Amazon Fine Food Reviews Analysis

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

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/ (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 [35]:

```
%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 [4]:

```
# 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 p
# 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 LIM
# for tsne assignment you can take 5k data points

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

# Give reviews with Score>3 a positive rating, and reviews with a score<3 a negati
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
filtered_data['Score'] = filtered_data['Score'].map(partition)
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (3000, 10)
```

Out[4]:

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

In [5]:
```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [6]:
```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[6]:

	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 [7]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[7]:

	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 [8]:
```
display['COUNT(*)'].sum()
```

Out[8]:
```
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 [9]:

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

Out[9]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomir
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	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 [10]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace
```

In [11]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, k
final.shape
```

Out[11]:
```
(2991, 10)
```

In [12]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[12]:
```
99.7
```

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 [13]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[13]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomir
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	

In [14]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [66]:

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

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

```
(2991, 10)
```

Out[66]:

```
1    2507
0     484
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)
```

```
Why is this $[...] when the same product is available for $[...] here?<br />ht
tp://www.amazon.com/VICTOR-FLY-MAGNET-BAIT-REFILL/dp/B00004RBDY<br /><br />The
Victor M380 and M502 traps are unreal, of course -- total fly genocide. Pretty
stinky, but only right nearby.
==================================================
i like it and i find the grinder bottle to be very convenient. i like the tast
e of it and i would definitely buy it again, and considering its healthy use f
or sure.
==================================================
I have lived out of the US for over 7 yrs now, and I so miss my Twizzlers!!  W
hen I go back to visit or someone visits me, I always stock up.  All I can say
is YUM!<br />Sell these in Mexico and you will have a faithful buyer, more oft
en than I'm able to buy them right now.
==================================================
```

In [18]:

```
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
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)
```

```
Why is this $[...] when the same product is available for $[...] here?<br /> /
><br />The Victor M380 and M502 traps are unreal, of course -- total fly genoc
ide. Pretty stinky, but only right nearby.
```

In [19]:

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-
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)
```

```
Why is this $[...] when the same product is available for $[...] here? />The V
ictor M380 and M502 traps are unreal, of course -- total fly genocide. Pretty
stinky, but only right nearby.
==================================================
i like it and i find the grinder bottle to be very convenient. i like the tast
e of it and i would definitely buy it again, and considering its healthy use f
or sure.
==================================================
I have lived out of the US for over 7 yrs now, and I so miss my Twizzlers!!  W
hen I go back to visit or someone visits me, I always stock up.  All I can say
is YUM!Sell these in Mexico and you will have a faithful buyer, more often tha
n I'm able to buy them right now.
==================================================
```

In [20]:

```
# 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]:

```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
I have lived out of the US for over 7 yrs now, and I so miss my Twizzlers!!  W
hen I go back to visit or someone visits me, I always stock up.  All I can say
is YUM!<br />Sell these in Mexico and you will have a faithful buyer, more oft
en than I am able to buy them right now.
==================================================
```

In [22]:

```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
Why is this $[...] when the same product is available for $[...] here?<br /> /
><br />The Victor  and  traps are unreal, of course -- total fly genocide. Pre
tty stinky, but only right nearby.
```

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 have lived out of the US for over 7 yrs now and I so miss my Twizzlers When
I go back to visit or someone visits me I always stock up All I can say is YUM
br Sell these in Mexico and you will have a faithful buyer more often than I a
m able to buy them right now
```

In [24]:

```
# 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 ste

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ours
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'h
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that'
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has'
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because',
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'thr
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off'
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all',
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than',
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've"
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "did
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn'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 st
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████████████| 2991/2991 [00:01<00:00, 1787.38it/
s]
```

In [26]:

```
preprocessed_reviews[1500]
```

Out[26]:

```
'lived us yrs miss twizzlers go back visit someone visits always stock say yum
sell mexico faithful buyer often able buy right'
```

[4] Featurization

[4.1] BAG OF WORDS

In [28]:

```
#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  ['aahhhs', 'aback', 'abates', 'abby', 'abdominal', 'abidin
g', 'ability', 'able', 'aboulutely', 'absence']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (2991, 10052)
the number of unique words  10052
```

[4.2] Bi-Grams and n-Grams.

In [30]:

```
#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/
# 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=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_shape())
print("the number of unique words including both unigrams and bigrams ", final_big
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (2991, 2002)
the number of unique words including both unigrams and bigrams  2002
```

[4.3] TF-IDF

In [32]:

```
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_n
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_
```

```
some sample features(unique words in the corpus) ['able', 'able find', 'absolu
te', 'absolutely', 'absolutely delicious', 'absolutely love', 'according', 'ac
id', 'across', 'actual']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (2991, 2002)
the number of unique words including both unigrams and bigrams  2002
```

[4.4] Word2Vec

In [33]:

```
# 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 [36]:

```
# 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-negative30
        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 = Tru
```

```
[('quite', 0.999578058719635), ('people', 0.99957275390625), ('perfect', 0.999
5571374893188), ('snack', 0.9995522499084473), ('good', 0.9995518922805786),
('things', 0.9995425939559937), ('light', 0.9995341300964355), ('want', 0.9995
332956314087), ('takes', 0.9995288252830505), ('thing', 0.9995276927947998)]
==================================================
[('fresh', 0.9994485974311829), ('heavy', 0.9993903636932373), ('cookies', 0.9
993882179260254), ('fluffy', 0.9993659257888794), ('must', 0.999357819557189
9), ('loved', 0.9993479251861572), ('tasted', 0.9993464946746826), ('extra',
0.9993451833724976), ('kids', 0.9993376135826111), ('keurig', 0.99933499097824
1)]
```

In [37]:

```
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  2811
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stinky',
'right', 'nearby', 'used', 'ca', 'not', 'beat', 'great', 'received', 'shipmen
t', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'instead', 'removed', 'e
asily', 'daughter', 'designed', 'printed', 'use', 'car', 'shop', 'program', 'g
oing', 'lot', 'fun', 'everywhere', 'like', 'computer', 'really', 'good', 'ide
a', 'outstanding', 'everybody', 'asks', 'bought', 'made', 'two', 'thumbs', 'gl
ad', 'cocker', 'standard']
```

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

[4.4.1.1] Avg W2v

In [38]:

```
# 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 nee
    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]))
```

```
100%|██████████████████████████████████████| 2991/2991 [00:03<00:00, 993.28it/
s]

2991
50
```

[4.4.1.2] TFIDF weighted W2v

In [89]:

```
# 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 value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [40]:
```
# 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 = tf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in thi
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
```

```
100%|██████████████████████████████████████| 2991/2991 [00:15<00:00, 189.35it/
s]
```

[5] Applying TSNE

In [0]:

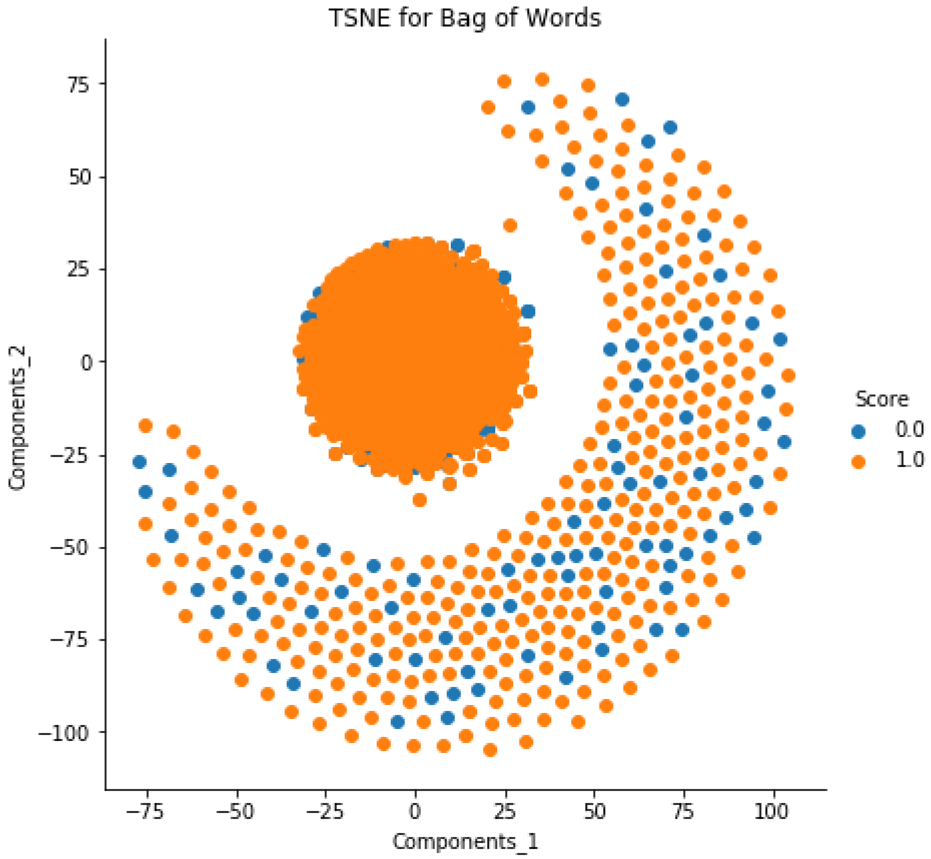
```
# https://github.com/pavlin-policar/fastTSNE
import numpy as np
from openTSNE import TSNE
from sklearn import datasets

iris = datasets.load_iris()
x, y = iris['data'], iris['target']

tsne = TSNE( n_components=2, perplexity=30, learning_rate=200, n_jobs=4,initializa
            metric='euclidean', early_exaggeration_iter=250, early_exaggeration=12

X_embedding = tsne.fit(x)

for_tsne = np.hstack((X_embedding, y.reshape(-1,1)))
for_tsne_df = pd.DataFrame(data=for_tsne, columns=['Dimension_x','Dimension_y','Sc
colors = {0:'red', 1:'blue', 2:'green'}
plt.scatter(for_tsne_df['Dimension_x'], for_tsne_df['Dimension_y'], c=for_tsne_df[
plt.show()
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-1-0026c1e9273a> in <module>()
      1 import numpy as np
----> 2 from openTSNE import TSNE
      3 from sklearn import datasets
      4 
      5 iris = datasets.load_iris()

ModuleNotFoundError: No module named 'openTSNE'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

[5.1] Applying TNSE on Text BOW vectors

In [78]:

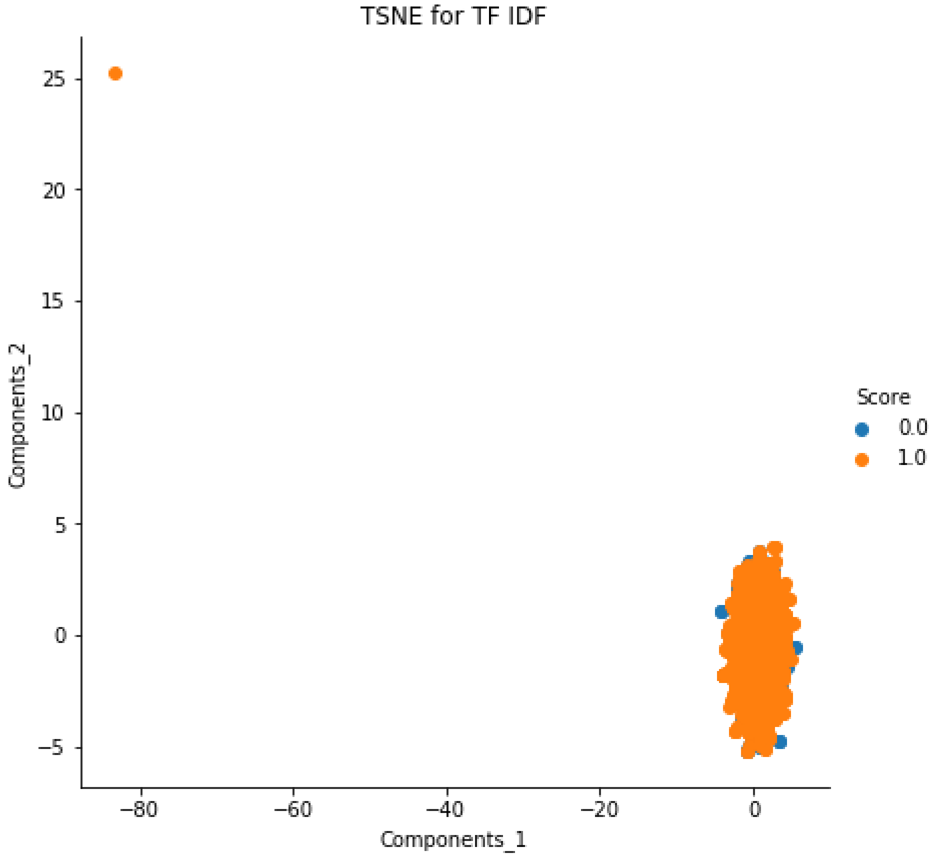
```
# please write all the code with proper documentation, and proper titles for each
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
from sklearn.manifold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

# Standardization
bow_standard_data = StandardScaler().fit_transform(final_counts.todense())

model = TSNE(n_components=2, random_state=0)

tsne_data = model.fit_transform(bow_standard_data)

label=final['Score']

tsne_data = np.vstack((tsne_data.T, label)).T

tsne_df = pd.DataFrame(data=tsne_data, columns=("Components_1","Components_2","Sco

sns.FacetGrid(tsne_df, hue="Score", size=6).map(plt.scatter, "Components_1", "Comp
plt.title("TSNE for Bag of Words")
plt.show()
```



[5.1] Applying TNSE on Text TFIDF vectors

In [77]:

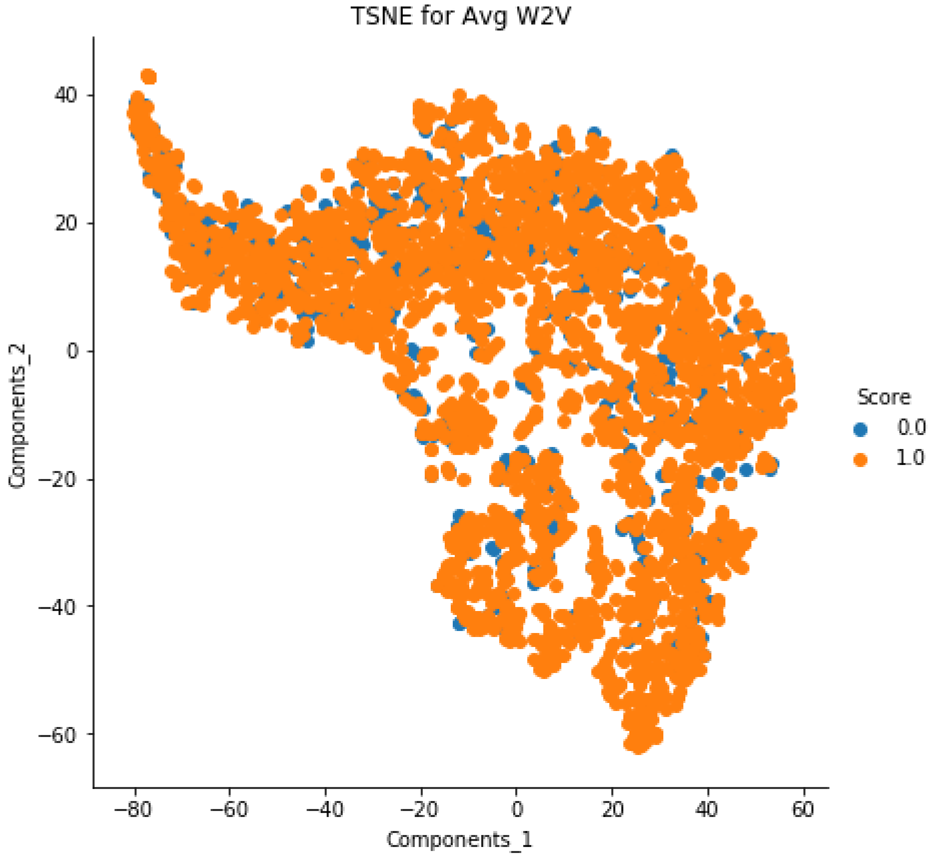
```
# please write all the code with proper documentation, and proper titles for each
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label

# Standardization
tfidf_standard_data = StandardScaler().fit_transform(final_tf_idf.todense())

model = TSNE(n_components=2, random_state=0)

tsne_data = model.fit_transform(tfidf_standard_data)

label=final['Score']

tsne_data = np.vstack((tsne_data.T, label)).T

tsne_df = pd.DataFrame(data=tsne_data, columns=("Components_1","Components_2","Sco

sns.FacetGrid(tsne_df, hue="Score", size=6).map(plt.scatter, "Components_1", "Comp
plt.title("TSNE for TF IDF")
plt.show()
```



[5.3] Applying TNSE on Text Avg W2V vectors

In [80]:

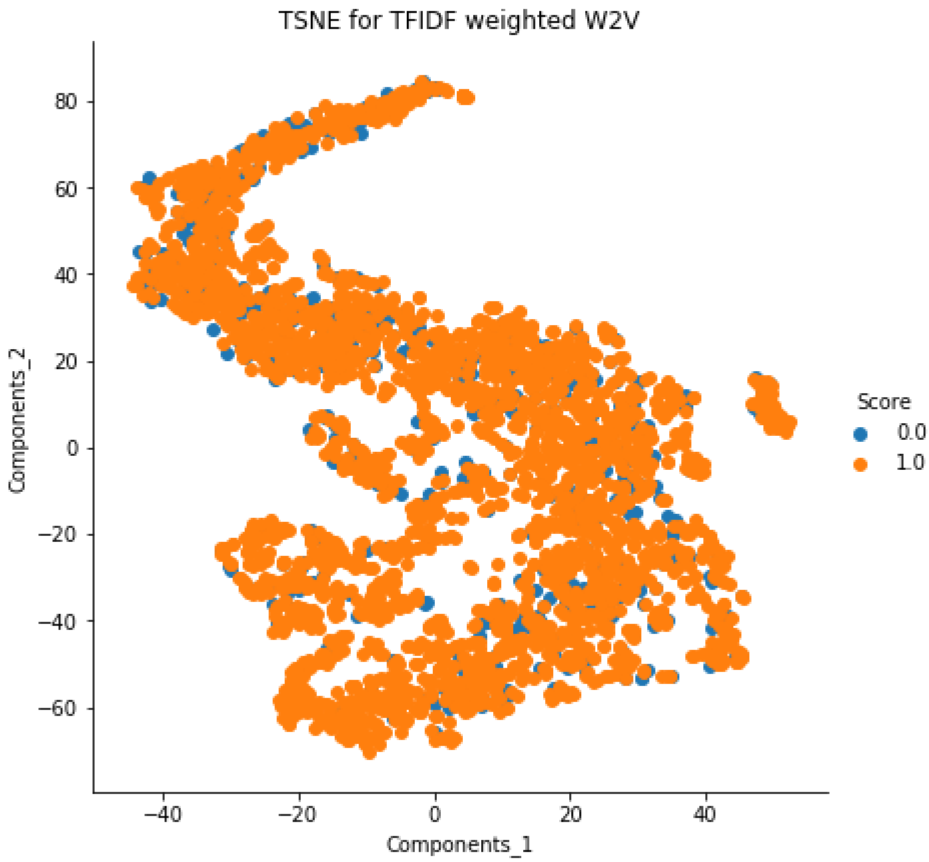
```
# please write all the code with proper documentation, and proper titles for each
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
# from sklearn.manifold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

# Standardization
AvgW2V_standard_data = StandardScaler().fit_transform(sent_vectors)

model = TSNE(n_components=2, random_state=0)

tsne_data = model.fit_transform(AvgW2V_standard_data)

label=final['Score']

tsne_data = np.vstack((tsne_data.T, label)).T

tsne_df = pd.DataFrame(data=tsne_data, columns=("Components_1","Components_2","Sco

sns.FacetGrid(tsne_df, hue="Score", size=6).map(plt.scatter, "Components_1", "Comp
plt.title("TSNE for Avg W2V")
plt.show()
```



[5.4] Applying TNSE on Text TFIDF weighted W2V vectors

In [81]:

```
# please write all the code with proper documentation, and proper titles for each
# 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

# Standardization
TFIDF_W2V_standard_data = StandardScaler().fit_transform(tfidf_sent_vectors)

model = TSNE(n_components=2, random_state=0)

tsne_data = model.fit_transform(TFIDF_W2V_standard_data)

label=final['Score']

tsne_data = np.vstack((tsne_data.T, label)).T

tsne_df = pd.DataFrame(data=tsne_data, columns=("Components_1","Components_2","Sco

sns.FacetGrid(tsne_df, hue="Score", size=6).map(plt.scatter, "Components_1", "Comp
plt.title("TSNE for TFIDF weighted W2V")
plt.show()
```



[6] Conclusions

1. Used dataset of 3K data points

2. In all 4 TSNE representation positive and negative reviews are massively overlapping each other