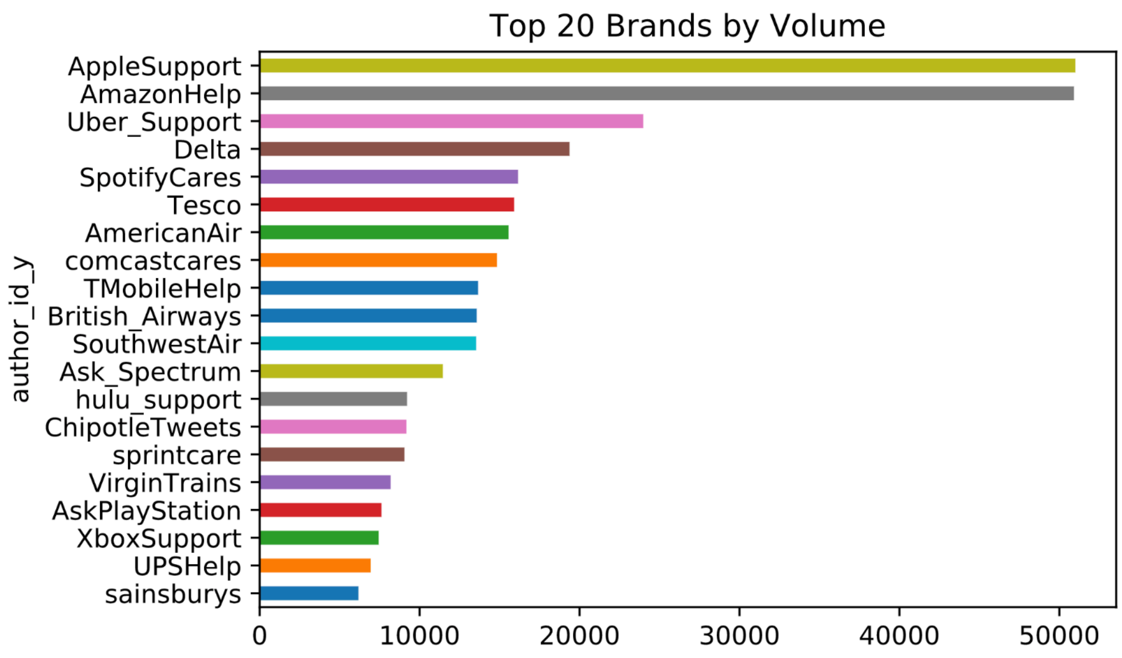
AIWIR ASSIGNMENT I

TEAM MEMBERS:

PES2UG20CS009	Abhay Warrier
PES2UG20CS134	Harsha Pai
PES2UG20CS174	Kunal E
PES2UG20CS175	Kuval Garg

About the Corpus / Dataset:

The Customer Support on Twitter dataset is a large, modern corpus of tweets and replies to aid innovation in natural language understanding and conversational models, and for study of modern customer support practices and impact.



The Customer Support on Twitter dataset offers a large corpus of modern English (mostly) conversations between consumers and customer support agents on

Twitter, and has three important advantages over other conversational text datasets:

- **Focused** - Consumers contact customer support to have a specific problem solved, and the manifold of problems to be discussed is relatively small, especially compared to unconstrained conversational datasets like the reddit Corpus.
- **Natural** - Consumers in this dataset come from a much broader segment than those in the Ubuntu Dialogue Corpus and have much more natural and recent use of typed text than the Cornell Movie Dialogs Corpus.
- **Succinct** - Twitter's brevity causes more natural responses from support agents (rather than scripted), and to-the-point descriptions of problems and solutions. Also, its convenient in allowing for a relatively low message limit size for recurrent nets.

Description of the Dataset:

The dataset is a CSV, where each row is a tweet. The different columns are described below. Every conversation included has at least one request from a consumer and at least one response from a company. Which user IDs are company user IDs can be calculated using the inbound field.

Different Fields in the dataset

tweet_id

A unique, anonymized ID for the Tweet. Referenced by response_tweet_id and in_response_to_tweet_id.

author_id

A unique, anonymized user ID. @s in the dataset have been replaced with their associated anonymized user ID.

inbound

Whether the tweet is "inbound" to a company doing customer support on Twitter. This feature is useful when re-organizing data for training conversational models.

created_at

Date and time when the tweet was sent.

text

Tweet content. Sensitive information like phone numbers and email addresses are replaced with mask values like __email__.

response_tweet_id

IDs of tweets that are responses to this tweet, comma-separated.

in_response_to_tweet_id

ID of the tweet this tweet is in response to, if any.

tweet_id	author_id	inbound	created_at	text	response_...
119237	105834	True	Wed Oct 11 06:55:44 +0000 2017	@AppleSupport causing the reply to be disregarded and the tapped notification under the keyboard is ...	119236
119238	ChaseSupport	False	Wed Oct 11 13:25:49 +0000 2017	@105835 Your business means a lot to us. Please DM your name, zip code and additional details about ...	
119239	105835	True	Wed Oct 11 13:00:09 +0000 2017	@76328 I really hope you all change but I'm sure you won't! Because you don't have to!	119238
119240	VirginTrains	False	Tue Oct 10 15:16:08 +0000 2017	@105836 LiveChat is online at the moment - https://t.co/SY94VtU8Kq or contact 03331 031 031 option 1...	119241

The dataset was taken from kaggle:
Below is the link to the dataset:
https://www.kaggle.com/datasets/thoughtvector/customer-support-on-twitter

Data structures used:

- **Data frames**:
  To handle csv files. They help us with handling csv files a lot easier by assigning rows and columns to the file. This helps work with the column we need and make necessary pre-processing steps to our dataset.

- **Lists**:
  Python lists are extremely simple to work with and are very flexible in nature. Python lists unlike arrays aren't very strict, Lists are heterogeneous which means you can store elements of different datatypes in them. We use them to store lemmatized text,etc.

- **Dictionary**:
  The fastest way to repeatedly lookup data with millions of entries in Python is using dictionaries. Because dictionaries are the built-in mapping type in Python thereby they are highly optimized.We use them to store in the inverted index, positional index,ect.

- **Arrays**:
  We have used TfidfVectorizer to convert necessary items into arrays to determine cosine similarity. This helps us find the similarity index for a given query.

Screenshot for:
Boolean Intersection Query

```
def get_intersection_postings(word1, word2):
    flag = False
    start=time.time()
    required = []
    answer = {}
    dictionary_items = postings.items()
    for i in dictionary_items:
        if(i[0] == word1):
            required.append(i)
        if(i[0] == word2):
            required.append(i)
        else:
            continue

    indexes = []
    list1 = []
    list2 = []

    for i in required:
        word, posting2 = i
        frequency, index = posting2[0], posting2[1]
        indexes.append(index)

    list1, list2 = indexes[0], indexes[1]

    list3 = [value for value in list1 if value in list2]
    answer[word1+ " AND " + word2]= list(set(list3))

    end=time.time()
    time_taken=end-start       #Time

    if len(list3):
        print(answer)
        print("Time taken to fetch (boolean query): ",time_taken,"seconds")
    else:
        print("No intersection possible")
```

✓ 0.0s

```
get_intersection_postings("apologies","help")
```

✓ 0.0s

```
{'apologies AND help': [81]}
Time taken to fetch (boolean query):  0.0 seconds
```

Screenshot for:

Boolean Query Union

```python
def get_union_postings(word1, word2):
    flag = False
    start=time.time()
    required = []
    answer = {}
    dictionary_items = postings.items()
    for i in dictionary_items:
        if(i[0] == word1):
            required.append(i)
        if(i[0] == word2):
            required.append(i)
        else:
            continue
    indexes = []
    list1 = []
    list2 = []
    for i in required:
        word, posting2 = i
        frequency, index = posting2[0], posting2[1]
        indexes.append(index)

    list1, list2 = indexes[0], indexes[1]

    list3 = list1 + list2
    answer[word1+ " OR " + word2]= list(set(list3))
    end=time.time()
    time_taken=end-start
    if len(list3):
        print(answer)
        print("Time taken to fetch (boolean query): ",time_taken,"seconds")
    else:
        print("No Union possible")
```

✓ 0.0s

```python
get_union_postings("apologies","help")
```

✓ 0.0s

```
{'apologies OR help': [35, 99, 76, 81, 24, 28, 61, 31]}
Time taken to fetch (boolean query):  0.0 seconds
```

Screenshot for:

Result with inverted index:on free text queries with rank based on similarity

```
[ ]  # Inverted index
     def generate_inverted_index(data: list):
         inv_idx_dict = {}

         for index, doc_text in enumerate(data):
             for word in doc_text.split():
                 if word not in inv_idx_dict.keys():
                     inv_idx_dict[word] = [index]
                 elif index not in inv_idx_dict[word]:
                     inv_idx_dict[word].append(index)
         return inv_idx_dict
```

```
[ ]  inverted_index = generate_inverted_index(filtered_Sentence)
     inverted_index
```

```
{'understand': [0, 9],
 'would': [0, 16, 18, 36],
 'like': [0, 9, 11, 16, 18, 76],
 'assist': [0, 3, 9, 26],
 'need': [0, 20, 21, 22, 30, 33, 38],
 'get': [0, 25, 37, 63, 72, 99],
 'private': [0, 2, 3, 5, 13],
 'secured': [0],
 'link': [0, 13, 23],
 'propose': [1],
```

```python
# Positional index
vocab = []
postings = {}
def generate_positional_index(data: list):
  for index,doc_text in enumerate(data):
      for word in doc_text.split():
          if word not in vocab:
              vocab.append(word)
          wordId = vocab.index(word)
          if word not in postings:
              postings[word] = [index]
          else:
              postings[word].append(index)
          #print(wordId,word)
  for i in postings:
      postings[i]=[len(set(postings[i])),list(set(postings[i]))]
  dictionary_items = postings.items()
  for i in dictionary_items:
    print(i)
  #print(postings)
```

```python
#term->[frequency,[position]]
pos_index = generate_positional_index(filtered_Sentence)
pos_index
```

```
('understand', [2, [0, 9]])
('would', [4, [0, 16, 18, 36]])
('like', [6, [0, 9, 11, 76, 16, 18]])
```

Screenshot for:
Retrieve relevant text using similarity index

```
[ ]  # Retrieved documents ranked by similarity score

     def search_similiar_documents():
         querry = str(input("Enter the word you want to search for: "))
         print(f"The documents similar to '{querry}' has been found in: ")
         inv_index = inverted_index[querry]
         similarity(inv_index)

[ ]  try:
       search_similiar_documents()
     except:
       print("No documents")

     Enter the word you want to search for: frustrations
     The documents similar to 'frustrations' has been found in:
     - Document 81: similarity score = 0.20566293021601065
     apologies sent dm help becky
     - Document 75: similarity score = 0.16080911068964043
     happy halloween since old trick treat look forward 3 booritos got mine earlier
     - Document 65: similarity score = 0.1437668254778442
     guac happy great experience becky
     The documents after being ranked are
     apologies sent dm help becky
     guac happy great experience becky
     happy halloween since old trick treat look forward 3 booritos got mine earlier
```

Screenshot for:
Result of Phrase queries

```
        final_tweets_id=[]
        pos_idx = []
        for p in ans:
            held_for_now=filtered_Sentence[p].split()

            if( held_for_now.index(str_to_process[0]) == (held_for_now.index(str_to_process[1])-1) ):
                final_tweets_id.append(p)
                pos_idx.append(len(final_tweets_id))
                pos_idx.append(final_tweets_id)

        end=time.time()
        time_taken=end-start

        print("The phrase is present in tweet ids:",pos_idx)
        print("Time taken to fetch the phrase query: ",time_taken,"seconds")

[ ]  # Single/Multiple term phrase query along with time taken to search

     try:
       ph_q = input("Enter the phrase query: ")
       get_phrase_query(ph_q)
     except:
       print("Phrase not found in any tweet")

     The phrase is present in tweet ids: [1, [0]]
     Time taken to fetch the phrase query:  1.4543533325195312e-05 seconds
```

Screenshot for:
Result of Wild Card queries

```
[ ]  # Wildcard query
     import re

     # Add or remove the words in this list to vary the results
     wordlist = lemma_word
     wild = input("Enter the wildcard query to be searched for: ")

     print("\nResults match:")
     for word in wordlist:
             # The .+ symbol is used in place of * symbol
             if re.search(wild+'.+', word) or re.search(wild+'.', word):
                     print (word)

     Enter the wildcard query to be searched for: frustr

     Results match:
     hello apologies frustrations inconvenience happy look mg
     frustrated ordered dinner saturday using app order wrong charged credit card twice
```

Screenshot for [Any one additional functionality]
Query to list out the number of non alpha numeric characters encountered in the corpus

```
# Additional query to list out the number of non alpha-numeric characters encountered in the corpus

patterns= [r'\W+']
phrase = str(lemma_word)

print("Number of non alpha-numeric characters in the corpus: ")
for p in patterns:
    match= re.findall(p, phrase)
    print(len(match))

Number of non alpha-numeric characters in the corpus:
684
```

Screenshot for
Relevance feedback + reranking of documents

```python
def relevance_feedback(rel_word):
    print("The documents after relevnce feedback is")
    inv_index = inverted_index[word]
    similarity(inv_index)
```

✓ 0.0s Python

```python
word = str(input("Enter the words you found relevent: "))
relevance_feedback(word)
```

✓ 1.5s Python

```
The documents after relevnce feedback is
- Document 61: similarity score = 0.3888598208575087
sorry please tell us help becky
- Document 99: similarity score = 0.2849917144592756
get help already
- Document 81: similarity score = 0.24012822935206632
apologies sent dm help becky
- Document 6: similarity score = 0.5705909047816624
worst customer service
- Document 61: similarity score = 0.21611304132574524
sorry please tell us help becky
- Document 25: similarity score = 0.19909642745461265
yo spectrum customer service reps super nice imma start trippin get service going
- Document 28: similarity score = 0.18701828103660634
help arrived sorry see trouble help hsb
- Document 87: similarity score = 0.17088803753286003
incredibly concerning please provide details investigate becky
- Document 99: similarity score = 0.16344840920046044
get help already
- Document 69: similarity score = 0.5616394411955679
incredibly concerning please tell us becky
- Document 28: similarity score = 0.3888598208575087
help arrived sorry see trouble help hsb
- Document 81: similarity score = 0.27393194672695026
apologies sent dm help becky
- Document 11: similarity score = 0.2199622582877821
h definitely like work long experiencing issue aa
- Document 82: similarity score = 0.20150634193316508
tried work rude
- Document 81: similarity score = 0.19718909355578115
apologies sent dm help becky
- Document 61: similarity score = 0.27393194672695026
sorry please tell us help becky
- Document 28: similarity score = 0.24012822935206632
help arrived sorry see trouble help hsb
- Document 99: similarity score = 0.20986492269189536
get help already
- Document 28: similarity score = 0.2849917144592756
help arrived sorry see trouble help hsb
- Document 63: similarity score = 0.2286986084135601
hopefully get point becky
- Document 81: similarity score = 0.20986492269189536
apologies sent dm help becky
------------------------------------------------------------------------------
The documents after being ranked are
apologies sent dm help becky
get help already
h definitely like work long experiencing issue aa
help arrived sorry see trouble help hsb
hopefully get point becky
incredibly concerning please provide details investigate becky
incredibly concerning please tell us becky
sorry please tell us help becky
tried work rude
worst customer service
yo spectrum customer service reps super nice imma start trippin get service going
```