Information about data:

```
                    ->We have the amazon reviews dataset from kaggle

                    ->Reviews are given for the product

                    ->The features of the data were:

                          Id

                          ProductId- unique identifier for the product

                          UserId- unqiue identifier for the user

                          ProfileName

                          HelpfullnessNumerator- number of users who fou
nd the review helpful

                          HelpfulnessDenominator- number of users who in
dicated whether they
                          found the review  helpful or not

                          Score-rating between 1 and 5

                          Time-timestamp for the review

                          Summary- brief summary of the review

                          Text- text of the review

                    -> Based on the score of the review  we classify the
m into positive and
                       negative


                    Number of reviews: 568,454
```

Objective:

```
->Use of different techniques to convert the text data to vectors to
process by using Bag Of
  Words,TFIDF, Word2vec, Average word2vec
```

Loading the required libraries to process

In [1]:

```
import sqlite3
import numpy as np
import pandas as pd
import matplotlib.pyplot as mp
import seaborn as s
import nltk
import string
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_auc_score,auc
from nltk.stem.porter import PorterStemmer
```

Loading the data

In [2]:

```
con = sqlite3.connect("database.sqlite")
```

Filtering the reviews with positive and negative based on the score

In [3]:

```
filtereddata = pd.read_sql_query("""SELECT * FROM Reviews WHERE Score !=3""
",con)
```

Analysis about the given data:

```
->The shape of data

->Number of dimension of the data

->Number of attributes and their names

->Sample subset of data
```

In [4]:

```
print(filtereddata.shape)
print(filtereddata.ndim)
print(filtereddata.columns)
print(filtereddata.head(5))
```

```
(525814, 10)
2
Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
       'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text'],
      dtype='object')
   Id   ProductId          UserId                      ProfileName  \
0   1  B001E4KFG0  A3SGXH7AUHU8GW                       delmartian
1   2  B00813GRG4  A1D87F6ZCVE5NK                           dll pa
2   3  B000LQOCH0   ABXLMWJIXXAIN  Natalia Corres "Natalia Corres"
3   4  B000UA0QIQ  A395BORC6FGVXV                             Karl
```

```
4    5  B006K2ZZ7K  A1UQRSCLF8GW1T    Michael D. Bigham "M. Wassir"

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     1                       1      5  1303862400
1                     0                       0      1  1346976000
2                     1                       1      4  1219017600
3                     3                       3      2  1307923200
4                     0                       0      5  1350777600

                 Summary                                               Text
0  Good Quality Dog Food  I have bought several of the Vitality canned d...

1      Not as Advertised  Product arrived labeled as Jumbo Salted Peanut...
2  "Delight" says it all  This is a confection that has been around a fe...

3         Cough Medicine  If you are looking for the secret ingredient i...

4            Great taffy  Great taffy at a great price.  There was a wid...
```

->function to classify reviews into positive and negative based on rating.

->Here we are considering that reviews with a rating more than 3 are as positive and reviews with rating

->less than 3 as negative. So considering 3 as the neutral rating, so neglecting the reviews which are give-n

with rating of 3

In [5]:

```
def  partition(x):
    if x>3:
        return 'positive'
    return 'negative'
```

In [6]:

```
actualscore = filtereddata['Score']
posneg = actualscore.map(partition)
filtereddata['Score'] = posneg
```

In [7]:

```
filtereddata.head(5)
```

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDe
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDe
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0

In [8]:

```
display = pd.read_sql_query("""SELECT * FROM Reviews WHERE Score != 3 AND U
serId = "ABXLMWJIXXAIN" ORDER BY ProductId""",con)
```

In [9]:

```
display
```

Out[9]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessI
0	320691	B000CQ26E0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	0	0
1	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia	1	1

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessD
2	468954	B004DMGQKE	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	0	0

In [10]:

```
display1 = pd.read_sql_query("""SELECT * FROM Reviews WHERE Score !=3 AND UserId="AR5J8UI46CURR" ORDER BY ProductID """,con)
```

In [11]:

```
display1
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessD
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2

Exploratory data analysis

Deduplication:removing duplicates

In [12]:

```
sorteddata = filtereddata.sort_values('ProductId',axis=0,inplace=False,asce
nding=True,kind='quicksort',na_position='last')
```

In [13]:

```
sorteddata.head(6)
```

Out[13]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfu
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0
138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	1
138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1	1
138690	150508	0006641040	AZGXZ2UUK6X	Catherine Hallberg " (Kate)"	1	1
138691	150509	0006641040	A3CMRKGE0P909G	Teresa	3	4
138693	150511	0006641040	A1C9K534BCI9GO	Laura Purdie Salas	0	0

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfu

In [14]:

```
final =
sorteddata.drop_duplicates(subset=("UserId",'Time','Text','ProfileName'),ke
ep='first',inplace=False)
```

In [15]:

```
final.shape
```

Out[15]:

```
(364173, 10)
```

In [16]:

```
filtereddata.shape
```

Out[16]:

```
(525814, 10)
```

In [17]:

```
(final['Id'].size/filtereddata['Id'].size)*100
```

Out[17]:

```
69.25890143662969
```

->One more observation is that for a product the useful review(helpfullnessnumerator) is greater that the

->Total number of reviews on the product(helpfullnessdenominator) which is not possible

In [18]:

```
final = final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [19]:

```
final.shape
```

Out[19]:

```
(364171, 10)
```

Text preprocessing: ->Which includes cleaning the text like

```
->Removing special characters,stemming,lemitization

->Checking a word length greater that 2

->Removing stopwords

->converting words to lowercase
```

In [20]:

```
import re
i=100
for sent in final['Text'].values:
    if(len(re.findall("<.*?>",sent))):
        print(i)
        print(sent)
        break
    i =i+1
```

```
106
I set aside at least an hour each day to read to my son (3 y/o). At this po
int, I consider myself a connoisseur of children's books and this is one of
the best. Santa Clause put this under the tree. Since then, we've read it p
erpetually and he loves it.<br /><br />First, this book taught him the mont
hs of the year.<br /><br />Second, it's a pleasure to read. Well suited to
1.5 y/o old to 4+.<br /><br />Very few children's books are worth owning. M
ost should be borrowed from the library. This book, however, deserves a per
manent spot on your shelf. Sendak's best.
```

In [21]:

```
import string
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/vthumati/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[21]:

```
True
```

->Removing stop words

->Stemming the words

In [22]:

```
stop = set(stopwords.words('english'))
```

In [23]:

```
sno = nltk.stem.SnowballStemmer('english')
```

In [24]:

```
def cleanhtml(sentence):
    clean = re.compile("<.*?>")
    cleantext = re.sub(clean," ",sentence)
    return cleantext
```

In [25]:

```
def cleanpunct(sentence):
    cleaned = re.sub(r'[?|!|\|#|"]',r' ',sentence)
```

```
    cleaned = re.sub(r'[<|,|)|(|>|<]',r' ',cleaned)
    return cleaned
```

->List of stop words

In [26]:

```
print(stop)
```

```
{'how', 'an', 'with', 'off', 'me', 'under', 'each', 'too', 'it', "couldn't"
, 'our', 'he', 'yourself', 'your', 'her', "wouldn't", 'of', 'yours', 'these
', 'but', 'while', 'after', 'whom', 'shouldn', 'this', 'won', "she's", 'is'
, "hadn't", 'why', 'and', 'own', "don't", 'between', 'until', "you'll", 'du
ring', 'once', 'any', 'myself', 'up', 'were', "you're", "mightn't", 'before
', 'weren', 'ma', 'll', 'below', 'they', 'to', 'then', 'being', 'out', 'its
', "shouldn't", 'himself', 'most', 'd', 'are', 'my', 'will', 'both', 'all',
"needn't", 'shan', 'did', 'ourselves', 'no', 'over', 're', 'when', "didn't"
, 'have', 'she', 'from', "that'll", 'y', 'we', 'am', 'herself',
'themselves', 'hadn', 'you', "won't", 'there', 'if', "hasn't", "mustn't", '
the', 'just', 'doesn', 'by', "it's", 'needn', 'here', 'them', 'further', "y
ou've", 'itself', 'ain', 'such', 'don', 'where', 'than', "doesn't", 'other'
, 'mustn', 'had', 'him', 'isn', 'a', 't', "weren't", 'which', 'that', 'on',
"you'd", 'very', 'hers', 'down', 'above', "isn't", 'not', 'now', 'those', '
in', 'nor', 'their', 'ours', 'through', 'what', 'only', "shan't", 'his', 'm
ightn', 'can', 'does', 'so', 'do', 'as', 's', 've', 'at', 'hasn',
'yourselves', 'against', 'few', 'be', 'more', 'same', 'has', 'o', 'm', 'int
o', 'didn', "haven't", "should've", 'wasn', 'aren', 'for', 'couldn', 'doing
', 'about', 'should', 'i', "aren't", 'who', 'some', 'been', 'again', 'would
n', 'haven', 'having', 'was', 'or', 'because', 'theirs', "wasn't"}
```

->Sample example of stemming

In [27]:

```
sno.stem('congratulations')
```

Out[27]:

```
'congratul'
```

In [28]:

```
i=0
final_string=[]
all_positive_words=[]
all_negative_words=[]
s=''
str1=''
for sent in final['Text']:
    filtered_sentence=[]
    sent = cleanhtml(sent)
    for w in sent.split():
        for cleaned_word in cleanpunct(w).split():
            if((cleaned_word.isalpha())&(len(cleaned_word)>2)):
                if(cleaned_word.lower() not in stop):
                    s = (sno.stem(cleaned_word.lower())).encode('utf-8')
                    filtered_sentence.append(s)
                    if(final['Score'].values)[i]=='positive':
                        all_positive_words.append(s)
                    if(final['Score'].values)[i]=='negative':
```

```
                                all_negative_words.append(s)
    str1 = b" ".join(filtered_sentence)
    final_string.append(str1)
    i=i+1
```

->Storing the cleaned data for any future purpose

In [29]:

```
final['clearedtext'] = final_string
```

->Sample of Cleaned data

In [30]:

```
final.head(5)
```

Out[30]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfu
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0
138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	1
138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1	1
138690	150508	0006641040	AZGXZ2UUK6X	Catherine Hallberg " (Kate)"	1	1

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfu
138691	150509	0006641040	A3CMRKGE0P909G	Teresa	3	4

->Comapring a sample data point from original given data and cleaned data

In [31]:

```
print(final['Text'][100])
print(final['Text'].shape)
print(final['clearedtext'][100])
print(final['clearedtext'].shape)
```

```
I was diappointed in the flavor and texture of this mix.  I usually like mo
st of the Low Carb things I have tried, but was diappointed in this specifi
c one.
(364171,)
b'diappoint flavor textur usual like low carb thing tri diappoint specif'
(364171,)
```

In [32]:

```
conn = sqlite3.connect('final.sqlite')
c=conn.cursor()
conn.text_factory = str
final.to_sql('Reviews', conn, flavor=None, schema=None, if_exists='replace'
, index=True, index_label=None, chunksize=None, dtype=None)
```

BAG OF WORDS

In [33]:

```
count_vect = CountVectorizer()

count_vect_data = count_vect.fit_transform(final['Text'].values)
```

In [34]:

```
print(type(count_vect_data))
print(count_vect_data.get_shape())
```

```
<class 'scipy.sparse.csr.csr_matrix'>
(364171, 115281)
```

In [35]:

```
count_vect_feat = count_vect.get_feature_names()
```

In [36]:

```
print(count_vect_feat.index('hi'))
print(count_vect_feat.index('no'))
```

```
54335
73256
```

Observation:

```
-> The resultant sparse matrix has 115281 dimensions

-> The matrix which contains lot of sparce vectors is known as spars
e matrix

-> Sparse vector is a vector in which most of the element has a valu
e of 0

-> The funcionality of count vectorizer in bag of words is that it t
akes all the unique elements from the whole          text data

-> The total number of unique elements were 115281

-> It works in such a way that if a review contains 80 words it allo
ts a value to each word corresponding to the          dimensions

-> This means that for that review it only has 80 dimensions with va
lues and rest of them with a value of 0

-> If a word is repeated more than one time then its value is increm
ented by 1.
```

Bi-Gram and n-Grams

In [37]:

```python
from nltk import FreqDist
```

In [38]:

```python
frequent_positive_words = nltk.FreqDist(all_negative_words)
frequent_negative_words = nltk.FreqDist(all_positive_words)
```

In [39]:

```python
print("The most common negative words were:")
print(frequent_negative_words.most_common(20))
print("The most common positive words were:")
print(frequent_positive_words.most_common(20))
```

```
The most common negative words were:
[(b'like', 136163), (b'tast', 112920), (b'love', 105640), (b'use', 99605),
(b'good', 96789), (b'great', 93600), (b'one', 89285), (b'flavor', 87387), (
b'tri', 78859), (b'make', 73438), (b'product', 71581), (b'get', 70815), (b'
tea', 69505), (b'coffe', 65161), (b'would', 55170), (b'food', 52924), (b're
alli', 52343), (b'buy', 52226), (b'eat', 48356), (b'also', 46304)]
The most common positive words were:
[(b'like', 31494), (b'tast', 30340), (b'product', 22783), (b'one', 18565),
(b'would', 17804), (b'tri', 16711), (b'flavor', 15419), (b'use', 14636), (b
```

```
(b'would', 17604), (b'tri', 10711), (b'flavor', 15419), (b'use', 14050), (b
'get', 13503), (b'buy', 13318), (b'good', 12699), (b'coffe', 12163), (b'ord
er', 11923), (b'even', 10990), (b'food', 10615), (b'make', 9723), (b'tea',
9692), (b'realli', 9274), (b'box', 9152), (b'eat', 8883)]
```

Observations:

```
-> We can observe that there were some words which are common in bot
h positive and negative

-> This is because of preserving relationship with the consecutive w
ord, for example like is present in positive
  and not like present in negative, similarly taste is present in p
ositive and not taste is present in negative

-> Here comes the bi-grams which preserves the relationship with the
consecutive word
```

TF-IDF

In [40]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
```

In [41]:

```
final_tf_idf = tf_idf_vect.fit_transform(final['Text'].values)
```

In [42]:

```
final_tf_idf.get_shape()
```

Out[42]:

```
(364171, 2910192)
```

In [43]:

```
type(final_tf_idf)
```

Out[43]:

```
scipy.sparse.csr.csr_matrix
```

In [44]:

```
features=len(tf_idf_vect.get_feature_names())
```

In [45]:

```
features
```

Out[45]:

```
2910192
```

Observation:

```
-> TF is caluated by the number of times a word occur in a sentence
```

```
-> TF is calucated by the number of times a word occur in a sentence

-> IDF is calculated by the number of times a word present in the to
tal number of sentences in a document

-> We can observe that the dimensions were increased immensively fro
m Bag of Words to TF-IDF

-> In BOW we have 115k dimensions and in TF-IDF we have 2910k dimesn
ions which is massive increment in dimension

-> This is because we are preserving the relationship between the co
nsecutive words
```

WORD2VEC

In [46]:

```
import gensim
```

In [47]:

```
from gensim.models import word2vec
from gensim.models import keyedvectors
```

In [48]:

```
i=0
listofsent=[]
for sent in final['Text'].values:
    filtered_sentences = []
    sent = cleanhtml(sent)
    for w in sent.split():
        for cleanedwordws in cleanpunct(w).split():
            if(cleanedwordws.isalpha()):
                filtered_sentences.append(cleanedwordws.lower())
    listofsent.append(filtered_sentences)
```

In [49]:

```
print(final['Text'].values[5])
```

```
A charming, rhyming book that describes the circumstances under which you e
at (or don't) chicken soup with rice, month-by-month. This sounds like the
kind of thing kids would make up while they're out of recess and sing over
and over until they drive the teachers crazy. It's cute and catchy and soun
ds really childlike but is skillfully written.
```

In [50]:

```
print(type(listofsent))
print(len(listofsent))
```

```
<class 'list'>
364171
```

In [51]:

```
listofsent[5]
```

Out[51]:

```
['a',
 'charming',
 'rhyming',
 'book',
 'that',
 'describes',
 'the',
 'circumstances',
 'under',
 'which',
 'you',
 'eat',
 'or',
 'chicken',
 'soup',
 'with',
 'rice',
 'this',
 'sounds',
 'like',
 'the',
 'kind',
 'of',
 'thing',
 'kids',
 'would',
 'make',
 'up',
 'while',
 'out',
 'of',
 'recess',
 'and',
 'sing',
 'over',
 'and',
 'over',
 'until',
 'they',
 'drive',
 'the',
 'teachers',
 'cute',
 'and',
 'catchy',
 'and',
 'sounds',
 'really',
 'childlike',
 'but',
 'is',
 'skillfully']
```

In [52]:

```
w2vmodel = gensim.models.Word2Vec(listofsent,min_count=5,size=50,workers=4)
```

In [53]:

```
len(w2vmodel.wv.vocab)
```

Out[53]:

```
31022
```

In [54]:

```
w2vmodel.most_similar('like')
```

```
/Users/vthumati/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: DeprecationWarning: Call to deprecated `most_similar` (Method will be removed in 4.0.0, use self.wv.most_similar() instead).
  """Entry point for launching an IPython kernel.
```

Out[54]:

```
[('prefer', 0.6898671388626099),
 ('resemble', 0.6574873924255371),
 ('dislike', 0.6289805173873901),
 ('alright', 0.5870552659034729),
 ('enjoy', 0.5781151652336121),
 ('mean', 0.5774832367897034),
 ('resembles', 0.5741690397262573),
 ('weird', 0.5725610852241516),
 ('fake', 0.5698838233947754),
 ('ok', 0.5697323083877563)]
```

In [55]:

```
w2vmodel.similarity('tasty','yummy')
```

```
/Users/vthumati/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: DeprecationWarning: Call to deprecated `similarity` (Method will be removed in 4.0.0, use self.wv.similarity() instead).
  """Entry point for launching an IPython kernel.
```

Out[55]:

```
0.8272214968499794
```

In [56]:

```
w2vmodel.vector_size
```

Out[56]:

```
50
```

In [57]:

```
print(w2vmodel['hello'])
print(w2vmodel.wv['hello'])
```

```
[-1.1854096e-01  5.5081671e-01  3.2559234e-01  3.3430362e-01
  7.1865129e-01  5.6037359e-02  1.6931847e-01  8.7945241e-01
  5.3867942e-01 -8.5828739e-01  9.6814454e-02 -7.7230543e-02
  4.3830204e-01  2.7907360e-01  7.6603460e-01  2.7463341e-02
 -3.9858788e-02 -1.7495755e-02  5.9058481e-01 -4.9451190e-01
```

```
 -3.5494241e-01  1.2329574e-01 -6.1149496e-01  1.4991503e+00
  1.9391315e-01 -3.7106583e-03  7.7545553e-01  4.5713753e-01
 -3.1934062e-01 -6.1609995e-01 -8.0694772e-02  2.6215130e-01
  5.2422311e-02 -1.0260347e-01 -2.5115854e-01 -6.8862783e-04
 -2.8417426e-01 -1.9599552e-01  1.1006934e-01  4.0413913e-01
 -9.9617380e-01 -2.2122382e-01 -4.6280849e-01  2.0782246e-01
  9.2246878e-01 -3.7995791e-01  9.3426514e-01 -4.2408022e-01
  1.2737328e-01  3.5329875e-01]
[-1.1854096e-01  5.5081671e-01  3.2559234e-01  3.3430362e-01
  7.1865129e-01  5.6037359e-02  1.6931847e-01  8.7945241e-01
  5.3867942e-01 -8.5828739e-01  9.6814454e-02 -7.7230543e-02
  4.3830204e-01  2.7907360e-01  7.6603460e-01  2.7463341e-02
 -3.9858788e-02 -1.7495755e-02  5.9058481e-01 -4.9451190e-01
 -3.5494241e-01  1.2329574e-01 -6.1149496e-01  1.4991503e+00
  1.9391315e-01 -3.7106583e-03  7.7545553e-01  4.5713753e-01
 -3.1934062e-01 -6.1609995e-01 -8.0694772e-02  2.6215130e-01
  5.2422311e-02 -1.0260347e-01 -2.5115854e-01 -6.8862783e-04
 -2.8417426e-01 -1.9599552e-01  1.1006934e-01  4.0413913e-01
 -9.9617380e-01 -2.2122382e-01 -4.6280849e-01  2.0782246e-01
  9.2246878e-01 -3.7995791e-01  9.3426514e-01 -4.2408022e-01
  1.2737328e-01  3.5329875e-01]
```

```
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/ipykernel_launcher.py:1: DeprecationWarning: Call to deprecated `_
_getitem__` (Method will be removed in 4.0.0, use self.wv.__getitem__() ins
tead).
  """Entry point for launching an IPython kernel.
```

In [58]:

```
type(w2vmodel)
```

Out[58]:

```
gensim.models.word2vec.Word2Vec
```

Observation:

```
-> We can use the word2vec model trained by google in which eacg has
300-dimensions

->In order to load this into RAM we should have minimum 16gb of RAM

-> We can train our own model with the available data, in which we c
an specify the number of dimensions for each
   word

-> By using word2vec we can the similarity between the words and als
o most similar words for the given word

-> We get each word in the form of 50-dimension vector
representation
```

AVERAGE WORD2VEC

In [60]:

```
cnt=0
sent_vectors = []
for sent in listofsent:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        try:
            vec = w2vmodel.wv[word]
            sent_vec += vec
            cnt += 1
        except:
            pass
    sent_vec /= cnt
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[8]))
```

```
364171
50
```

In [61]:

```
len(sent_vectors)
```

Out[61]:

```
364171
```

In [62]:

```
sent_vectors[2]
```

Out[62]:

```
array([ 0.29711604, -0.07952446, -0.13827401,  0.12055589,  0.06898881,
       -0.02556613, -0.12011359, -0.18377848, -0.05153003,  0.02314267,
        0.00867077,  0.06547073, -0.27274745,  0.34108909, -0.15453824,
        0.03780076, -0.14235292,  0.12771546,  0.06324952, -0.22189752,
        0.21001871,  0.07057208,  0.03940238, -0.11574526, -0.20464066,
       -0.04807252,  0.08293264,  0.01046649, -0.05196467,  0.01703962,
       -0.09501723, -0.17810969,  0.10342992, -0.02257037,  0.02479422,
       -0.18104209,  0.21668923,  0.04378558, -0.16756496, -0.0722719 ,
       -0.04385533,  0.11049835,  0.27700842,  0.10832192, -0.01456592,
        0.0746142 , -0.06094364, -0.15741823,  0.14786104,  0.02934933])
```

In [63]:

```
type(sent_vectors)
```

Out[63]:

```
list
```

In [65]:

```
len(sent_vectors[99999])
```

Out[65]:

```
50
```

Observation:

```
-> Using word2vec we got the vector representation of each word

-> In average word2vec we can get the vector represenation of each s
entence by using word2vec for each word

-> Compute the average of all the word2vec representation of the wor
ds to its length
```

TF-IDF WORD2VEC

In [68]:

```
tf_idf_features = tf_idf_vect.get_feature_names()
```

In [69]:

```
len(tf_idf_features)
```

Out[69]:

```
2910192
```

In [88]:

```
tfidf_sent_vec = []
row=0
for sent in listofsent:
        sent_vector = np.zeros(50)
        sum =0
        for word in sent:
            try:
                vec = w2vmodel.wv[word]
                tf_idf = final_tf_idf[row, tfidf_features.index(word)]
                sent_vec += (vec * tf_idf)
                sum += tf_idf
            except:
                pass
        sent_vec /= sum
        tfidf_sent_vec.append(sent_vector)
        row += 1
```

In [81]:

```
len(tfidf_sent_vec)
```

Out[81]:

```
364171
```

In [90]:

```
print(final_tf_idf[0,tf_idf_features.index('in')])
print(w2vmodel.wv['in'])
```

```
a = final_tf_idf[0,tf_idf_features.index('in')]
b = w2vmodel.wv['in']
c = a*b
print(c)
```

```
0.03277183177760655
[-1.3472861  -2.6658442  -2.2370605   0.4774281  -2.928193   -0.32357216
 -2.0251245  -0.9878434  -2.2339106  -0.03716455  2.5197098   2.881545
  0.4819802  -0.8746339  -0.6758463  -2.0316405   1.6041346   0.46214825
  0.46068484 -1.0548087  -0.36071628 -1.2633047   1.890839    0.2935755
 -2.4268842   2.0213099  -0.66063696 -2.0583522   2.3773334   2.7423663
 -0.66039616 -0.02048293  3.2774856   1.198115    1.521976   -0.60031295
  1.9289638   0.98776215  1.1197149  -1.6213804  -2.0004926   3.123306
  3.642648    2.2052634   1.2420686   0.87117904  0.844115    0.36920503
 -0.40260243  0.16546378]
[-0.04415303 -0.0873646  -0.07331257  0.0156462  -0.09596226 -0.01060405
 -0.06636705 -0.03237344 -0.07320935 -0.00121795  0.08257551  0.09443352
  0.01579537 -0.02866336 -0.02214872 -0.06658059  0.05257043  0.01514545
  0.01509749 -0.03456802 -0.01182133 -0.04140081  0.06196626  0.00962101
 -0.07953344  0.06624203 -0.02165028 -0.06745598  0.07790957  0.08987238
 -0.02164239 -0.00067126  0.10740921  0.03926443  0.04987795 -0.01967335
  0.06321568  0.03237078  0.03669511 -0.05313561 -0.06555981  0.10235646
  0.11937625  0.07227052  0.04070487  0.02855013  0.0276632   0.01209953
 -0.01319402  0.00542255]
```

Note:

```
-> It works in such a way that it constructs word2vec for each word 
in a sentence and will get the tf-idf value
   of the same word from tf-idf vectorizer

-> It will do the product of both the values and to that value it wi
ll do average with total tf-idf values of that sentence
```

In [95]:

```
print(tf_idf_features.index('in'))
print(w2vmodel.wv['in'])
print(tfidf_sent_vec[999])
```

```
1266669
[-1.3472861  -2.6658442  -2.2370605   0.4774281  -2.928193   -0.32357216
 -2.0251245  -0.9878434  -2.2339106  -0.03716455  2.5197098   2.881545
  0.4819802  -0.8746339  -0.6758463  -2.0316405   1.6041346   0.46214825
  0.46068484 -1.0548087  -0.36071628 -1.2633047   1.890839    0.2935755
 -2.4268842   2.0213099  -0.66063696 -2.0583522   2.3773334   2.7423663
 -0.66039616 -0.02048293  3.2774856   1.198115    1.521976   -0.60031295
  1.9289638   0.98776215  1.1197149  -1.6213804  -2.0004926   3.123306
  3.642648    2.2052634   1.2420686   0.87117904  0.844115    0.36920503
 -0.40260243  0.16546378]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
```

Observation:

```
-> The alternate strategy to construct sentence vectors is TFIDF-WOR
D2vec
```

```
-> It computes the tfidf vector for text and then it computes the wo
rd2vec for the text

-> The result is the average of product of tfidf vector and word2vec
vector to the tfidf vector
```

CONCLUSION:

```
-> Based on the business requirement we can do pre-preocessing like
removing stop words,
  stemming of words and protecting lemitization

-> We have seen different approaches to convert the text to vectors
using techniques like Bag
   of Words, TF-IDF, WORD2VEC, AVERAGE WORD2VEC, TF-IDF-WORD2VEC
```