df.answers)
cloud = WordCloud(background_color='white', width=1920,
height=1080).generate(text)
plt.figure(figsize=(32, 18))
plt.axis('off')
plt.imshow(cloud)
plt.savefig('questions_wordcloud.png')
```

OUTPUT:

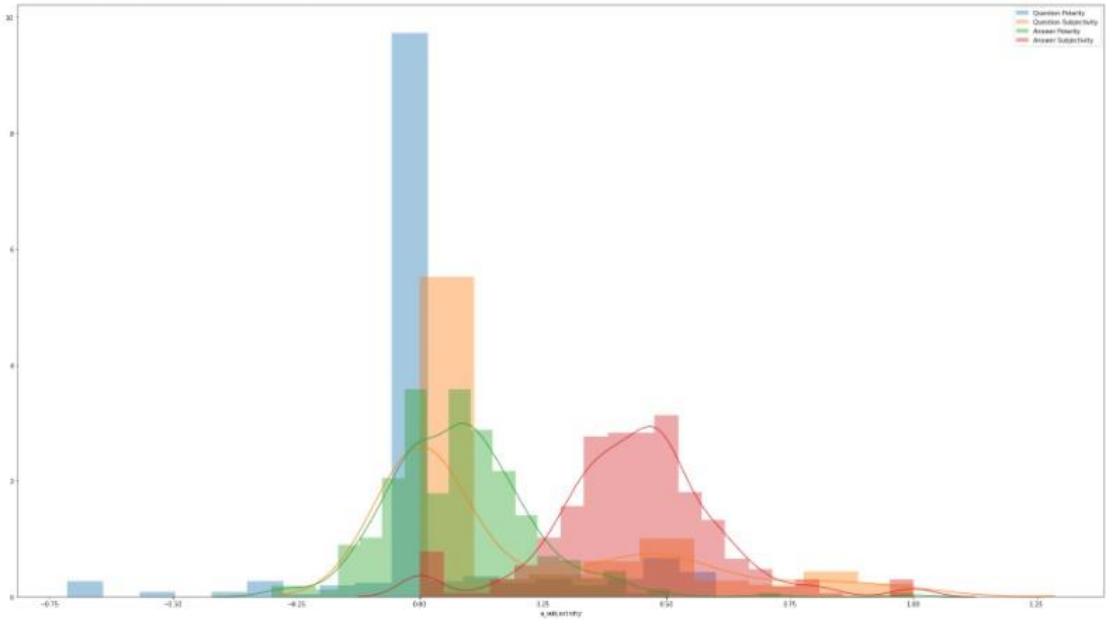
Now, the sentiment polarity and subjectivity is calculated and plotted.

```
get_polarity = lambda x: TextBlob(x).sentiment.polarity
get_subjectivity = lambda x: TextBlob(x).sentiment.subjectivity

df['q_polarity'] = df.questions.apply(get_polarity)
df['a_polarity'] = df.answers.apply(get_polarity)
df['q_subjectivity'] = df.questions.apply(get_subjectivity)
df['a_subjectivity'] = df.answers.apply(get_subjectivity)
```

OUTPUT:

```
Out[58]: <matplotlib.legend.Legend at 0x1a0efc29640>
```



<u>STEP 4:</u> Cleaning the Dataset

Functions to clean the dataset are defined. We obtained two types of cleaned sentences, one with stopwords and one without stopwords. These are used later in the code.

```
import re
import gensim
from gensim.parsing.preprocessing import remove_stopwords
def clean_sentence(sentence, stopwords=False):
    sentence = sentence.lower().strip()
    sentence = re.sub(r'[^a-z0-9\s]', '', sentence)
    if stopwords:
        sentence = remove_stopwords(sentence)
    return sentence
def get_cleaned_sentences(df,stopwords=False):
    sents=df[["questions"]];
    cleaned_sentences=[]
    for index,row in df.iterrows():
        cleaned=clean_sentence(row["questions"],stopwords);
        cleaned_sentences.append(cleaned);
    return cleaned_sentences;
cleaned_sentences_without_stopwords = get_cleaned_sentences(df, stopwords = True)
cleaned_sentences_with_stopwords=get_cleaned_sentences(df, stopwords=False)
```

STEP 5: Tokenization of the Dataset

Function to Tokenize the dataset is defined. This function is later used to implement the TF-IDF model.

```
stopwords_list = stopwords.words('english')
lemmatizer = WordNetLemmatizer()
def my_tokenizer(doc):
    words = word_tokenize(doc)
    pos_tags = pos_tag(words)
    non_stopwords = [w for w in pos_tags if not w[0].lower() in stopwords_list]
    non_punctuation = [w for w in non_stopwords if not w[0] in string.punctuation]
    lemmas = []
    for w in non_punctuation:
        if w[1].startswith('J'):
            pos = wordnet.ADJ
        elif w[1].startswith('V'):
            pos = wordnet.VERB
        elif w[1].startswith('N'):
            pos = wordnet.NOUN
        elif w[1].startswith('R'):
            pos = wordnet.ADV
        else:
            pos = wordnet.NOUN
        lemmas.append(lemmatizer.lemmatize(w[0], pos))
    return lemmas
```

STEP 6: Phrase Embeddings of Dataset

Functions are defined to obtain the Phrase Embeddings of the Cleaned sentences using a particular model, such as Glove, Word2Vec or BERT

```
def getWordVec(word,model):
        samp=model['computer'];
        vec=[0]*len(samp);
        try:
                vec=model[word];
        except:
                vec=[0]*len(samp);
        return (vec)

def getPhraseEmbedding(phrase,embeddingmodel):
        samp=getWordVec('computer', embeddingmodel);
        vec=np.array([0]*len(samp));
        den=0;
        for word in phrase.split():
           den=den+1;          vec=vec+np.array(getWordVec(word,embeddingmodel));
        return vec.reshape(1, -1)
```

STEP 7: Bag of Words Model

```
from gensim import corpora
import pprint

sentence_words = [[word for word in document.split() ]
        for document in cleaned_sentences_with_stopwords]
dictionary = corpora.Dictionary(sentence_words)
#for key, value in dictionary.items():
   #print(key, ' : ', value)
bow_corpus = [dictionary.doc2bow(text) for text in sentence_words]
for sent,embedding in zip(sentences,bow_corpus):
   print(sent)
   print(embedding)
```

```
def ask_question_bow(question):
   question=clean_sentence(question,stopwords=False);
   question_embedding = dictionary.doc2bow(question.split())
   max_sim=-1;
   index_sim=-1;
   for index,faq_embedding in enumerate(bow_corpus):
      sim=cosine_similarity(faq_embedding,question_embedding)[0][0];
      if sim>max_sim:
         max_sim=sim;
         index_sim=index;
   print("*****BAG OF WORDS MODEL*****\n")
```

```
    print('Your question:\t\t', question)
    print('Closest question found:\t', df['questions'].iloc[index_sim])
    print("\n")
    print('Cosine Similarity: {:.2%}'.format(max_sim))
    print("\n")
    print('Answer:', df.iloc[index_sim]['answers'])
```

STEP 8: TF- IDF Model

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer(tokenizer=my_tokenizer)
tfidf_matrix = tfidf_vectorizer.fit_transform(tuple(df['questions']))
def ask_question_tfidf(question):
    query_vect = tfidf_vectorizer.transform([question])
    similarity = cosine_similarity(query_vect, tfidf_matrix)
    max_similarity = np.argmax(similarity, axis=None)

    print("*****TF-IDF MODEL*****\n")
    print('Your question:\t\t', question)
    print('Closest question found:\t', df.iloc[max_similarity]['questions'])
    print("\n")
    print('Cosine Similarity: {:.2%}'.format(similarity[0, max_similarity]))
    print("\n")
    print('Answer:', df.iloc[max_similarity]['answers'])
```

STEP 9: Word2Vec Model

```
from gensim.models import Word2Vec
import gensim.downloader as api

v2w_model=None;
try:
    v2w_model = gensim.models.KeyedVectors.load("./w2vecmodel.mod")
    print("Loaded w2v model")
except:
    v2w_model = api.load('word2vec-google-news-300')
    v2w_model.save("./w2vecmodel.mod")
    print("Saved glove model")
w2vec_embedding_size=len(v2w_model['computer']);
```

```
def ask_question_w2v(question):
    sent_embeddings=[];
    for sent in cleaned_sentences_with_stopwords:
        sent_embeddings.append(getPhraseEmbedding(sent,v2w_model));
question_embedding=getPhraseEmbedding(question,v2w_model);
    max_sim=-1;
    index_sim=-1;
```

```
    for index,faq_embedding in enumerate(sent_embeddings):
        sim=cosine_similarity(faq_embedding,question_embedding)[0][0];
        if sim>max_sim:
            max_sim=sim;
            index_sim=index;

    print("*****WORD2VEC MODEL*****\n")
    print('Your question:\t\t', question)
    print('Closest question found:\t', df['questions'].iloc[index_sim])
    print("\n")
    print('Cosine Similarity: {:.2%}'.format(max_sim))
    print("\n")
    print('Answer:', df.iloc[index_sim]['answers'])
```

STEP 10: GloVe Model

```
glove_model=None;
try:
    glove_model = gensim.models.KeyedVectors.load("./glovemodel.mod")
    print("Loaded glove model")
except:
    glove_model = api.load('glove-twitter-25')
    glove_model.save("./glovemodel.mod")
    print("Saved glove model")
glove_embedding_size=len(glove_model['computer']);
```

```
def ask_question_glove(question):
    sent_embeddings=[];
    for sent in cleaned_sentences_with_stopwords:
        sent_embeddings.append(getPhraseEmbedding(sent,glove_model));
    question_embedding=getPhraseEmbedding(question,glove_model);
    max_sim=-1;
    index_sim=-1;
    for index,faq_embedding in enumerate(sent_embeddings):
        sim=cosine_similarity(faq_embedding,question_embedding)[0][0];
        if sim>max_sim:
            max_sim=sim;
            index_sim=index;

    print("*****GLOVE MODEL*****\n")
    print('Your question:\t\t', question)
    print('Closest question found:\t', df['questions'].iloc[index_sim])
    print("\n")
    print('Cosine Similarity: {:.2%}'.format(max_sim))
    print("\n")
    print('Answer:', df.iloc[index_sim]['answers'])
```

STEP 11: BERT Model

```
from bert_serving.client import BertClient
bc = BertClient(check_length=False)
res=bc.encode(['ML', 'AI'])

def ask_question_bert(question):
    question = clean_sentence(question,stopwords=False);
    sent_bertphrase_embeddings=[];
    for sent in cleaned_sentences_with_stopwords:
        sent_bertphrase_embeddings.append(bc.encode([sent]));
    question_embedding=bc.encode([question]);
    max_sim=-1;
    index_sim=-1;
    for index,faq_embedding in enumerate(sent_bertphrase_embeddings):
        sim=cosine_similarity(faq_embedding,question_embedding)[0][0];
        if sim>max_sim:
            max_sim=sim;
            index_sim=index;

    print("*****BERT MODEL*****\n")
    print('Your question:\t\t', question)
    print('Closest question found:\t', df['questions'].iloc[index_sim])
    print("\n")
    print('Cosine Similarity: {:.2%}'.format(max_sim))
    print("\n")
    print('Answer:', df.iloc[index_sim]['answers'])
```

STEP 12: Compare the accuracies of all the models

```
def ask_question(question):
    ask_question_bow(question)
    print("\n ------------------- \n")
    ask_question_tfidf(question)
    print("\n ------------------- \n")
    ask_question_w2v(question)
    print("\n ------------------- \n")
    ask_question_glove(question)
    print("\n ------------------- \n")
    ask_question_bert(question)

question = "What is Coronavirus?"
ask_question(question)
```

4.2 OUTPUT:

Question: “What is Coronavirus?”

```
*****BAG OF WORDS MODEL*****

Your question:           what is coronavirus
Closest question found:  Why is the disease being called coronavirus disease 2019, COVID-19?


Cosine Similarity: 100.00%


Answer: On February 11, 2020 the World Health Organization announced an ocial name for the dis
novel coronavirus outbreak, rst identied in Wuhan China. The new name of this disease is coron
ted as COVID-19. In COVID-19, ‘CO’ stands for ‘corona,’ ‘VI’ for ‘virus,’ and ‘D’ for disease.
eferred to as “2019 novel coronavirus” or “2019-nCoV”. There are many types of human coronavir
only cause mild upper-respiratory tract illnesses. COVID-19 is a new disease, caused by a nove
as not previously been seen in humans.

--------------------

*****TF-IDF MODEL*****

Your question:           What is Coronavirus?
Closest question found:  What is a coronavirus?


Cosine Similarity: 100.00%


Answer: Coronaviruses are a large family of viruses which may cause illness in animals or humans
viruses are known to cause respiratory infections ranging from the common cold to more severe di
Respiratory Syndrome (MERS) and Severe Acute Respiratory Syndrome (SARS). The most recently disc
oronavirus disease COVID-19.

--------------------

*****WORD2VEC MODEL*****

Your question:           What is Coronavirus?
Closest question found:  What is COVID-19?


Cosine Similarity: 82.34%


Answer: COVID-19, otherwise known as coronavirus disease 2019, is a new illness that affec
particularly severe in older populations and people with underlying health conditions.

--------------------

*****GLOVE MODEL*****

Your question:           What is Coronavirus?
Closest question found:  What is COVID-19?


Cosine Similarity: 97.97%


Answer: COVID-19, otherwise known as coronavirus disease 2019, is a new illness that affects y
particularly severe in older populations and people with underlying health conditions.

--------------------
```

```
*****BERT MODEL*****

Your question:           what is coronavirus
Closest question found:  What is a coronavirus?


Cosine Similarity: 98.71%


Answer: Coronaviruses are a large family of viruses which may cause illness in animals c
viruses are known to cause respiratory infections ranging from the common cold to more s
Respiratory Syndrome (MERS) and Severe Acute Respiratory Syndrome (SARS). The most recer
oronavirus disease COVID-19.
```

4.3 TEST CASES:

1. Should I make my own hand sanitizer if I can't find it in the stores?

Model	Closest Question Found	Results
Bag of Words	How does the virus spread?	FAIL
TF - IDF	Where can I buy hand sanitizer? If I can't find it in the store, can I make my own?	PASS
Word2Vec	Can I make my own hand sanitizer?	PASS
GloVe	Where can I buy hand sanitizer? If I can't find it in the store, can I make my own?	PASS
BERT	Should I make my own hand sanitizer if I can't find it in the stores?	PASS

2. Is my child with an underlying medical condition at higher risk for severe illness from COVID-19?

Model	Closest Question Found	Result
Bag of Words	What is community spread?	FAIL
TF - IDF	Is my child with an underlying medical condition at higher risk for severe illness from COVID-19	PASS
Word2Vec	Is my child with an underlying medical condition at higher risk for severe illness from COVID-19	PASS
GloVe	Is my child with an underlying	PASS

	medical condition at higher risk for severe illness from COVID-19	
BERT	Is my child with an underlying medical condition at higher risk for severe illness from COVID-19	PASS

3. When will a vaccine for Coronavirus be developed?

Model	Closest Question Found	Result
Bag of Words	What is a novel coronavirus?	FAIL
TF - IDF	Is there a vaccine?	PASS
Word2Vec	When will a COVID-19 vaccine be available?	PASS
GloVe	When will a COVID-19 vaccine be available?	PASS
BERT	When will a COVID-19 vaccine be available?	PASS

4. If I participate in contact tracing for COVID-19 using a digital tool, is my personal health information secure?

Model	Closest Question Found	Result
Bag of Words	What is a novel coronavirus?	FAIL
TF - IDF	If I participate in contact tracing for COVID-19 using a digital tool, is my personal health information secure?	PASS
Word2Vec	If I participate in contact tracing for COVID-19 using a digital tool, is my personal health information secure?	PASS
GloVe	If I participate in contact tracing for COVID-19 using a digital tool, is my personal health information secure?	PASS
BERT	If I participate in contact tracing for COVID-19 using a digital tool, is my personal health information secure?	PASS

5. What precautions can I take to stay safe from the virus?

Model	Closest Question Found	Result
Bag of Words	What cleaning products should I use to protect against COVID-19?	PASS
TF - IDF	Should wastewater workers take extra precautions to protect themselves from the virus that causes COVID-19?	FAIL
Word2Vec	Should wastewater workers take extra precautions to protect themselves from the virus that causes COVID-19?	FAIL
GloVe	What steps can my family take to reduce our risk of getting COVID-19?	PASS
BERT	What precautions should I take to prevent COVID-19 in clinical practice?	PASS

ACCURACY:

Model	Accuracy
Bag of Words	44.4
TF - IDF	92.3
Word2Vec	82.9
GloVe	87.2
BERT	96.5

5. CONCLUSION

In this project, we examined the task of automatically retrieving a suitable response to customer questions from the FAQs of Various Government Sites answering people's queries on the recent Pandemic Covid-19. Often websites have comprehensive FAQs, but manually searching and finding the answer to a specific question from these FAQs is not trivial and is very time-consuming. The purpose of this project is to develop a model that will answer user

queries by automatically retrieving the closest question and answer from predefined FAQs which is the most appropriate.

We used various Models to implement our Project and compared the accuracy of answers we retrieved from them. One of them was the Bag of Words model in which the closest matching answer can be retrieved by finding the cosine similarity of the query vector with each of the FAQ question vectors. We found out that BOW representation did not do very well and retrieved the wrong answer since it is looking for an exact word match. We then implemented the Word2Vec skip-gram model to make word embeddings and then word embeddings were converted to phrase embeddings which were used in this model and another model as well which is popularly known as the Glove model which gave similar outputs but with different accuracies. While both the techniques are popular, glove performs better on some datasets while the word2vec skip-gram model performs better on some. Even after implementing these, the accuracy and few test cases weren't up to mark, so we implemented an advanced model called BERT and it retrieved the most accurate answers to the user's questions.

So, this model is very effective and efficient to help common people find the answer to their query from one place instead of going to different websites and not finding the solution to their query.

FUTURE ENHANCEMENTS:

Future Enhancements should be focussed on optimization and modularization of already existing, better performing Question Answering techniques. This will motivate the re-use of quality modules and help researchers focus and isolate core research problems. This model that has been developed can be put to use in the real world by embedding this feature into an application for a common user to retrieve correct information from reliable resources.

- Github Repository Link of this Project: https://github.com/Blackshine99/QA_System_COVID19

6. REFERENCES

[1] Anne R. Diekema, Ozgur Yilmazel, and Elizabeth D. Liddy,"Evaluation of Restricted Domain Question-Answering Systems".(1999) Center for Natural Language Processing.

[2] Abdessamad Echihabi and Daniel Marcu "A Noisy-Channel Approach to Question Answering", (2000).

[3] Benamara.F.,"Cooperative Question Answering in Restricted Domains: the WEBCOOP Experiment",2004. In Proceedings of the ACL Workshop on Question Answering in Restricted Domains.

[4] Berger, A., Caruana, R., Cohn, D., Freitag, D., and Mittal, V. 2000. Bridging the lexical chasm: statistical approaches to answer-finding, In Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval, pp. 192–99.

[5] C. Kwok, O.Etzioni, D. Weld.“Scaling Question Answering to the Web”. In Proceedings of WWW10, Hong Kong, 2001.

[6] Chung,H.,Han, K., Rim, H., Kim, S., Lee, J., Song, Y. & Yoon, D. “A Practical QA System in Restricted Domains” (2004). In Proceedings of the ACL Workshop on Question Answering in Restricted Domains.

[7] Dell Zhang and Wee Sun Lee.“A Web-based Question Answering System” (2002) ,October 31.

[8] D.R.Radev,W.Fan,H. Qi, H. Wu and A. Grewal.“Probabilistic Question Answering from the Web”,(2002), In Proceedings of the 11t World Wide Web Conference,Hawaii.

[9] Diekema A.R, Yilmazel Ozgur, and Liddy E.D. “ Evaluation of Restricted Domain Question-Answering Systems” (2004).In Proceedings of the ACL2004 Workshop on Question Answering in Restricted Domain ,p.p 2-7,.

[10] Ellen Riloff and MichaelThelen.“A Rule Based Question Answering System for Reading Comprehension Tests”,(2003){rioloff,thelen}.

[11] Green W, Chomsky C, and Laugherty K..BASEBALL: An automatic question answerer.(1961). Proceedings of the Western Joint ComputerConference, p.p. 219-224.

[12] Hermjakob.U.“Parsing and Question Classification for Question Answering” (2001). In Proceedings of the ACL Workshop on Open-Domain Question Answering.

[13] Agichtein, E., and Savenkov, D. 2016. CRQA: Crowd-Powered Real-Time Automatic Question Answering System. HCOMP.

[14] Ahmed, W., and Babu, P. A. 2016. Answer Extraction and Passage Retrieval for Question Answering Systems. ISSN:2278-1323 International Journal of Advanced Research in Computer Engineering and Technology (IJARCET), 5(12).

[15] Bouma, G. 2005. Linguistic Knowledge and Question Answering. In: Proceeding KRAQ ’06 Proceedings of the Workshop KRAQ’06 on Knowledge and Reasoning for Language Processing. pp. 2–3.

[16] Green, B. F., Wolf, A. K., Chomsky, C., and Laughery, K. 1961. Baseball: An automatic question answerer. Readings in natural language processing. pp. 545–9. Morgan Kaufmann Publishers Inc.

[17] Hans, A. F., Feiyu, U. K., Uszkoreit, X. H., Brigitte, B. C., and Schäfer, J. U. 2006. Question answering from structured knowledge sources, German Research Center for Artificial Intelligence, DFKI.

[18] Kan, K.L., and Lam,W. 2006. Using semantic relations with world knowledge for question answering. In: Proceedings of TREC.

[19] Kolomiyets, O. 2011. A survey on question answering technology from an information retrieval perspective. Inf. Sci. 181(24):5412–34.

[20] Monz, C. 2004. Minimal span weighting retrieval for question answering, In SIGIR Workshop on Information Retrieval for Question Answering, pp. 23–30.