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
```

```
C:\Users\Rishabh\Anaconda3\lib\site-packages\gensim\utils.py:1212: User
Warning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_seria
l")
```

[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 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 50000""", 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 'negative'
    return 'positive'

#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 (50000, 10)
```

Out[2]:

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

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

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	COU

	UserId	ProductId	ProfileName	Time	Score	Text	COU
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	(

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

In [6]:
```
display['COUNT(*)'].sum()
```
Out[6]:
```
393063
```

In [7]:
```
filtered_data.shape
```
Out[7]:
```
(50000, 10)
```

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	Helpfuln
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

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]:
```
(46072, 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]:
```
92.144
```

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	Helpfuln
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2

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()
```

```
(46071, 10)
```

Out[14]:
```
positive    38479
negative     7592
Name: Score, dtype: int64
```

In [15]:
```
# picking top 2500 Negative reviews and 2500 Positive Reviews

finnal_negative = final.loc[final['Score']=='negative']  # storing all
 negative reviews in finnal_negative
finnal_negative=finnal_negative.head(2500)               # picking top
 2500 negative reviews
finnal_positive = final.loc[final['Score']=='positive']  # storing all
 positive reviews in finnal_positive
finnal_positive = finnal_positive.head(2500)             # picking top
 2500 positive reviews
frames = [finnal_negative,finnal_positive]               # putting all
 positive and negative reviews in frame
result = pd.concat(frames)                               # concating th
e frame
result['Score'].value_counts()                           #
```

Out[15]:
```
negative    2500
positive    2500
Name: Score, dtype: int64
```

In [16]:
```
final = result
final['Score'].value_counts()
```

Out[16]:
```
negative    2500
positive    2500
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 [17]:

```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)

sent_1000 = final['Text'].values[1000]
print(sent_1000)
print("="*50)

sent_1500 = final['Text'].values[1500]
print(sent_1500)
print("="*50)

sent_4900 = final['Text'].values[4900]
print(sent_4900)
print("="*50)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
```

```
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
==================================================
arrived on time, all that is well, but did not taste good! i mean it's
quick, microwaveable, etc., but the taste was off and i couldn't even f
inish the pack of 6.
==================================================
I guess I am spoiled by my liquid creamers, but these did nothing for m
e. And I even tried using 4 tablets at a time! Gave them away, and no o
ne remarked that they liked them either. Sticking to my fat free liquid
creamer for now.
==================================================
I started drinking the French Vanilla coffee awhile back, once to twice
a day, but then my local Wal-Mart stopped selling it!  I was furious, b
ut luckily found it on Amazon.  This is the ONLY coffee I will drink, a
nd combined with some Splenda and sugar free french vanilla creamer, it
is OH SO YUMMY!
==================================================
```

In [18]:

```python
# remove urls from text python: https://stackoverflow.com/a/40823105/40
84039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
```

In [19]:

```python
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how
-to-remove-all-tags-from-an-element
from bs4 import BeautifulSoup

soup = BeautifulSoup(sent_0, 'lxml')
```

```
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1000, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1500, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_4900, 'lxml')
text = soup.get_text()
print(text)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
==================================================
arrived on time, all that is well, but did not taste good! i mean it's
quick, microwaveable, etc., but the taste was off and i couldn't even f
inish the pack of 6.
==================================================
I guess I am spoiled by my liquid creamers, but these did nothing for m
e. And I even tried using 4 tablets at a time! Gave them away, and no o
ne remarked that they liked them either. Sticking to my fat free liquid
creamer for now.
==================================================
I started drinking the French Vanilla coffee awhile back, once to twice
a day, but then my local Wal-Mart stopped selling it!  I was furious, b
ut luckily found it on Amazon.  This is the ONLY coffee I will drink, a
nd combined with some Splenda and sugar free french vanilla creamer, it
is OH SO YUMMY!
```

In [20]:
```python
# 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 [21]:
```python
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
I guess I am spoiled by my liquid creamers, but these did nothing for m
e. And I even tried using 4 tablets at a time! Gave them away, and no o
ne remarked that they liked them either. Sticking to my fat free liquid
creamer for now.
==================================================
```

In [22]:
```python
#remove words with numbers python: https://stackoverflow.com/a/1808237
0/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
```

In [23]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
I guess I am spoiled by my liquid creamers but these did nothing for me
And I even tried using 4 tablets at a time Gave them away and no one re
marked that they liked them either Sticking to my fat free liquid cream
er for now
```

In [24]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'no
t'
# <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', 'o
urs', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselve
s', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'it
s', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'th
is', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'h
ave', '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', 'h
ow', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 's
o', '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', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "is
n'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 [25]:
```
# Combining all the above stundents
from tqdm import tqdm
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%|████████████████████████████████████| 5000/5000 [00:04<00:00, 120
7.16it/s]
```

In [26]:
```
preprocessed_reviews[1500]
```

Out[26]:
```
'guess spoiled liquid creamers nothing even tried using tablets time ga
ve away no one remarked liked either sticking fat free liquid creamer'
```

[4] Featurization

[4.1] BAG OF WORDS

In [27]:
```
#BoW
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])
```

```
some feature names  ['aa', 'aaaaah', 'aachen', 'aafco', 'aback', 'abbe
y', 'abdomen', 'abdominal', 'ability', 'abit']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (5000, 14756)
the number of unique words  14756
```

In [28]:
```
final_counts.shape
```

Out[28]:
```
(5000, 14756)
```

In [29]:
```
type(final_counts)
```

Out[29]:
```
scipy.sparse.csr.csr_matrix
```

Observation -

1. Type of matrix is sparse that means having lot of '0' in it.

In [31]:
```
# Converting sparse matrix into densed matrix by using (        .toarra
y()             )

final_countss = final_counts.toarray()
type(final_countss)
```

Out[31]:
```
numpy.ndarray
```

In [32]:
```
# Checking score is of what type

labled = final['Score']
type(labled)
```

Out[32]: pandas.core.series.Series

In [34]:
```
# As the Final['Score'] is of pandas.core.series.Series converting it t
o numpy.ndarray for furture computation

lables = np.array(labled)
type(lables)
```

Out[34]: numpy.ndarray

In [35]:
```
# standardizing the data so that mean become 0 and std-dev becomes 1

from sklearn.preprocessing import StandardScaler

standardized_data = StandardScaler().fit_transform(final_countss)
print(standardized_data.shape)
```

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

```
(5000, 14756)
```

[4.2] Bi-Grams and n-Grams.

In [37]:
```
#bi-gram, tri-gram and n-gram
```

```
#removing stop words like "not" should be avoided before building n-gra
ms
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation http://scikit-learn.
org/stable/modules/generated/sklearn.feature_extraction.text.CountVecto
rizer.html
# you can choose these numebrs min_df=10, max_features=5000, of your ch
oice
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features
=5000)
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_s
hape())
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  (5000, 3334)
the number of unique words including both unigrams and bigrams  3334
```

[4.3] TF-IDF

In [38]:

```
# computing the TF-IDF on Preprocessed_reviews to get unique words in c
orpus

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.ge
t_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])
```

```
some sample features(unique words in the corpus) ['able', 'able find', 'able get', 'absolute', 'absolutely', 'absolutely delicious', 'absolutely love', 'absolutely no', 'acceptable', 'according']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (5000, 3334)
the number of unique words including both unigrams and bigrams  3334
```

In [40]:

```
# As the matrix is sparse converting it to densed matrix

densed_tf_idf= final_tf_idf.toarray()
type(densed_tf_idf)
```

Out[40]:

```
numpy.ndarray
```

In [41]:

```
# standardizing the data so that mean,std-dev become 0 and 1

standardized_dataa = StandardScaler().fit_transform(densed_tf_idf)
print(standardized_dataa.shape)
```

```
(5000, 3334)
```

[4.4] Word2Vec

In [42]:

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

In [43]:

```
# 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 val
ues
# To use this code-snippet, download "GoogleNews-vectors-negative300.bi
n"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edi
t
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17
SRFAzZPY
# 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=True)
        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_trai
n_w2v = True, to train your own w2v ")
```

```
[('feel', 0.9969224333763123), ('actually', 0.996658205986023), ('loo
k', 0.9964472055435181), ('overall', 0.9964209198951721), ('either', 0.
9962457418441772), ('looking', 0.9962129592895508), ('pretty', 0.996106
0285568237), ('say', 0.9961022138595581), ('else', 0.996100127696991),
('spiciness', 0.9960631728172302)]
```

```
==================================================
[('version', 0.9993889331817627), ('cookie', 0.9992836117744446), ('bev
erage', 0.9992769360542297), ('drinks', 0.9992743134498596), ('light',
0.9992599487304688), ('blend', 0.9992581605911255), ('wow', 0.999252200
126648), ('horrible', 0.9992465972900391), ('far', 0.9992419481277466),
('refreshing', 0.9992396831512451)]
```

In [45]:
```
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  4379
sample words  ['dogs', 'loves', 'chicken', 'product', 'china', 'wont',
'buying', 'anymore', 'hard', 'find', 'products', 'made', 'usa', 'one',
'isnt', 'bad', 'good', 'take', 'chances', 'till', 'know', 'going', 'fav
orite', 'places', 'frequent', 'afford', 'nice', 'meal', 'pungent', 'bes
ides', 'steaks', 'macadamia', 'best', 'things', 'place', 'coffee', 'ser
ve', 'fine', 'desserts', 'actually', 'main', 'reasons', 'go', 'night',
'enjoying', 'asked', 'dessert', 'got', 'served', 'rancid']
```

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

[4.4.1.1] Avg W2v

In [46]:
```
# 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, yo
u 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/re
view
    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]))
```

```
100%|██████████████████████████████████████| 5000/5000 [00:12<00:00, 40
2.52it/s]
```

```
5000
50
```

In [47]:
```
#standardizing the data

standardized_dataaa = StandardScaler().fit_transform(sent_vectors)
print(standardized_dataaa.shape)
```

```
(5000, 50)
```

[4.4.1.2] TFIDF weighted W2v

In [48]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
model.fit(preprocessed_reviews)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [49]:
```
# 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 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 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/r
eview
    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
```

```
100%|██████████████████████████████████████████████████| 5000/5000 [01:02<00:00, 11
3.98it/s]
```

In [50]:

```
# standarzing the data

standardized_dataa3 = StandardScaler().fit_transform(tfidf_sent_vectors
)
print(standardized_dataa3.shape)
```

```
(5000, 50)
```

[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)

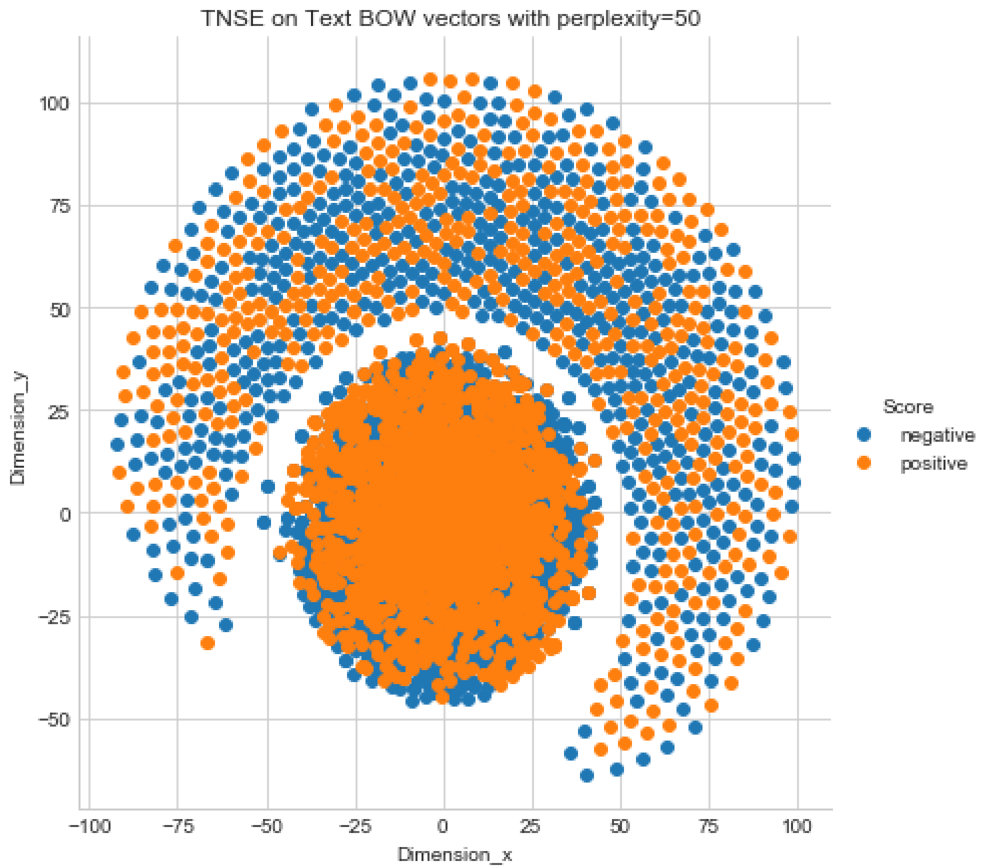
2. Note 1: The TSNE accepts only dense matrices
3. Note 2: Consider only 5k to 6k data points

[5.1] Applying TNSE on Text BOW vectors

In [53]:

```
# https://github.com/pavlin-policar/fastTSNE (Reference code)

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



x = standardized_data   # standarzing of data have been performed in th
e section of BOW vector
y = lables

tsne = TSNE(n_components=2, perplexity=50, learning_rate=200)

X_embedding = tsne.fit_transform(x)
# if x is a sparse matrix you need to pass it as X_embedding = tsne.fit
_transform(x.toarray()) , .toarray() will convert the sparse matrix int
o dense matrix

for_tsne = np.hstack((X_embedding, y.reshape(-1,1)))
for_tsne_df = pd.DataFrame(data=for_tsne, columns=['Dimension_x','Dimen
sion_y','Score'])


sns.set_style("whitegrid");
sns.FacetGrid(for_tsne_df, hue="Score", size=6) \
   .map(plt.scatter, 'Dimension_x', 'Dimension_y') \
   .add_legend();
```

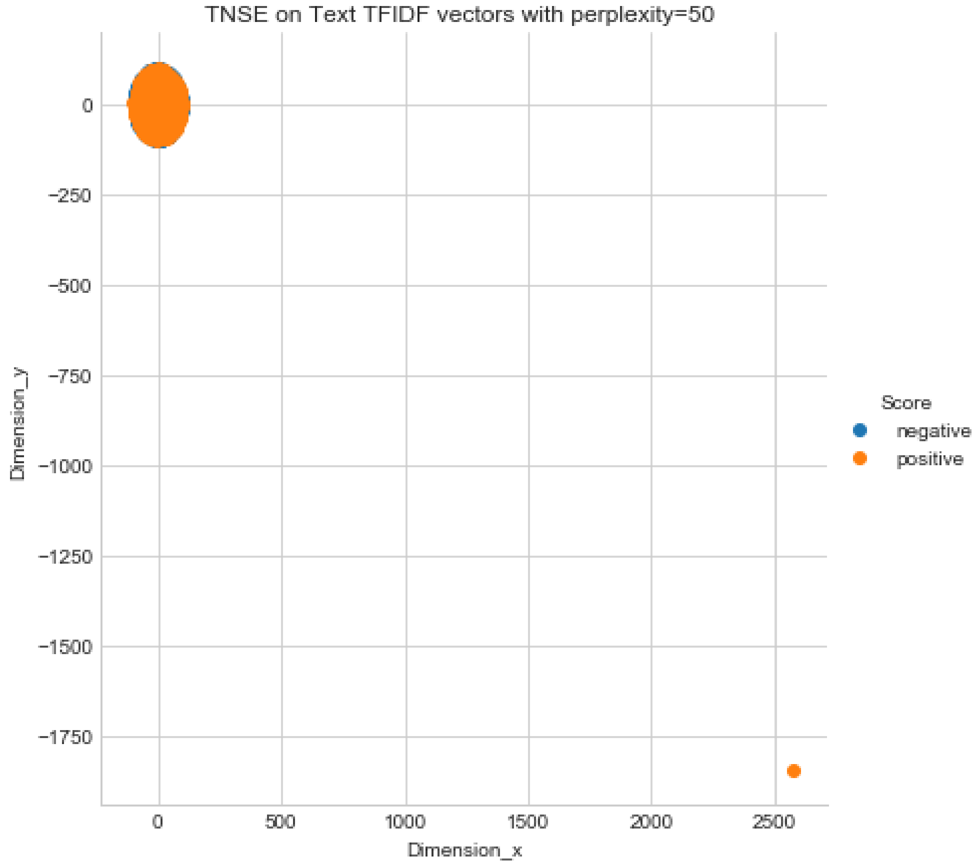
```
plt.title("TNSE on Text BOW vectors with perplexity=50 ")
plt.show();
```



[5.2] Applying TNSE on Text TFIDF vectors

In [54]:
```
# https://github.com/pavlin-policar/fastTSNE (Reference code)

x = standardized_dataa  # standarzing of data have been performed in th
e section of TFIDF vector
y = lables
```

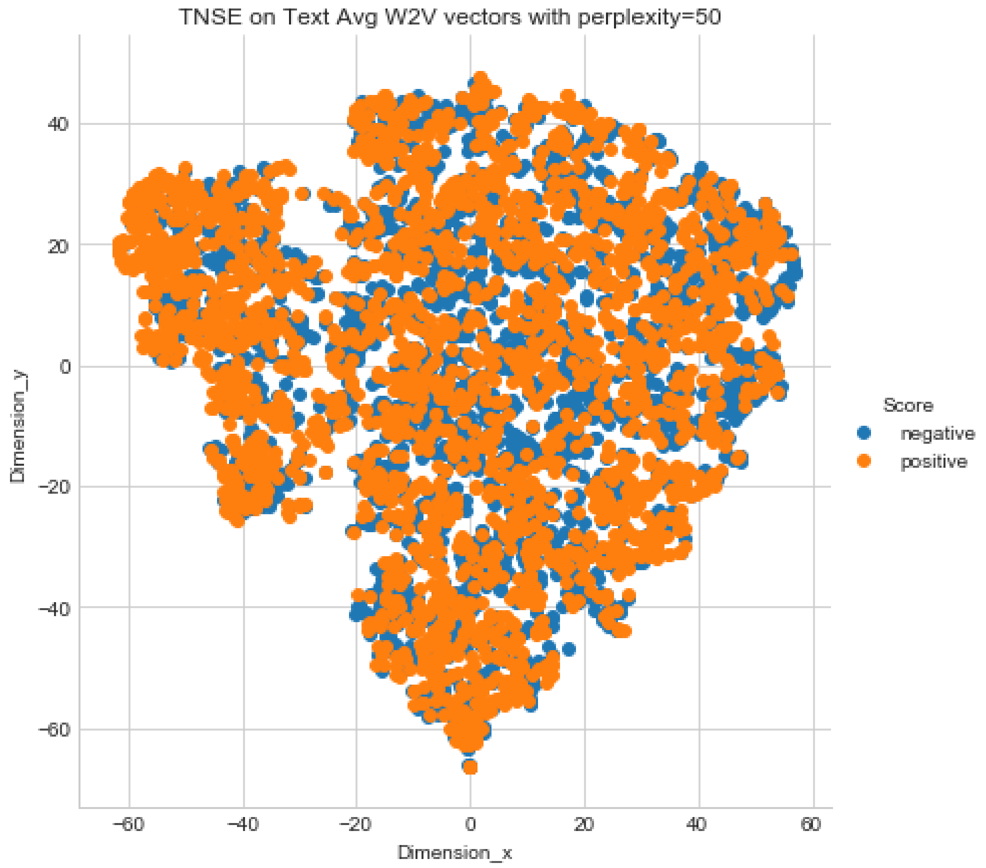
```
tsne = TSNE(n_components=2, perplexity=50, learning_rate=200)

X_embedding = tsne.fit_transform(x)
# if x is a sparse matrix you need to pass it as X_embedding = tsne.fit
_transform(x.toarray()) , .toarray() will convert the sparse matrix int
o dense matrix

for_tsne = np.hstack((X_embedding, y.reshape(-1,1)))
for_tsne_df = pd.DataFrame(data=for_tsne, columns=['Dimension_x','Dimen
sion_y','Score'])

sns.set_style("whitegrid");
sns.FacetGrid(for_tsne_df, hue="Score", size=6) \
   .map(plt.scatter, 'Dimension_x', 'Dimension_y') \
   .add_legend();
plt.title("TNSE on Text TFIDF vectors with perplexity=50 ")
plt.show();
```



[5.3] Applying TNSE on Text Avg W2V vectors

In [56]:
```
# https://github.com/pavlin-policar/fastTSNE (Reference code)

x = standardized_dataaa  # standarzing of data have been performed in t
he section of Avg W2V vector
y = lables

tsne = TSNE(n_components=2, perplexity=50, learning_rate=200)
```

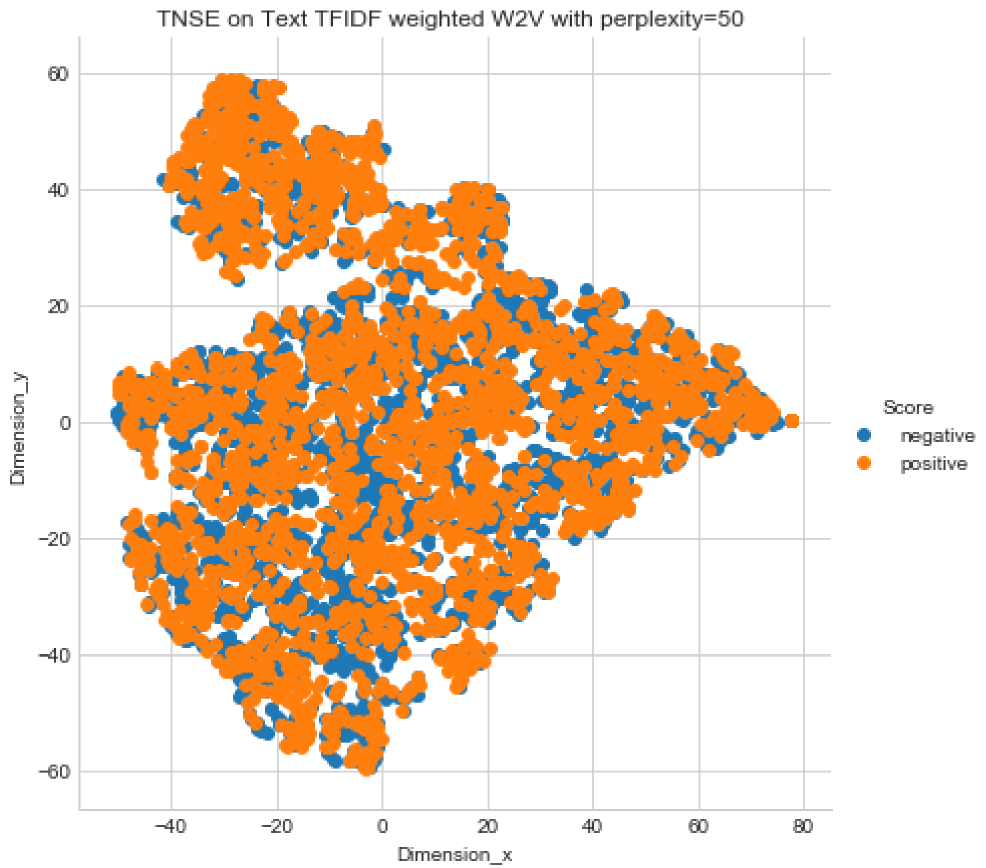
```
X_embedding = tsne.fit_transform(x)
# if x is a sparse matrix you need to pass it as X_embedding = tsne.fit
_transform(x.toarray()) , .toarray() will convert the sparse matrix int
o dense matrix

for_tsne = np.hstack((X_embedding, y.reshape(-1,1)))
for_tsne_df = pd.DataFrame(data=for_tsne, columns=['Dimension_x','Dimen
sion_y','Score'])

sns.set_style("whitegrid");
sns.FacetGrid(for_tsne_df, hue="Score", size=6) \
   .map(plt.scatter, 'Dimension_x', 'Dimension_y') \
   .add_legend();
plt.title("TNSE on Text Avg W2V vectors with perplexity=50 ")
plt.show();
```



[5.4] Applying TNSE on Text TFIDF weighted W2V vectors

In [57]:
```
# https://github.com/pavlin-policar/fastTSNE (Reference code)

x = standardized_dataa3 # standarzing of data have been performed in th
e section of TFIDF Weighted W2V vector
y = lables

tsne = TSNE(n_components=2, perplexity=50, learning_rate=200)
```

```
X_embedding = tsne.fit_transform(x)
# if x is a sparse matrix you need to pass it as X_embedding = tsne.fit_transform(x.toarray()) , .toarray() will convert the sparse matrix into dense matrix

for_tsne = np.hstack((X_embedding, y.reshape(-1,1)))
for_tsne_df = pd.DataFrame(data=for_tsne, columns=['Dimension_x','Dimension_y','Score'])

sns.set_style("whitegrid");
sns.FacetGrid(for_tsne_df, hue="Score", size=6) \
   .map(plt.scatter, 'Dimension_x', 'Dimension_y') \
   .add_legend();
plt.title("TNSE on Text TFIDF weighted W2V with perplexity=50 ")
plt.show();
```



[6] Conclusions

1. T-SNE plot for BOW Vector the positive and negative point are not well seperated plane can't be drawn to seperate the points.
2. T-SNE plot for TFIDF vectors the positive point almost completely overlap the negative point.
3. T-SNE plot for Avg W2V vectors the positive and negative point are not well seperated plane can't be drawn to seperate the points.

4. T-SNE plot for TFIDF weighted W2V the positive and negative point are not well seperated plane can't be drawn to seperate the points.
5. Plot for above have been drawn after trying multipal value of perplexity and the best one is plotted Here.