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).

[1]. Reading Data

[1.1] Loading the data The dataset is available in two forms

.csv file SQLite Database In order to load the data, We have used the SQLITE dataset as it is 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 is above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [2]:
```
%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
```

```python
from nltk.stem.wordnet import WordNetLemmatizer

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

from tqdm import tqdm
import os
```

```
C:\Users\Excel\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWa
rning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_seria
l")
```

In [3]:

```python
# REFERED FROM APPLIED AI COURSE VIDEOS
# using SQLite Table to read data.
con = sqlite3.connect('C:/Users/Excel/Desktop/sqlite/amazon-fine-food-r
eviews/sqlite 1/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 50
0000 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 Sco
re != 3 LIMIT 500000""", con)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score
 != 3 LIMIT 5000""", con)

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

```python
#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)
```

```
Number of data points in our data (5000, 10)
```

Out[3]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1

[2.1] 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 [4]:
```
display=pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score !=3 and userID='AR5J8UI46CURR'
ORDER BY ProductID
""",con)
```

In [5]:
```
print(display.shape)
display.head()
```

```
(5, 10)
```

Out[5]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2

As it can be seen above that same user has multiple reviews with 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 [6]:
```
#Sorting the data according to productID in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0,ascending=True)
```

In [7]:
```
# drop the duplicate data
final=sorted_data.drop_duplicates(subset={'UserId',"ProfileName","Time","Text"},keep="first",inplace=False)
```

In [8]:
```
final.shape
```

Out[8]:
```
(4986, 10)
```

In [9]:
```
#checking to see how much % of data is still remaining
(final["Id"].size*1.0)/(filtered_data["Id"].size*1.0)*100
```

Out[9]:
```
99.72
```

In [10]:
```
# we know that helpfullness numerator is always LESSTHAN EQUALTO  helpfulness of denominator
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
print(final.shape)
final["Score"].value_counts()
```

```
(4986, 10)
```

Out[10]:
```
positive    4178
negative     808
Name: Score, dtype: int64
```

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

[3] Preprocessing

[3.1]. Preprocessing Review Text Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

1. Hence in the Preprocessing phase we do the following in the order below:-
2. Begin by removing the html tags
3. Remove any punctuations or limited set of special characters like , or . or # etc.
4. Check if the word is made up of english letters and is not alpha-numeric
5. 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)
6. Convert the word to lowercase
7. Remove Stopwords
8. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)
9. After which we collect the words used to describe positive and negative reviews

In [11]:

```
#refered from applied ai course videos
import re
i=0;
for sent in final['Text'].values:
    if (len(re.findall('<.*?>', sent))):
        print(i)
        print(sent)
        break;
    i += 1;
```

```
0
Why is this $[...] when the same product is available for $[...] here?<
br />http://www.amazon.com/VICTOR-FLY-MAGNET-BAIT-REFILL/dp/B00004RBDY<
br /><br />The Victor M380 and M502 traps are unreal, of course -- tota
l fly genocide. Pretty stinky, but only right nearby.
```

In [12]:
```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Excel\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[12]: True

In [13]:
```
#refered from applied AI Course
import re

import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

stop = set(stopwords.words('english')) #set of stopwords
sno = nltk.stem.SnowballStemmer('english') #initialising the snowball s
temmer

def cleanhtml(sentence): #function to clean the word of any html-tags
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext
def cleanpunc(sentence): #function to clean the word of any punctuation
 or special characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
print(stop)
print('************************************')
print(sno.stem('tasty'))
```

```
{'their', 'weren', 're', 'but', 'what', "you're", 'hadn', 'his', 'need
n', 'as', 'very', 'an', 'how', "doesn't", 'nor', 'won', 'haven', "might
n't", 'be', 'himself', 'are', 'been', "couldn't", 'our', 'couldn', 'ow
n', "weren't", "needn't", 'both', 'should', 'any', 'isn', 't', 'yoursel
ves', 'about', 'to', 'ours', "aren't", 'when', 'than', 'have', 'your',
'with', 'me', 'further', 'o', 'down', 'for', 'more', 'doesn', "shan't",
```

```
'at', 'yourself', 'once', 'only', 'off', "you'd", 'did', 'through', 'v
e', 'during', 'hers', 'in', 'by', 'y', 'because', "hadn't", 'against',
'no', 's', 'do', 'doing', "should've", 'some', 'wasn', 'they', 'themsel
ves', 'shouldn', 'wouldn', 'd', "you'll", 'its', 'until', 'we', 'ther
e', 'the', 'after', 'why', 'who', "won't", 'here', "it's", 'of', 'whic
h', 'over', 'myself', 'him', 'herself', 'these', 'don', 'before', "have
n't", 'such', "shouldn't", 'or', 'being', "you've", 'he', 'yours', 'the
m', 'has', 'just', 'now', 'all', "didn't", 'that', 'most', 'and', 'm',
"isn't", 'didn', 'is', 'hasn', 'will', 'while', 'so', 'few', 'shan', 'm
ightn', 'was', 'ourselves', 'not', 'each', "that'll", 'her', 'ain', 'ot
her', 'a', 'had', 'you', 'into', 'were', 'my', 'll', 'from', 'between',
'where', 'i', 'those', 'am', 'ma', "mustn't", "wasn't", 'then', 'does',
'same', 'whom', 'mustn', 'on', "she's", 'above', "don't", "wouldn't",
'under', 'too', "hasn't", 'aren', 'itself', 'can', 'she', 'up', 'this',
'again', 'having', 'below', 'out', 'if', 'it', 'theirs'}
************************************
tasti
```

In [14]:

```
#refer from applied ai course videos
i=0
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
for sent in final['Text'].values:
    filtered_sentence=[]
    #print(sent);
    sent=cleanhtml(sent) # remove HTMl tags
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                if(cleaned_words.lower() not in stop):
                    s=(sno.stem(cleaned_words.lower())).encode('utf8')
                    filtered_sentence.append(s)
                    if (final['Score'].values)[i] == 'positive':
                        all_positive_words.append(s) #list of all words
 used to describe positive reviews
                    if(final['Score'].values)[i] == 'negative':
```

```
                            all_negative_words.append(s) #list of all words
 used to describe negative reviews reviews
                        else:
                            continue
                else:
                    continue
    #print(filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    #print("***********************************************************
************")

    final_string.append(str1)
    i+=1
```

In [15]:
```
final['preprocessedtext']=final_string
```

In [16]:
```
final.head(3) #below the processed review can be seen in the CleanedTex
t Column


# store final table into an SQlLite table for future.
conn = sqlite3.connect('final.sqlite')
c=conn.cursor()
conn.text_factory = str
final.to_sql('Reviews', conn, schema=None, if_exists='replace', index=T
rue, index_label=None, chunksize=None, dtype=None)
```

Bag of Words

NOTE: WE USE ONLY UNIGRAM AND WE DONT USE THE BIGRAM BECAUSE THE ISSUE OF MEMORY

In [17]:
```
# this all xi's set
final.shape
```

Out[17]:
```
(4986, 11)
```

In [18]:
```
# we taken yi's as score which is called as label
score=final['Score']
```

In [19]:
```
score.shape
```

Out[19]:
```
(4986,)
```

In [20]:
```
#fitting the data set of xi's (preprocssed review into bag of words)
#we are top maximum features (1000)
count_vect=CountVectorizer(max_features=1000)
final_counts=count_vect.fit_transform(final['preprocessedtext'].values)
```

In [21]:
```
type(final_counts) # data is in sparse so tsne will not able work on th
is sparse
```

Out[21]:
```
scipy.sparse.csr.csr_matrix
```

In [22]:
```
final_counts.get_shape()
```

Out[22]:
```
(4986, 1000)
```

In [23]:
```
# standardised the data
from sklearn.preprocessing import StandardScaler

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

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

Out[23]: (4986, 1000)

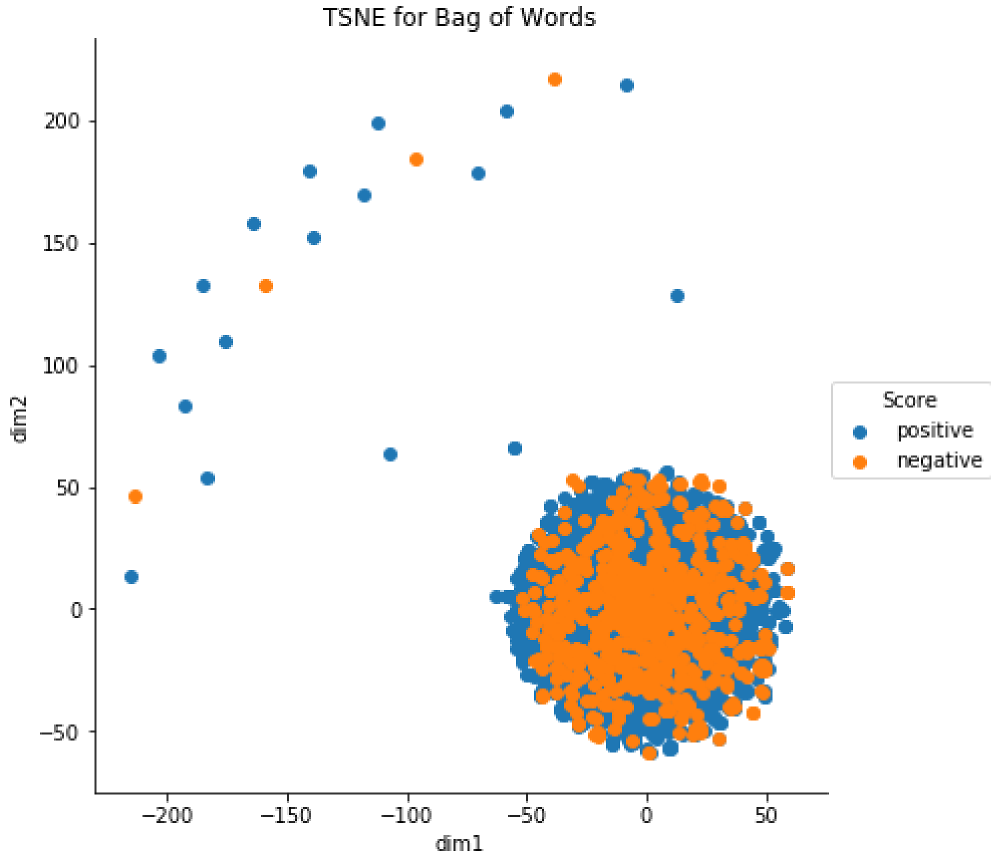
In [24]:
```
std_data = std_data.todense()  # we should convert sparse to dense beca
use its sparse doesnot work with TSNE
```

In [25]:
```
type(std_data)
```

Out[25]: numpy.matrixlib.defmatrix.matrix

TSNE for Bag of Words

In [28]:
```
from sklearn.manifold import TSNE
model = TSNE(n_components=2, random_state=0, perplexity = 30, n_iter =
5000)
# configuring the parameteres
# the number of components = 2
# default learning rate = 200


tsne_data = model.fit_transform(std_data)


# creating a new data frame for plotting the dataset visualization
tsne_data = np.vstack((tsne_data.T, score)).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', 'd
im2').add_legend()
plt.title("TSNE for Bag of Words")
plt.show()
```



Observation :

1. Hence the here the reviews are not linearly separable so we can't apply hyperplane to separate it.
2. There is alternate way to separate this reviews by feature transforming or feature enggineering.

TF-IDF

NOTE: WE USE ONLY UNIGRAM AND WE DONT USE THE BIGRAM BECAUSE THE ISSUE OF MEMORY

In [26]:
```
# we are taking top maximum features
#fitting data into tfidf vectorizer
tf_idf_vect=TfidfVectorizer(max_features=1000)
final_tf_idf=tf_idf_vect.fit_transform(final['preprocessedtext'].values
)

final_tf_idf.get_shape()
```

Out[26]: (4986, 1000)

In [27]:
```
#standarizing the preprocessed review
from sklearn.preprocessing import StandardScaler

std_data_tf = StandardScaler(with_mean = False).fit_transform(final_tf_
idf)
std_data_tf.shape
```

Out[27]: (4986, 1000)

In [28]:
```
type(std_data_tf)# data is in sparse so tsne will not able work on this
 sparse
```

Out[28]: scipy.sparse.csr.csr_matrix

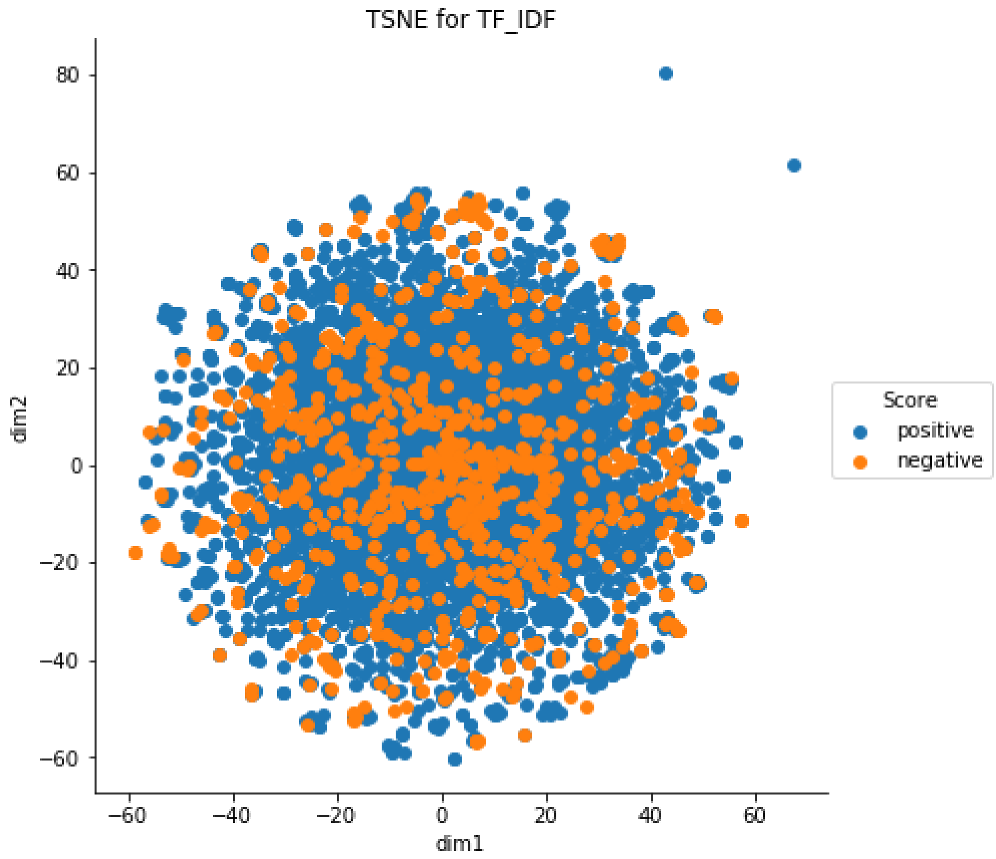
In [29]:
```
std_data_tf=std_data_tf.todense()# we should convert sparse to dense be
cause its sparse doesnot work with TSNE
```

In [30]:
```
type(std_data_tf)
```

Out[30]: numpy.matrixlib.defmatrix.matrix

In [28]:
```
#applying the TSNE for TF-IDF
from sklearn.manifold import TSNE

model = TSNE() # Here i have used default value even for iteration(100
```

```
0) because of run time complexity
# configuring the parameteres
# the number of components = 2
# default perplexity = 30
# default learning rate = 200

tsne_data_tf = model.fit_transform(std_data_tf)


# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data_tf.T, score)).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', 'd
im2').add_legend()
plt.title("TSNE for TF_IDF")
plt.show()
```



Observation :

1. Hence the here the reviews are not linearly separable so we can't apply hyperplane to separate it.
2. There is alternate way to separate this reviews by feature transforming or feature enggineering.

WORD2VEC

In [31]:
```
#refer from APPLIED AI COURSE VIDEOS
from gensim.models import word2vec
from gensim.models import keyedvectors
import pickle
```

In [32]:
```
#refered from Applied AI course videos
# Train your own Word2Vec model using your own text corpus
import gensim
list_of_sent = []
for sent in final['preprocessedtext'].values:
    filtered_sentence = []
    sent = cleanhtml('sent')
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if(cleaned_words.isalpha()):
                filtered_sentence.append(cleaned_words.lower())
            else:
                continue
    list_of_sent.append(filtered_sentence)
print(final['preprocessedtext'].values[0])
print("*****************************************************************
*")
print(list_of_sent[0])
```

```
b'product avail www amazon com victor trap unreal cours total fli genoc
id pretti stinki right nearbi'
*****************************************************************
['sent']
```

In [33]:
```
w2v_model=gensim.models.Word2Vec(list_of_sent,min_count=5,size=50, work
ers=4)
```

In [34]:
```
w2v = w2v_model[w2v_model.wv.vocab]
```

In [35]:
```
w2v.shape
```

Out[35]:
```
(1, 50)
```

In [36]:
```
words = list(w2v_model.wv.vocab)
print(len(words))
```

```
1
```

AVG W2V

In [37]:
```
#refered from applied AI Course videos
sent_vectors = []; # the avg-w2v for each sentence/review is stored in
 this list
for sent in list_of_sent: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence
    for word in sent: # for each word in a review/sentence
        try:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
        except:
            pass
    sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
4986
50
```

In [38]:
```
type(sent_vectors)
```

Out[38]:
```
list
```

In [39]:
```
from sklearn.preprocessing import StandardScaler

std_data_vec = StandardScaler(with_mean = False).fit_transform(sent_vec
tors)
std_data_vec.shape
```

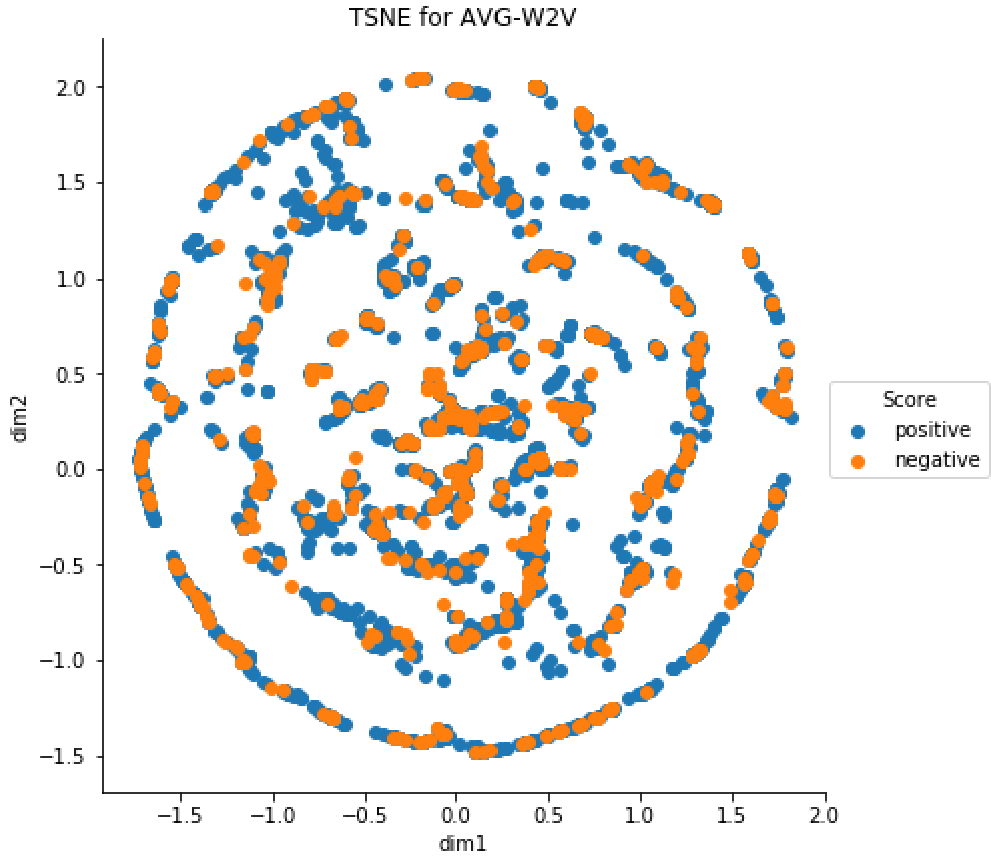
Out[39]: (4986, 50)

In [40]:
```
type(std_data_vec)
```

Out[40]: numpy.ndarray

In [42]:
```
from sklearn.manifold import TSNE
```

In [44]:
```
model=TSNE(n_iter=5000)
tsne_data=model.fit_transform(std_data_vec)
tsne_data=np.vstack((tsne_data.T,score)).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 AVG-W2V')
plt.show()
```



Observation :

1. Hence the here the reviews are like coil shape and its not linearly separable. So we can't apply hyperplane to separate it.
2. There is alternate way to separate this reviews by feature transforming or feature enggineering.

TF-IDF W2V

In [73]:
```
#refered from applied ai course videos
tfidf_feat = tf_idf_vect.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and ce
ll_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is st
ored in this list
row=0;
for sent in list_of_sent: # 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/r
eview
    for word in sent: # for each word in a review/sentence
        try:
            vec = w2v_model.wv[word]
            # obtain the tf_idfidf of a word in a sentence/review
            tf_idf = final_tf_idf[row, tfidf_feat.index(word)]
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
        except:
            pass
    sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

In [75]:
```
type(tfidf_sent_vectors)
```

Out[75]: list

In [76]:
```
from sklearn.preprocessing import StandardScaler

std_data_tfw2v = StandardScaler(with_mean = False).fit_transform(final_
tf_idf)
std_data_tfw2v.shape
```

Out[76]: (4986, 1000)

In [77]:
```
type(std_data_tfw2v) # its in the form of sparse so we need to convert
 into dense for the performnace of TSNE
```

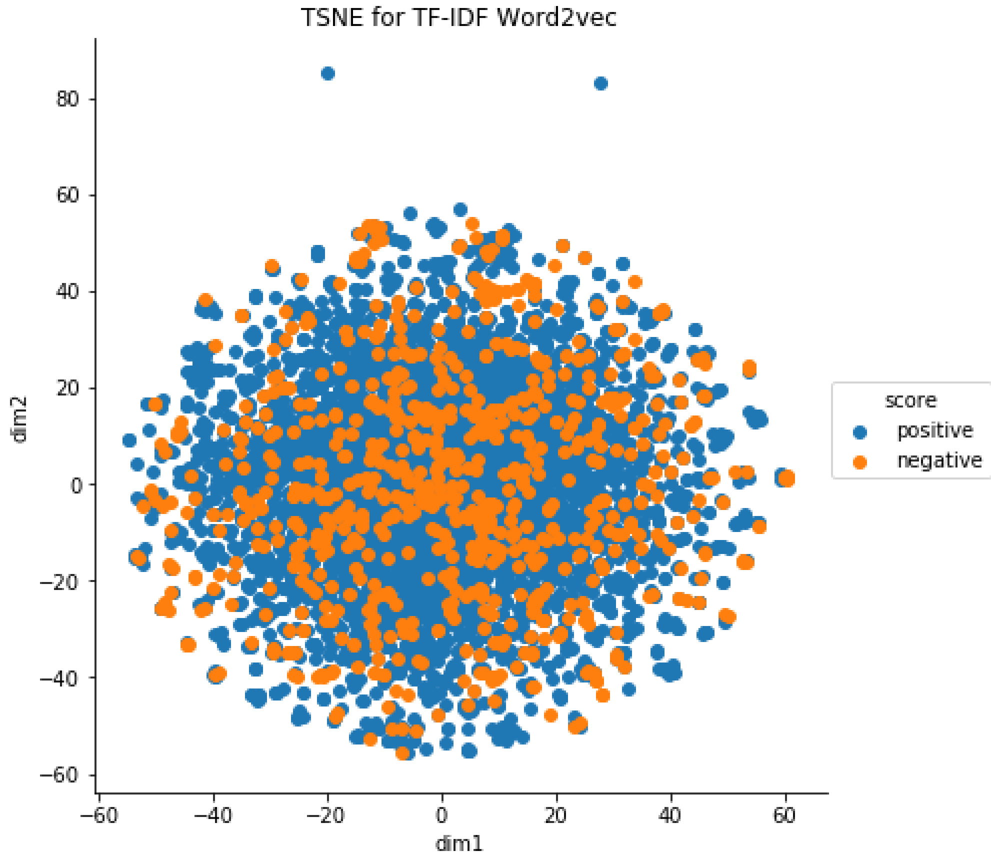
Out[77]: scipy.sparse.csr.csr_matrix

In [78]:
```
std_data_tfw2v=std_data_tfw2v.todense()
```

In [79]:
```
type(std_data_tfw2v)
```

Out[79]: numpy.matrixlib.defmatrix.matrix

In [80]:
```
#APPLYING TSNE FOR TF-IDF W2V
from sklearn.manifold import TSNE
model = TSNE()

tsne_data = model.fit_transform(std_data_tfw2v)

tsne_data = np.vstack((tsne_data.T, score)).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', 'd
im2').add_legend()
plt.title("TSNE for TF-IDF Word2vec")
plt.show()
```



Observation :

1. Hence the here the reviews are not linearly separable so we can't apply hyperplane to separate it.
2. There is alternate way to separate this reviews by feature transforming or feature enggineering.