Amazon Fine Food Reviews Analysis

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454
Number of users: 256,059
Number of products: 74,258
Timespan: Oct 1999 - Oct 2012
Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Objective:

Given a review, determine whether the review is positive (Rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative?

[Ans] We could use the Score/Rating. A rating of 4 or 5 could be cosnidered a positive review. A review of 1 or 2 could be considered negative. A review of 3 is nuetral and ignored. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score id above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")



import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
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_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os
```

[1]. Reading Data

In [2]:

```
# using the SQLite Table to read data.
con = sqlite3.connect('database.sqlite')
#filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data points
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000""", co
n)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 Limit 30000""", con)

# Give reviews with Score>3 a positive rating, and reviews with a score<3 a negative rating.
def partition(x):
    if x < 3:
        return 0
    return 1

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
filtered_data.shape[0]
```

```
Number of data points in our data (30000, 10)
```

Out[2]:

```
30000
```

In [3]:

```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [4]:

```
print(display.shape)
display.head()
```

(80668, 7)

Out[4]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ET0	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B007Y59HVM	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ET0	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBE1U	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [5]:

```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	5	I was recommended to try green tea extract to ...	5

In [7]:

```
display['COUNT(*)'].sum()
```

Out[7]:

393063

Exploratory Data Analysis

[2] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

In [8]:

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

Out[8]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995776

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(

As can be seen above the same user has multiple reviews of the with the same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that

ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8)

ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

In [9]:

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')
```

In [10]:

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final.shape
```

Out[10]:

```
(28072, 10)
```

In [11]:

```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[11]:

```
93.57333333333332
```

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than

HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

In [12]:

```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[12]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883

In [13]:

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

In [14]:

```
#Before starting the next phase of preprocessing lets see the number of entries left
print(final.shape)

#How many positive and negative reviews are present in our dataset?
final['Score'].value_counts()
```

```
(28072, 10)
```

Out[14]:

```
1    23606
0     4466
Name: Score, dtype: int64
```

[3]. Text Preprocessing.

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

In [15]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [16]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "y
ou're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [17]:

```
# Combining all the above stundents
from tqdm import tqdm
from bs4 import BeautifulSoup

preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
```

```
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████████████████████████| 28072/28072 [00:13<00:00, 2069.60it/s]
```

[3.2] Preprocess Summary

In [18]:

```
## Similartly you can do preprocessing for review summary also.

from tqdm import tqdm
preprocessed_summary = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Summary'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwords)
    preprocessed_summary.append(sentance.strip())
```

```
 74%|██████████████████████        | 20799/28072 [00:07<00:02, 
2994.23it/s]C:\ProgramData\Anaconda3\lib\site-packages\bs4\__init__.py:219: UserWarning: "b'...'" 
looks like a filename, not markup. You should probably open this file and pass the filehandle into 
Beautiful Soup.
  ' Beautiful Soup.' % markup)
100%|██████████████████████████████| 28072/28072 [00:09<00:00, 2844.46it/s]
```

In [19]:

```
final.shape
```

Out[19]:

```
(28072, 10)
```

[4] Featurization

[4.1] BAG OF WORDS

In [20]:

```
#BoW

data_pos = final[final["Score"] == 1].sample(n = 1200)
data_neg = final[final["Score"] == 0].sample(n = 1200)
final_2400 = pd.concat([data_pos, data_neg])
final_2400.shape
# count_vect = CountVectorizer() #in scikit-learn
# count_vect.fit(preprocessed_reviews)
# print("some feature names ", count_vect.get_feature_names()[:10])
# print('='*50)

# final_counts = count_vect.transform(preprocessed_reviews)
# print("the type of count vectorizer ",type(final_counts))
# print("the shape of out text BOW vectorizer ",final_counts.get_shape())
# print("the number of unique words ", final_counts.get_shape()[1])
```

Out[20]:

```
(2400, 10)
```

[4.2] Bi-Grams and n-Grams.

In [21]:

```
#bi-gram, tri-gram and n-gram

#removing stop words like "not" should be avoided before building n-grams
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
# you can choose these numebrs min_df=10, max_features=5000, of your choice
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=2400)
final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_bigram_counts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (28072, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[4.3] TF-IDF

In [94]:

```
# tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
# tf_idf_vect.fit(preprocessed_reviews)
# print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names()[0:10])
# print('='*50)

# final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
# print("the type of count vectorizer ",type(final_tf_idf))
# print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
# print("the number of unique words including both unigrams and bigrams ", final_tf_idf.get_shape()[1])

tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
final_tf_idf = tf_idf_vect.fit_transform(final_2400['Text'].values)
```

[4.4] Word2Vec

In [143]:

```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews_TFIDFW2V:
    list_of_sentance.append(sentance.split())
```

In [144]:

```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
```

```
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=Tr
ue)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your
own w2v ")
```

```
WARNING:gensim.models.base_any2vec:consider setting layer size to a multiple of 4 for greater perf
ormance
```

```
[('wonderful', 0.9985113143920898), ('excellent', 0.9983680248260498), ('definitely',
0.9982941150665283), ('tasting', 0.9982283711433411), ('annie', 0.9981883764266968), ('regular', 0
.9981613755226135), ('alternative', 0.9981475472450256), ('pretty', 0.9981463551521301),
('looking', 0.9981260299682617), ('overall', 0.9981017112731934)]
==================================================
[('mother', 0.9995893239974976), ('web', 0.9995521306991577), ('sent', 0.9995428323745728), ('us',
0.999521017074585), ('main', 0.9994969367980957), ('plants', 0.9994878172874451), ('loved',
0.9994723796844482), ('half', 0.9994656443595886), ('packages', 0.9994639158248901),
('completely', 0.9994621872901917)]
```

In [145]:

```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  3485
sample words  ['dogs', 'love', 'saw', 'pet', 'store', 'tag', 'regarding', 'made', 'china',
'satisfied', 'safe', 'loves', 'chicken', 'product', 'wont', 'buying', 'anymore', 'hard', 'find', '
products', 'usa', 'one', 'isnt', 'bad', 'good', 'take', 'chances', 'till', 'know', 'going',
'available', 'course', 'total', 'pretty', 'stinky', 'right', 'nearby', 'used', 'ca', 'not', 'beat'
, 'great', 'received', 'shipment', 'could', 'hardly', 'wait', 'try', 'call', 'instead']
```

[4.4.1] Converting text into vectors using wAvg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

In [146]:

```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this
to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
  0%|          | 0/4000 [00:00<?, ?it/s]
  1%|▌         | 58/4000 [00:00<00:06, 574.20it/s]
  3%|█▏        | 126/4000 [00:00<00:06, 602.30it/s]
  5%|█▊        | 194/4000 [00:00<00:06, 623.66it/s]
  7%|██▊       | 291/4000 [00:00<00:05, 698.46it/s]
 10%|███▋      | 386/4000 [00:00<00:04, 758.71it/s]
 12%|████▎     | 460/4000 [00:00<00:04, 735.03it/s]
 13%|█████     | 535/4000 [00:00<00:04, 739.43it/s]
 15%|█████▊    | 607/4000 [00:00<00:04, 685.30it/s]
 17%|██████▍   | 675/4000 [00:00<00:04, 677.55it/s]
 19%|███████   | 742/4000 [00:01<00:05, 583.50it/s]
 20%|███████▋  | 815/4000 [00:01<00:05, 617.71it/s]
 23%|████████▌ | 907/4000 [00:01<00:04, 683.71it/s]
 26%|█████████▋| 1024/4000 [00:01<00:03, 781.09it/s]
 28%|██████████▌| 1110/4000 [00:01<00:03, 738.99it/s]
```

```
30%|██████████                        | 1190/4000 [00:01<00:04, 680.95it/s]
32%|██████████▋                       | 1263/4000 [00:01<00:04, 584.71it/s]
33%|███████████▎                      | 1328/4000 [00:02<00:05, 499.92it/s]
35%|███████████▊                      | 1385/4000 [00:02<00:05, 502.57it/s]
36%|████████████▍                     | 1460/4000 [00:02<00:04, 557.76it/s]
38%|████████████▉                     | 1521/4000 [00:02<00:04, 559.84it/s]
40%|█████████████▍                    | 1581/4000 [00:02<00:04, 552.36it/s]
42%|██████████████▍                   | 1692/4000 [00:02<00:03, 649.23it/s]
45%|███████████████▏                  | 1786/4000 [00:02<00:03, 715.63it/s]
47%|███████████████▊                  | 1866/4000 [00:02<00:03, 694.70it/s]
49%|████████████████▌                 | 1942/4000 [00:02<00:02, 701.22it/s]
50%|█████████████████▏                | 2017/4000 [00:02<00:02, 693.33it/s]
52%|█████████████████▊                | 2090/4000 [00:03<00:03, 621.24it/s]
54%|██████████████████▎               | 2158/4000 [00:03<00:02, 637.76it/s]
56%|██████████████████▉               | 2232/4000 [00:03<00:02, 665.32it/s]
58%|███████████████████▌              | 2301/4000 [00:03<00:02, 640.67it/s]
```

```
59%|██████████████████████                  | 2375/4000 [00:03<00:02, 663.94it/s]

61%|██████████████████████                  | 2443/4000 [00:03<00:02, 647.64it/s]

63%|███████████████████████                 | 2509/4000 [00:03<00:02, 585.42it/s]

64%|████████████████████████                | 2570/4000 [00:03<00:02, 533.48it/s]

66%|████████████████████████                | 2636/4000 [00:04<00:02, 566.02it/s]

67%|█████████████████████████               | 2695/4000 [00:04<00:02, 542.93it/s]

69%|█████████████████████████               | 2751/4000 [00:04<00:02, 499.53it/s]

71%|██████████████████████████              | 2826/4000 [00:04<00:02, 546.64it/s]

72%|███████████████████████████             | 2886/4000 [00:04<00:01, 560.04it/s]

74%|███████████████████████████             | 2948/4000 [00:04<00:01, 575.15it/s]

75%|████████████████████████████            | 3007/4000 [00:04<00:01, 533.88it/s]

77%|████████████████████████████            | 3062/4000 [00:04<00:01, 524.73it/s]

78%|█████████████████████████████           | 3129/4000 [00:04<00:01, 559.81it/s]

80%|█████████████████████████████           | 3187/4000 [00:05<00:01, 564.05it/s]

81%|██████████████████████████████          | 3245/4000 [00:05<00:01, 560.48it/s]
```

```
 83%|██████████████████████████████      | 3310/4000 [00:05<00:01, 581.48it/s]

 84%|███████████████████████████████     | 3371/4000 [00:05<00:01, 588.03it/s]

 86%|███████████████████████████████▌    | 3444/4000 [00:05<00:00, 622.85it/s]

 88%|████████████████████████████████    | 3510/4000 [00:05<00:00, 633.53it/s]

 90%|█████████████████████████████████   | 3580/4000 [00:05<00:00, 652.09it/s]

 92%|█████████████████████████████████▌  | 3662/4000 [00:05<00:00, 694.75it/s]

 93%|██████████████████████████████████  | 3733/4000 [00:05<00:00, 651.14it/s]

 95%|███████████████████████████████████ | 3803/4000 [00:05<00:00, 665.04it/s]

 97%|███████████████████████████████████▌| 3871/4000 [00:06<00:00, 650.24it/s]

 98%|████████████████████████████████████| 3937/4000 [00:06<00:00, 588.47it/s]

100%|████████████████████████████████████▌| 3998/4000 [00:06<00:00, 540.95it/s]

100%|█████████████████████████████████████| 4000/4000 [00:06<00:00, 627.59it/s]
```

```
4000
50
```

[4.4.1.2] TFIDF weighted W2v

In [140]:

```
preprocessed_reviews_TFIDFW2V = preprocessed_reviews[: 4000]
```

In [147]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
model.fit(preprocessed_reviews_TFIDFW2V)
```

```
# # we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [148]:

```
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
  0%|                                        | 0/4000 [00:00<?, ?it/s]

  0%|▏                                  | 20/4000 [00:00<00:21, 188.70it/s]

  1%|▎                                  | 31/4000 [00:00<00:26, 148.44it/s]

  1%|▍                                   | 38/4000 [00:00<00:47, 83.63it/s]

  1%|▍                                   | 45/4000 [00:00<00:51, 77.19it/s]

  1%|▌                                   | 54/4000 [00:00<00:49, 79.99it/s]

  2%|▌                                   | 61/4000 [00:00<00:53, 73.80it/s]

  2%|▋                                   | 78/4000 [00:00<00:44, 87.51it/s]

  2%|▊                                   | 88/4000 [00:00<00:51, 76.35it/s]
```

```
2%|█                                                    | 97/4000 [00:01<00:51, 75.94it/s]

3%|█                                                   | 109/4000 [00:01<00:45, 85.16it/s]

3%|█▏                                                  | 119/4000 [00:01<00:46, 83.55it/s]

3%|█▎                                                  | 129/4000 [00:01<00:44, 86.97it/s]

4%|█▎                                                  | 141/4000 [00:01<00:40, 94.58it/s]

4%|█▍                                                  | 151/4000 [00:01<00:41, 92.41it/s]

4%|█▌                                                  | 162/4000 [00:01<00:41, 91.50it/s]

4%|█▋                                                 | 176/4000 [00:01<00:37, 101.23it/s]

5%|█▊                                                 | 187/4000 [00:01<00:37, 101.14it/s]

5%|█▉                                                 | 199/4000 [00:02<00:37, 102.34it/s]

5%|██                                                 | 217/4000 [00:02<00:32, 116.65it/s]

6%|██▎                                                | 240/4000 [00:02<00:27, 136.64it/s]

6%|██▍                                                | 258/4000 [00:02<00:25, 146.21it/s]

7%|██▌                                                | 275/4000 [00:02<00:24, 152.21it/s]

7%|██▊                                                | 292/4000 [00:02<00:24, 150.48it/s]
```

```
  8%|███                                      | 312/4000 [00:02<00:23, 159.44it/s]

  8%|███                                      | 335/4000 [00:02<00:20, 174.81it/s]

  9%|███                                      | 354/4000 [00:02<00:20, 176.12it/s]

  9%|████                                     | 376/4000 [00:03<00:19, 183.13it/s]

 10%|████                                     | 397/4000 [00:03<00:19, 186.88it/s]

 11%|████                                     | 423/4000 [00:03<00:17, 203.15it/s]

 11%|████                                     | 445/4000 [00:03<00:18, 196.78it/s]

 12%|████                                     | 466/4000 [00:03<00:20, 169.90it/s]

 12%|█████                                    | 485/4000 [00:03<00:20, 173.55it/s]

 13%|█████                                    | 504/4000 [00:03<00:20, 169.60it/s]

 13%|█████                                    | 522/4000 [00:03<00:22, 158.05it/s]

 13%|█████                                    | 539/4000 [00:04<00:22, 155.26it/s]

 14%|█████                                    | 557/4000 [00:04<00:21, 160.64it/s]

 15%|██████                                   | 583/4000 [00:04<00:18, 181.07it/s]

 15%|██████                                   | 603/4000 [00:04<00:29, 114.61it/s]

 16%|██████                                   | 623/4000 [00:04<00:25, 131.19it/s]
```

```
16%|██████                                     | 640/4000 [00:04<00:26, 126.67it/s]
16%|██████▏                                    | 659/4000 [00:04<00:23, 140.44it/s]
17%|██████▎                                    | 676/4000 [00:05<00:23, 140.81it/s]
17%|██████▍                                     | 692/4000 [00:05<00:39, 84.11it/s]
18%|██████▌                                     | 705/4000 [00:05<00:36, 90.92it/s]
18%|██████▌                                    | 720/4000 [00:05<00:32, 100.21it/s]
18%|███████                                     | 733/4000 [00:05<00:32, 99.23it/s]
19%|███████                                    | 752/4000 [00:05<00:28, 115.20it/s]
19%|███████▏                                   | 775/4000 [00:05<00:23, 135.25it/s]
20%|███████▌                                   | 795/4000 [00:06<00:21, 149.47it/s]
21%|███████▊                                   | 821/4000 [00:06<00:18, 168.98it/s]
21%|████████                                   | 845/4000 [00:06<00:17, 181.25it/s]
22%|████████▏                                  | 866/4000 [00:06<00:17, 183.57it/s]
22%|████████▍                                  | 886/4000 [00:06<00:17, 177.21it/s]
23%|████████▌                                  | 905/4000 [00:06<00:19, 159.06it/s]
```

```
23%|█████████▏                              | 922/4000 [00:06<00:20, 153.83it/s]
24%|█████████▍                              | 944/4000 [00:06<00:18, 168.32it/s]
24%|█████████▋                              | 969/4000 [00:06<00:16, 186.20it/s]
25%|█████████▉                              | 989/4000 [00:07<00:17, 172.88it/s]
25%|██████████                             | 1012/4000 [00:07<00:16, 186.35it/s]
26%|██████████▎                            | 1032/4000 [00:07<00:16, 183.95it/s]
26%|██████████▌                            | 1052/4000 [00:07<00:16, 175.59it/s]
27%|██████████▋                            | 1071/4000 [00:07<00:20, 142.81it/s]
27%|██████████▊                             | 1087/4000 [00:07<00:31, 91.89it/s]
28%|███████████                             | 1100/4000 [00:08<00:30, 95.22it/s]
28%|███████████▏                           | 1119/4000 [00:08<00:25, 111.59it/s]
29%|███████████▍                           | 1143/4000 [00:08<00:21, 132.70it/s]
29%|███████████▌                           | 1160/4000 [00:08<00:27, 102.87it/s]
29%|███████████▊                           | 1179/4000 [00:08<00:24, 116.21it/s]
30%|███████████▉                           | 1198/4000 [00:08<00:21, 127.57it/s]
```

```
30%|███████████▎                          | 1214/4000 [00:08<00:24, 114.29it/s]
31%|███████████▌                          | 1231/4000 [00:09<00:21, 126.47it/s]
31%|███████████▋                          | 1253/4000 [00:09<00:19, 143.55it/s]
32%|███████████▊                          | 1270/4000 [00:09<00:20, 135.46it/s]
32%|███████████▉                          | 1295/4000 [00:09<00:17, 153.30it/s]
33%|████████████                          | 1313/4000 [00:09<00:17, 156.26it/s]
33%|████████████▏                         | 1330/4000 [00:09<00:18, 146.11it/s]
34%|████████████▏                         | 1348/4000 [00:09<00:17, 153.28it/s]
34%|████████████▎                         | 1369/4000 [00:09<00:15, 166.40it/s]
35%|████████████▎                         | 1387/4000 [00:09<00:15, 165.57it/s]
35%|████████████▍                         | 1405/4000 [00:10<00:15, 165.90it/s]
36%|█████████████                         | 1423/4000 [00:10<00:15, 169.42it/s]
36%|█████████████▏                        | 1441/4000 [00:10<00:15, 161.34it/s]
37%|█████████████▍                        | 1462/4000 [00:10<00:14, 172.97it/s]
37%|█████████████▌                        | 1480/4000 [00:10<00:15, 163.57it/s]
37%|█████████████▋                        | 1497/4000 [00:10<00:16, 154.20it/s]
```

```
38%|█████████████▋                      | 1513/4000 [00:10<00:16, 153.65it/s]
38%|█████████████▉                      | 1540/4000 [00:10<00:13, 176.12it/s]
39%|██████████████                      | 1560/4000 [00:10<00:13, 178.28it/s]
40%|██████████████▏                     | 1580/4000 [00:11<00:13, 177.89it/s]
40%|██████████████▍                     | 1599/4000 [00:11<00:14, 164.42it/s]
40%|██████████████▌                     | 1617/4000 [00:11<00:14, 163.74it/s]
41%|██████████████▊                     | 1647/4000 [00:11<00:12, 189.21it/s]
42%|███████████████                     | 1668/4000 [00:11<00:13, 178.14it/s]
42%|███████████████▏                    | 1688/4000 [00:11<00:17, 134.70it/s]
43%|███████████████▎                    | 1705/4000 [00:11<00:16, 137.05it/s]
43%|███████████████▌                    | 1722/4000 [00:12<00:15, 142.58it/s]
43%|███████████████▋                    | 1739/4000 [00:12<00:15, 149.44it/s]
44%|███████████████▊                    | 1755/4000 [00:12<00:15, 140.79it/s]
44%|███████████████▉                    | 1770/4000 [00:12<00:15, 142.21it/s]
45%|████████████████                    | 1785/4000 [00:12<00:16, 137.71it/s]
```

```
45%|████████████████▌          | 1800/4000 [00:12<00:15, 140.39it/s]

45%|████████████████▊          | 1815/4000 [00:12<00:18, 120.11it/s]

46%|████████████████▉          | 1833/4000 [00:12<00:18, 120.32it/s]

46%|█████████████████           | 1846/4000 [00:13<00:23, 89.89it/s]

47%|█████████████████          | 1861/4000 [00:13<00:20, 101.96it/s]

47%|█████████████████          | 1873/4000 [00:13<00:20, 104.28it/s]

47%|█████████████████▎         | 1887/4000 [00:13<00:19, 107.46it/s]

48%|█████████████████▌         | 1906/4000 [00:13<00:17, 123.09it/s]

48%|█████████████████▋         | 1924/4000 [00:13<00:15, 133.27it/s]

48%|█████████████████▊         | 1939/4000 [00:13<00:19, 107.62it/s]

49%|██████████████████          | 1952/4000 [00:14<00:23, 86.70it/s]

49%|██████████████████▍         | 1963/4000 [00:14<00:22, 92.12it/s]

49%|██████████████████         | 1979/4000 [00:14<00:19, 104.52it/s]

50%|██████████████████▊         | 1991/4000 [00:14<00:20, 98.55it/s]

50%|██████████████████         | 2005/4000 [00:14<00:18, 107.91it/s]

51%|██████████████████▍        | 2025/4000 [00:14<00:15, 123.58it/s]
```

```
51%|██████████████████▊                  | 2039/4000 [00:14<00:22, 85.58it/s]
51%|███████████████████                  | 2056/4000 [00:15<00:19, 100.03it/s]
52%|███████████████████▏                 | 2073/4000 [00:15<00:17, 111.87it/s]
52%|███████████████████▎                 | 2092/4000 [00:15<00:14, 127.36it/s]
53%|███████████████████▍                 | 2108/4000 [00:15<00:14, 128.17it/s]
53%|███████████████████▋                 | 2123/4000 [00:15<00:16, 112.85it/s]
53%|███████████████████▊                 | 2136/4000 [00:15<00:16, 110.61it/s]
54%|███████████████████▉                 | 2149/4000 [00:15<00:16, 113.67it/s]
54%|███████████████████▉                 | 2162/4000 [00:15<00:17, 103.71it/s]
54%|████████████████████                 | 2174/4000 [00:16<00:17, 101.53it/s]
55%|████████████████████▎                | 2193/4000 [00:16<00:15, 114.60it/s]
55%|████████████████████▍                | 2206/4000 [00:16<00:15, 112.36it/s]
55%|████████████████████▌                | 2219/4000 [00:16<00:15, 116.81it/s]
56%|████████████████████▋                | 2240/4000 [00:16<00:13, 132.46it/s]
56%|████████████████████▉                | 2258/4000 [00:16<00:12, 143.17it/s]
```

```
57%|█████████████████████                | 2274/4000 [00:16<00:12, 133.75it/s]
57%|█████████████████████▏               | 2289/4000 [00:16<00:13, 126.37it/s]
58%|█████████████████████▎               | 2306/4000 [00:16<00:12, 136.59it/s]
58%|█████████████████████▌               | 2327/4000 [00:17<00:11, 151.27it/s]
59%|█████████████████████▋               | 2347/4000 [00:17<00:10, 162.41it/s]
59%|█████████████████████▉               | 2370/4000 [00:17<00:09, 177.70it/s]
60%|██████████████████████               | 2390/4000 [00:17<00:08, 182.35it/s]
60%|██████████████████████▎              | 2413/4000 [00:17<00:08, 192.49it/s]
61%|██████████████████████▌              | 2433/4000 [00:17<00:08, 191.88it/s]
61%|██████████████████████▋              | 2453/4000 [00:17<00:08, 188.76it/s]
62%|██████████████████████▉              | 2473/4000 [00:17<00:08, 173.51it/s]
62%|███████████████████████              | 2491/4000 [00:17<00:08, 170.43it/s]
63%|███████████████████████▏             | 2509/4000 [00:18<00:09, 152.51it/s]
63%|███████████████████████▎             | 2525/4000 [00:18<00:09, 154.24it/s]
64%|███████████████████████▌             | 2541/4000 [00:18<00:09, 152.36it/s]
```

```
64%|███████████████████████▋             | 2557/4000 [00:18<00:11, 127.89it/s]
64%|███████████████████████▊             | 2573/4000 [00:18<00:10, 134.72it/s]
65%|███████████████████████▉             | 2593/4000 [00:18<00:09, 147.36it/s]
65%|████████████████████████▏            | 2609/4000 [00:18<00:09, 142.10it/s]
66%|████████████████████████▎            | 2632/4000 [00:18<00:08, 157.86it/s]
66%|████████████████████████▌            | 2652/4000 [00:18<00:08, 167.25it/s]
67%|████████████████████████▋            | 2670/4000 [00:19<00:08, 150.71it/s]
67%|████████████████████████▊            | 2686/4000 [00:19<00:08, 151.64it/s]
68%|████████████████████████▉            | 2702/4000 [00:19<00:09, 143.30it/s]
68%|█████████████████████████▏           | 2717/4000 [00:19<00:09, 132.55it/s]
68%|█████████████████████████▎           | 2731/4000 [00:19<00:10, 115.67it/s]
69%|█████████████████████████▍           | 2744/4000 [00:19<00:10, 114.57it/s]
69%|█████████████████████████▌           | 2766/4000 [00:19<00:09, 128.87it/s]
70%|█████████████████████████▋           | 2783/4000 [00:19<00:08, 138.62it/s]
70%|█████████████████████████▉           | 2802/4000 [00:20<00:07, 150.15it/s]
71%|██████████████████████████▏          | 2825/4000 [00:20<00:07, 167.24it/s]
```

```
71%|██████████████████████████▎          | 2843/4000 [00:20<00:07, 159.10it/s]

72%|██████████████████████████▍          | 2860/4000 [00:20<00:07, 147.45it/s]

72%|██████████████████████████▌          | 2876/4000 [00:20<00:07, 145.65it/s]

72%|██████████████████████████▊          | 2894/4000 [00:20<00:07, 152.92it/s]

73%|███████████████████████████          | 2919/4000 [00:20<00:06, 172.37it/s]

73%|███████████████████████████▏         | 2938/4000 [00:20<00:06, 171.55it/s]

74%|███████████████████████████▎         | 2956/4000 [00:21<00:07, 147.49it/s]

74%|███████████████████████████▍         | 2972/4000 [00:21<00:06, 150.61it/s]

75%|███████████████████████████▋         | 2988/4000 [00:21<00:06, 146.98it/s]

75%|███████████████████████████▊         | 3006/4000 [00:21<00:06, 153.95it/s]

76%|███████████████████████████▉         | 3024/4000 [00:21<00:06, 159.65it/s]

76%|████████████████████████████▏        | 3041/4000 [00:21<00:06, 152.15it/s]

76%|████████████████████████████▎        | 3057/4000 [00:21<00:06, 145.58it/s]

77%|████████████████████████████▍        | 3074/4000 [00:21<00:06, 147.77it/s]

77%|████████████████████████████▋        | 3098/4000 [00:21<00:05, 165.65it/s]
```

```
78%|████████████████████████████▊        | 3120/4000 [00:22<00:04, 177.61it/s]

78%|█████████████████████████████        | 3139/4000 [00:22<00:05, 168.17it/s]

79%|█████████████████████████████▎       | 3167/4000 [00:22<00:04, 190.68it/s]

80%|█████████████████████████████▍       | 3188/4000 [00:22<00:04, 175.90it/s]

80%|█████████████████████████████▋       | 3207/4000 [00:22<00:04, 176.40it/s]

81%|█████████████████████████████▊       | 3226/4000 [00:22<00:04, 173.84it/s]

81%|██████████████████████████████       | 3245/4000 [00:22<00:04, 164.93it/s]

82%|██████████████████████████████▏      | 3265/4000 [00:22<00:04, 170.09it/s]

82%|██████████████████████████████▍      | 3289/4000 [00:22<00:03, 184.24it/s]

83%|██████████████████████████████▌      | 3309/4000 [00:23<00:04, 161.31it/s]

83%|██████████████████████████████▊      | 3327/4000 [00:23<00:04, 137.19it/s]

84%|██████████████████████████████▉      | 3346/4000 [00:23<00:04, 146.22it/s]

84%|███████████████████████████████▏     | 3368/4000 [00:23<00:03, 159.40it/s]

85%|███████████████████████████████▎     | 3386/4000 [00:23<00:03, 161.95it/s]

85%|███████████████████████████████▌     | 3407/4000 [00:23<00:03, 172.61it/s]

86%|███████████████████████████████▋     | 3429/4000 [00:23<00:03, 184.07it/s]
```

```
86%|████████████████████████████████▍    | 3451/4000 [00:23<00:02, 192.04it/s]

87%|████████████████████████████████▌    | 3472/4000 [00:24<00:02, 196.00it/s]

87%|████████████████████████████████▋    | 3494/4000 [00:24<00:02, 200.97it/s]

88%|████████████████████████████████▉    | 3515/4000 [00:24<00:02, 202.43it/s]

89%|█████████████████████████████████▏   | 3542/4000 [00:24<00:02, 217.80it/s]

89%|█████████████████████████████████▎   | 3565/4000 [00:24<00:02, 192.93it/s]

90%|█████████████████████████████████▌   | 3589/4000 [00:24<00:02, 203.43it/s]

90%|█████████████████████████████████▊   | 3615/4000 [00:24<00:01, 216.02it/s]

91%|██████████████████████████████████   | 3638/4000 [00:24<00:01, 218.78it/s]

92%|██████████████████████████████████▎  | 3663/4000 [00:24<00:01, 221.27it/s]

92%|██████████████████████████████████▍  | 3686/4000 [00:25<00:01, 208.03it/s]

93%|██████████████████████████████████▋  | 3708/4000 [00:25<00:01, 198.87it/s]

93%|██████████████████████████████████▉  | 3729/4000 [00:25<00:01, 186.48it/s]

94%|███████████████████████████████████  | 3750/4000 [00:25<00:01, 192.44it/s]

94%|███████████████████████████████████▎ | 3770/4000 [00:25<00:01, 187.01it/s]
```

```
 95%|██████████████████████████████████  | 3793/4000 [00:25<00:01, 196.10it/s]

 95%|██████████████████████████████████▍ | 3818/4000 [00:25<00:00, 205.53it/s]

 96%|██████████████████████████████████▌ | 3839/4000 [00:25<00:00, 177.49it/s]

 96%|██████████████████████████████████▋ | 3858/4000 [00:25<00:00, 178.02it/s]

 97%|██████████████████████████████████▊ | 3878/4000 [00:26<00:00, 183.08it/s]

 97%|███████████████████████████████████ | 3897/4000 [00:26<00:00, 150.03it/s]

 98%|███████████████████████████████████ | 3914/4000 [00:26<00:00, 138.76it/s]

 98%|███████████████████████████████████▏| 3932/4000 [00:26<00:00, 147.54it/s]

 99%|███████████████████████████████████▌| 3948/4000 [00:26<00:00, 150.65it/s]

 99%|███████████████████████████████████▋| 3964/4000 [00:26<00:00, 135.43it/s]

 99%|███████████████████████████████████▊| 3979/4000 [00:26<00:00, 133.90it/s]

100%|███████████████████████████████████▉| 3996/4000 [00:26<00:00, 141.94it/s]

100%|████████████████████████████████████| 4000/4000 [00:26<00:00, 148.17it/s]
```

In [114]:

```
len(tfidf_sent_vectors)
```

Out[114]:

```
0
```

[5] Applying TSNE

1. you need to plot 4 tsne plots with each of these feature set
    A. Review text, preprocessed one converted into vectors using (BOW)
    B. Review text, preprocessed one converted into vectors using (TFIDF)
    C. Review text, preprocessed one converted into vectors using (AVG W2v)
    D. Review text, preprocessed one converted into vectors using (TFIDF W2v)
2. Note 1: The TSNE accepts only dense matrices
3. Note 2: Consider only 5k to 6k data points

[5.1] Applying TNSE on Text BOW vectors

In [21]:

```
score_2400 = final_2400["Score"]
score_2400.shape
final_2400.head()
```

Out[21]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
20927	22856	B000RZAJL8	AHHN2Q0GEOI9U	D. McCarthy "SciFiMan"	1	1	1	11
14901	16251	B007TJGZ54	A39QRS8WVWAZHY	Doranjmm	0	0	1	13
19618	21378	B002QWP89S	A3BVSVB8I7GU3K	JD 068	0	0	1	13
4886	5304	B004XMIRU6	A1EYRQ4AN2LCZL	William B. Edwards	1	1	1	13
12889	14065	B0045XE32E	A261Q5U692JF17	BeachBrights "beachbrights blogspot"	2	4	1	12

In [22]:

```
count_vect = CountVectorizer() #in scikit-learn
final_counts = count_vect.fit_transform(final_2400['Text'].values)
```

In [23]:

```
type(final_counts)
final_counts.get_shape()
```

Out[23]:

```
(2400, 10342)
```

In [24]:

```
count_vect = CountVectorizer(ngram_range=(1,2))
final_bigram_counts = count_vect.fit_transform(final_2400['Text'].values)
```

```
final_bigram_counts.get_shape()
```

Out[24]:

```
(2400, 97346)
```

In [25]:

```
from sklearn.preprocessing import StandardScaler

std_data = StandardScaler(with_mean = False).fit_transform(final_bigram_counts)
std_data.shape
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning:
Data with input dtype int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning:
Data with input dtype int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

Out[25]:

```
(2400, 97346)
```

In [26]:

```
type(std_data)
```

Out[26]:

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

In [27]:

```
std_data = std_data.todense()
```

In [28]:

```
type(std_data)
```

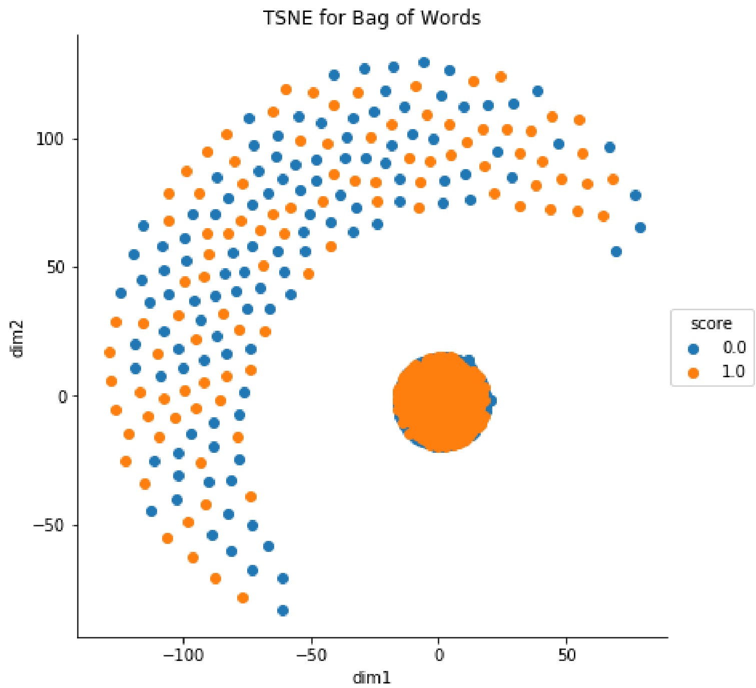
Out[28]:

```
numpy.matrixlib.defmatrix.matrix
```

In [29]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label

import numpy as np
from sklearn.manifold import TSNE
from sklearn import datasets
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sn

#TSNE

model = TSNE(n_components=2, random_state=0, perplexity = 30, n_iter = 2400)

tsne_data = model.fit_transform(std_data)

# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, score_2400)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("dim1", "dim2", "score"))

# Ploting the result of tsne
```

```
sns.FacetGrid(tsne_df, hue="score", size=6).map(plt.scatter, 'dim1', 'dim2').add_legend()
plt.title("TSNE for Bag of Words")
plt.show()
```



Obervations :

->here i have taken 1200 postive and 1200 negative data points and then i am standardzing the data. then appling TSNE for BOW. -> from the TSNE plots , all the positive and negative reviews are overlapped with each other

[5.1] Applying TNSE on Text TFIDF vectors

In [48]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

In [120]:

```
#TFIDF

tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
final_tf_idf = tf_idf_vect.fit_transform(final_2400['Text'].values)
```

In [121]:

```
# Standardization
from sklearn.preprocessing import StandardScaler
std = StandardScaler(with_mean = False)
std_data = std.fit_transform(final_tf_idf)
```

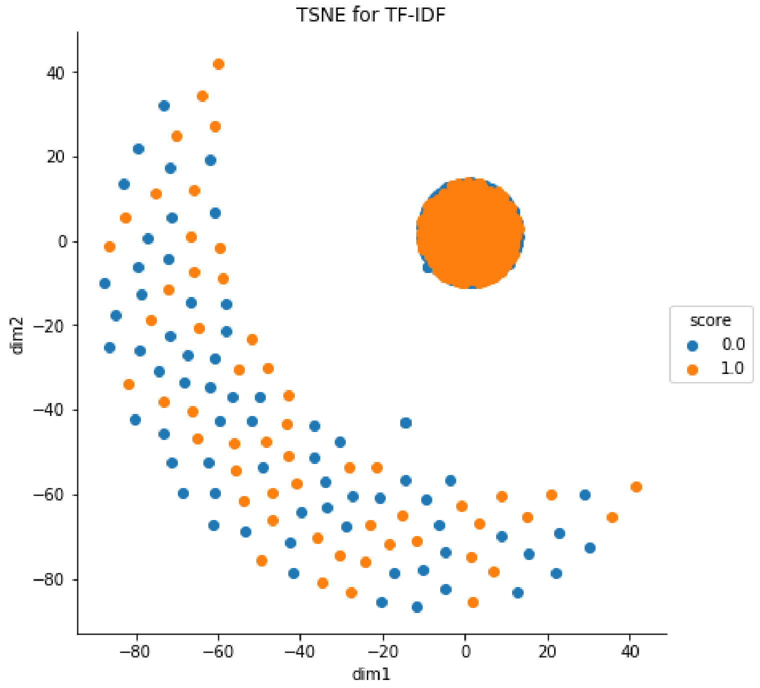
In [122]:

```
std_data = std_data.todense()
```

In [33]:

```
# tsne
from sklearn.manifold import TSNE
model = TSNE(n_components = 2, perplexity = 50)
```

```
tsne_data = model.fit_transform(std_data)

tsne_data = np.vstack((tsne_data.T, score_2400)).T
tsne_df = pd.DataFrame(data = tsne_data, columns = ("dim1", "dim2", "score"))
sns.FacetGrid(tsne_df, hue = "score", size = 6).map(plt.scatter, "dim1", "dim2").add_legend()
plt.title("TSNE for TF-IDF")
plt.show()
```



In [82]:

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

Out[82]:

```
97346
```

In [86]:

```
print(final_tf_idf[3,:].toarray()[0])
```

```
[0. 0. 0. ... 0. 0. 0.]
```

In [87]:

```
# source: https://buhrmann.github.io/tfidf-analysis.html
def top_tfidf_feats(row, features, top_n=25):
    ''' Get top n tfidf values in row and return them with their corresponding feature names.'''
    topn_ids = np.argsort(row)[::-1][:top_n]
    top_feats = [(features[i], row[i]) for i in topn_ids]
    df = pd.DataFrame(top_feats)
    df.columns = ['feature', 'tfidf']
    return df

top_tfidf = top_tfidf_feats(final_tf_idf[1,:].toarray()[0],features,25)
```

Observations :

This one also looks same as like BOW, i.e both postive and negative reviews are over lapped each other.

[5.3] Applying TNSE on Text Avg W2V vectors

In [0]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

In [46]:

```
import pickle
def savetofile(obj,filename):
    pickle.dump(obj,open(filename+".p","wb"), protocol=4)
savetofile(sent_vectors,"avg_w2v_vec")
```

In [48]:

```
#Loading the variable from file
def openfromfile(filename):
    temp = pickle.load(open(filename+".p","rb"))
    return temp
avg_vec = openfromfile("avg_w2v_vec")
```

In [52]:

```
avg_vec = np.array(avg_vec)
avg_vec.shape[0]
```

Out[52]:

```
28072
```

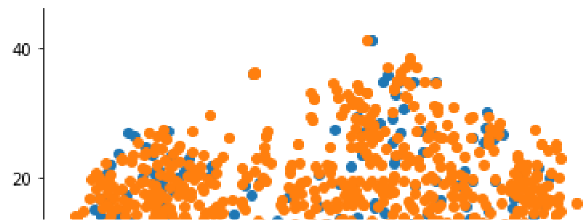
In [62]:

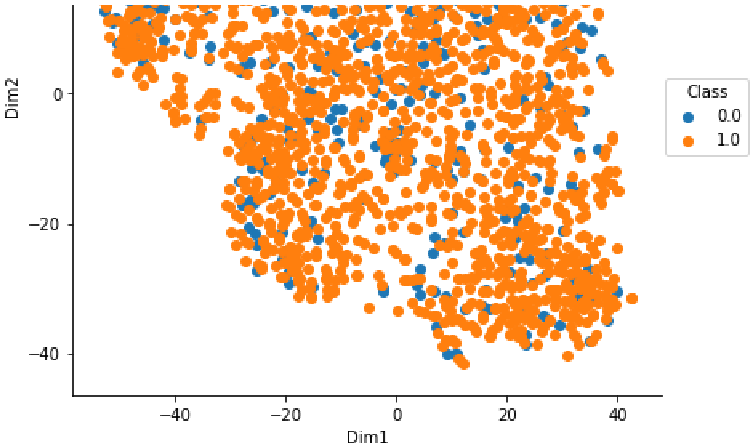
```
%%time
from sklearn.manifold import TSNE
from time import time
import random

n_samples = 2000
sample_cols = random.sample(range(1, avg_vec.shape[0]), n_samples)
sample_features = avg_vec[sample_cols]
# sample_features = df
sample_class = final['Score'][sample_cols]
sample_class = sample_class[:,np.newaxis]
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=30)

embedded_data = model.fit_transform(sample_features)
# print(embedded_data.shape,sample_class.shape)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])
```

```
(2000, 50) (2000, 1)
(2000, 3)
Wall time: 1min 1s
```

In [63]:

```
sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```





Here to same as above like BOW and TFIDF , which are not well separated

[5.4] Applying TNSE on Text TFIDF weighted W2V vectors

In [0]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

In [ ]:

```
len(tfidf_sent_vectors)
```

In [150]:

```
data_pos1 = final[final["Score"] == 1].sample(n = 2000)
data_neg1 = final[final["Score"] == 0].sample(n = 2000)
final_4000 = pd.concat([data_pos1, data_neg1])
final_4000.shape
```

Out[150]:

```
(4000, 10)
```

In [151]:

```
score_4000 = final_4000["Score"]
score_4000.shape
final_4000.head()
```

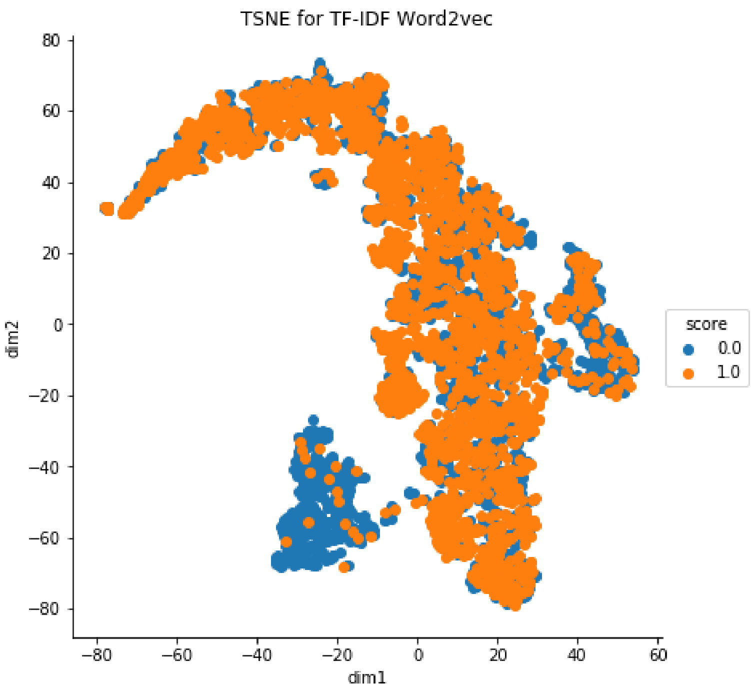
Out[151]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
5147	5582	B000G1X45G	A2FRFAQCWZJT3Q	B. Davis "The Happy Hermit"	6	7	1	119
15434	16877	B001LGGH40	A2IO1ESNSIAXG3	L. A. Kane	1	1	1	123
18934	20658	B00066CRRM	A16S7LO0ZS1CDU	June M. Selzer	0	0	1	132

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
5740	6216	B000NY31I6	A2C2KNOC2FF3J2	N. Bohan	1	1	1	127
13710	14963	B000UVKZXQ	A2GEXXITA54RN	Chelsea	1	1	1	133

In [152]:

```
#tsne
from sklearn.manifold import TSNE
model = TSNE(n_components=2, random_state=0, perplexity = 50, n_iter = 5000)

tsne_data = model.fit_transform(tfidf_sent_vectors)

tsne_data = np.vstack((tsne_data.T, score_4000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("dim1", "dim2", "score"))

# Ploting the result of tsne
sns.FacetGrid(tsne_df, hue="score", size=6).map(plt.scatter, 'dim1', 'dim2').add_legend()
plt.title("TSNE for TF-IDF Word2vec")
plt.show()
```



Here i have taken 4000 data points to see better plots and from the above plot we can see slightly some positive points are separated.

[6] Conclusions

In [0]:

```
# Write few sentance about the results that you got and observation that you did from the analysis
For BOW and TFIDF , i have 2400 points as the tnse plot is taking too long time.

From above all tsne plots ,negative and positive data points are over lapped with each other.
So as per the above plots we cannot make any assumptions which are not separable.
so we need to do other models to make some assumptions.
for TFIDF W2V , there is slightly plots which are showing that those are separated.
```

```
References:

    I have refered with some of Github  , kaggle sites and taken some code from provided .ipy noteb
ooks in the course which is modified.
```