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)

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.

[7.1] 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]:

```
#importing Libraries
%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\HIMANSHU NEGI\Anaconda3\lib\site-packages\gensim\utils.py:1212: U
serWarning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

[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.
filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500
0""", con)

# Give reviews with Score>3 a positive rating, and reviews with a score<3 a negative ra
ting.
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)
```

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

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Hel
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

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

In [5]:

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

Out[5]:

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

In [6]:

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

Out[6]:

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 is necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

In [7]:

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

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helr
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 [8]:

```
#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 [9]:

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

Out[9]:

```
(4986, 10)
```

In [10]:

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

Out[10]:

```
99.72
```

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 [11]:

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

display.head()
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Help
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2

In [12]:

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

In [13]:

```
#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()
```

```
(4986, 10)
```

Out[13]:

```
1    4178
0     808
Name: Score, dtype: int64
```

In [14]:

```
final['Text'].shape
```

Out[14]:

```
(4986,)
```

[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]:

```python
# 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
/>http://www.amazon.com/VICTOR-FLY-MAGNET-BAIT-REFILL/dp/B00004RBDY<br /><
br />The Victor M380 and M502 traps are unreal, of course -- total fly gen
ocide. Pretty stinky, but only right nearby.
==================================================
I recently tried this flavor/brand and was surprised at how delicious thes
e chips are.  The best thing was that there were a lot of "brown" chips in
the bsg (my favorite), so I bought some more through amazon and shared wit
h family and friends.  I am a little disappointed that there are not, so f
ar, very many brown chips in these bags, but the flavor is still very goo
d.  I like them better than the yogurt and green onion flavor because they
do not seem to be as salty, and the onion flavor is better.  If you have
n't eaten Kettle chips before, I recommend that you try a bag before buyin
g bulk.  They are thicker and crunchier than Lays but just as fresh out of
the bag.
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they w
ere ordering; the other wants crispy cookies.  Hey, I'm sorry; but these r
eviews do nobody any good beyond reminding us to look  before ordering.<br
/><br />These are chocolate-oatmeal cookies.  If you don't like that combi
nation, don't order this type of cookie.  I find the combo quite nice, rea
lly.  The oatmeal sort of "calms" the rich chocolate flavor and gives the
cookie sort of a coconut-type consistency.  Now let's also remember that t
astes differ; so, I've given my opinion.<br /><br />Then, these are soft,
chewy cookies -- as advertised.  They are not "crispy" cookies, or the blu
rb would say "crispy," rather than "chewy."  I happen to like raw cookie d
ough; however, I don't see where these taste like raw cookie dough.  Both
are soft, however, so is this the confusion?  And, yes, they stick togethe
r.  Soft cookies tend to do that.  They aren't individually wrapped, which
would add to the cost.  Oh yeah, chocolate chip cookies tend to be somewha
t sweet.<br /><br />So, if you want something hard and crisp, I suggest Na
biso's Ginger Snaps.  If you want a cookie that's soft, chewy and tastes l
ike a combination of chocolate and oatmeal, give these a try.  I'm here to
place my second order.
==================================================
love to order my coffee on amazon.  easy and shows up quickly.<br />This k
cup is great coffee.  dcaf is very good as well
==================================================
```

In [16]:

```
# 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 f
ly genocide. Pretty stinky, but only right nearby.
```

In [17]:

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup
# removing-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_1001, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1502, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_4903, 'lxml')
text = soup.get_text()
print(text)
```

```
Why is this $[...] when the same product is available for $[...] here? />T
he Victor M380 and M502 traps are unreal, of course -- total fly genocide.
Pretty stinky, but only right nearby.
==================================================
We are always pleased with Amazon's packaging. Chips are never crushed or
broken, always fresh. We really liked this brand/flavor of chips. Very tas
ty!
==================================================
I purchased these thinking they would just be ordinary oatmeal raisin cook
ies. Wrong! I like chocolate chip and I like oatmeal raisin but these comb
ination cookies just don't work well together. I wish I had read the item
listing more carefully. Even my kids said "Blech!".
==================================================
This is a great coffee.  Its dark but not bitter, a little chocolaty, and
it reminds me of an Adirondack lodge.  I can get them for a better price i
n a Caribou Coffee store, however.
```

In [18]:

```
# 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 [19]:

```
sent_1502 = decontracted(sent_1502)
print(sent_1502)
print("="*50)
```

```
I purchased these thinking they would just be ordinary oatmeal raisin cook
ies. Wrong! I like chocolate chip and I like oatmeal raisin but these comb
ination cookies just do not work well together. I wish I had read the item
listing more carefully. Even my kids said "Blech!".
==================================================
```

In [20]:

```
#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 genoc
ide. Pretty stinky, but only right nearby.
```

In [21]:

```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1502 = re.sub('[^A-Za-z0-9]+', ' ', sent_1502)
print(sent_1502)
```

```
I purchased these thinking they would just be ordinary oatmeal raisin cook
ies Wrong I like chocolate chip and I like oatmeal raisin but these combin
ation cookies just do not work well together I wish I had read the item li
sting more carefully Even my kids said Blech
```

In [22]:

```
# 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', 'ourselve
s', 'you', "you'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', 't
hey', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
at'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
d', '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', 'ov
er', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
y', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too'
, 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
w', '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', "isn't", 'ma', 'migh
tn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
asn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [23]:

```
# 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 stopwor
ds)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|████████████████████████████████████████████████████████████████████
███████| 4986/4986 [00:02<00:00, 1853.76it/s]
```

In [24]:

```
preprocessed_reviews[1502]
```

Out[24]:

```
'purchased thinking would ordinary oatmeal raisin cookies wrong like choco
late chip like oatmeal raisin combination cookies not work well together w
ish read item listing carefully even kids said blech'
```

[3.2] Preprocess Summary for reviews

In [93]:

```
##  preprocessing for review summary.
final['Summary'].shape
print(final['Summary'])
```

```
2546                                          thirty bucks?
2547                                           Flies Begone
1145                            WOW Make your own 'slickers' !
1146                                          Great Product
2942                                            Good stuff!
2941                               Premium Quality Dog Food!!!
1071                                          Cats love it!
2187                         Nice, Big Pieces & Big Almond Flavor
4695                          A Summer Treat Fat Free, Guilt Free
2068    Don't buy this product unless you are looking ...
2069                                          Little Flavor
2806                                    A Staple in my house
2805                            A favorite quick meal solution
4099    Best Hot Sauce and Taco Sauce Available In Ame...
4096                                  Pico Pica ... the BEST
4097                                    This is the stuff!
4098    What everyone is saying here about Pico Pica i...
1332                                             not edible
1330                                           shining star
1329            The Inexpensive Alternative to Gold Leaf!
1328                         Create Exquisite Cake Decorations
1331                                    gold dust is awesome
4320                                Perfect for my sons cake.
4321                     Really cute - made a great golf cake
4322                                               Golf Set
4323                                            Cake Topper
4054                                              Fantastic
2477                                           Adzuki beans
2476                                                   Yum.
2478                                                  yummy
                                   ...
2214                                            burnt toast
2215                                          Love Caribou!
2212                                  down to almost none ....
2216                                                    yum
2217                                           great coffee
677                                       cute ,cute, cute!
678                                       A Surprising Find
3663                                          It's the best
3662                                               Huge Fan
3664                                         Awesome sauce!
3580                         Good Malta, baffling business model
1110                                       Service was good
1109                            I was able to eat bread again!
1108                                              Love Taro
1107                                     Delicious and easy
1106                               Favorite thing about Brazil
1232                                               Delicous
4714                                  Best Italian olive oil
2013                              Mild Taste, But Delicious.
3567                             Great drink, horrible price!
3271                                            More for Me
220                             OMG best chocolate jelly belly
4117                                            GREAT STUFF
4118                                    You won't believe it
712                                 One of the better T-Discs
711                              great coffee - terrible price
710                                   Best of the Tassimo's
709                                  Good Tasting cup o' joe
713                                        Kona for Tassimo
```

```
1362    weak coffee not good for a premium product and...
Name: Summary, Length: 4986, dtype: object
```

In [158]:

```
# printing some random reviews
summ_0 = final['Summary'].values[0]
print(summ_0)
print("="*50)

summ_1000 = final['Summary'].values[1000]
print(summ_1000)
print("="*50)

summ_1500 = final['Summary'].values[1500]
print(summ_1500)
print("="*50)

summ_4900 = final['Summary'].values[4900]
print(summ_4900)
print("="*50)
```

```
thirty bucks?
==================================================
Best sour cream & onion chip I've had
==================================================
Are We Reviewing Our Mistakes Or These Cookies?
==================================================
caribou
==================================================
```

In [159]:

```
# remove urls from text python:
summ_0 = re.sub(r"http\S+", "", summ_0)
summ_1000 = re.sub(r"http\S+", "", summ_1000)
summ_1500 = re.sub(r"http\S+", "", summ_1500)
summ_4900 = re.sub(r"http\S+", "", summ_4900)

print(summ_0)
```

```
thirty bucks?
```

In [160]:

```
# remove urls from text python:
i=0;
for sm in x:
    if (len(re.findall('<.*?>', sm))):
        summ = re.sub(r"http\S+", "", sm)
        print(i)
        print(sent)

    i += 1;



print(sm)
```

```
weak coffee not good for a premium product and price
```

In [161]:

```
# removing-all-tags-from-an-element
from bs4 import BeautifulSoup

soup = BeautifulSoup(summ_0, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(summ_1000, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(summ_1500, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(summ_4900, 'lxml')
text = soup.get_text()
print(text)
```

```
thirty bucks?
==================================================
Best sour cream & onion chip I've had
==================================================
Are We Reviewing Our Mistakes Or These Cookies?
==================================================
caribou
```

In [151]:

```
# function to clean the word of any punctuation or special characters
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 [162]:

```
summ_1500 = decontracted(summ_1500)
print(summ_1500)
print("="*50)
```

```
Are We Reviewing Our Mistakes Or These Cookies?
==================================================
```

In [163]:

```
#remove words with numbers python:
summ_0 = re.sub("\S*\d\S*", "", summ_0).strip()
print(summ_0)
```

```
thirty bucks?
```

In [164]:

```
#remove spacial character:
summ_1500 = re.sub('[^A-Za-z0-9]+', ' ', summ_1500)
print(summ_1500)
```

```
Are We Reviewing Our Mistakes Or These Cookies
```

In [155]:

```
# 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', 'ourselve
s', 'you', "you'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', 't
hey', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
at'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
d', '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', 'ov
er', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
y', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too'
, 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
w', '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', "isn't", 'ma', 'migh
tn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
asn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [168]:

```
# Combining all the above stundents
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 stopwor
ds)
    preprocessed_summary.append(sentance.strip())
```

```
100%|██████████████████████████████████████████████████████████████████████
███████| 4986/4986 [00:01<00:00, 2627.73it/s]
```

In [169]:

```
preprocessed_summary[1500]
```

Out[169]:

```
'reviewing mistakes cookies'
```

[4] Featurization

[4.1] BAG OF WORDS

In [25]:

```
#BoW
count_vect = CountVectorizer() #function 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', 'aahhhs', 'aback', 'abandon', 'abates', 'abbot
t', 'abby', 'abdominal', 'abiding', 'ability']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 12997)
the number of unique words  12997
```

[4.2] Bi-Grams and n-Grams.

In [26]:

```
#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/modul
es/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=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_bigram_c
ounts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.3] TF-IDF

In [27]:

```
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.g
et_shape()[1])
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'able
find', 'able get', 'absolute', 'absolutely', 'absolutely delicious', 'abso
lutely love', 'absolutely no', 'according']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

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

[4.4.1.1] Avg W2v

In [32]:

```
# 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]))
```

```
100%|██████████████████████████████████████████████████████████████████████
████████| 4986/4986 [00:07<00:00, 706.58it/s]

4986
50
```

[4.4.1.2] TFIDF weighted W2v

In [33]:

```
# 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 [34]:

```
# 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 lis
t
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%|██████████████████████████████████████████████████████████████████████
███████| 4986/4986 [00:33<00:00, 147.21it/s]
```

[5] Applying TSNE

[5.0] Applying TNSE on Text BOW vectors

BOW vectors are sparse matrices so we have to convert it into dense matrices using todense() function

In [36]:

```
final_counts = final_counts.todense()
```

In [63]:

```
type(final_counts)
```

Out[63]:

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

In [41]:

```
print(final_counts.shape)
```

```
(4986, 12997)
```

In [52]:

```
final['Score'].value_counts()
```

Out[52]:

```
1    4178
0     808
Name: Score, dtype: int64
```

In [62]:

```
print(final['Score'].shape)
```

```
(4986,)
```

t-SNE of Bag of Words(BoW) with perplexity = 30 and n_iter = 1000

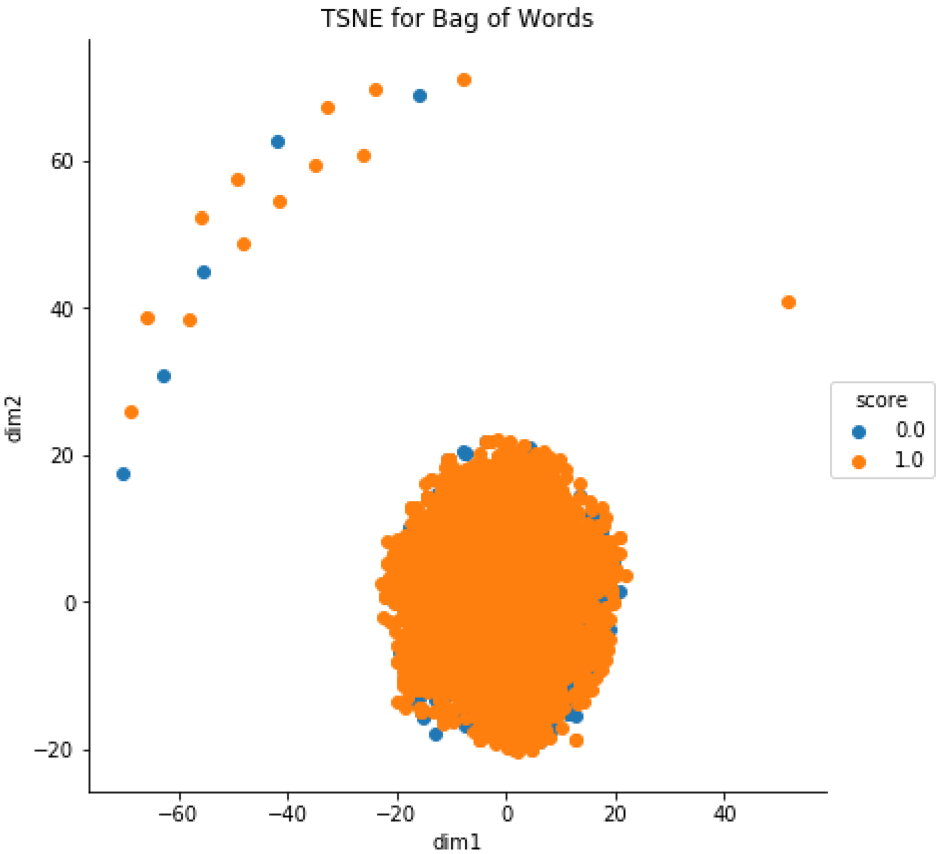
In [66]:

```
from sklearn.manifold import TSNE
model = TSNE(n_components=2, random_state=0, perplexity = 30, n_iter = 1000)
# configuring the parameteres
# the number of components = 2
# default perplexity = 30
# default learning rate = 200
# default Maximum number of iterations for the optimization = 1000

tsne_datAbow = model.fit_transform(final_counts)


# creating a new data frame which help us in ploting the result data
tsne_datAbow = np.vstack((tsne_datAbow.T, final['Score'])).T
tsnebow_df = pd.DataFrame(data=tsne_datAbow, columns=("dim1", "dim2", "score"))

# Ploting the result of tsne
sns.FacetGrid(tsnebow_df, hue="score", size=6).map(plt.scatter, 'dim1', 'dim2').add_leg
end()
plt.title("TSNE for Bag of Words")
plt.show()
```



OBSERVATION:-

1> From above plots it is observed that around 90% and above points are overlapping so it is very difficult to classify the polarity of the reviews i.e positive or negative.

2> It is hard to classify points using a simple linear line or a plane.

3> Here are some points which are very far from most of the points which can be outliers and can easily affect the models

[5.1] Applying TNSE on Text TFIDF vectors

TFIDF vectors are sparse matrices so we have to convert it into dense matrices using todense() function

In [69]:

```
type(final_tf_idf)
```

Out[69]:

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

In [70]:

```
final_tf_idf = final_tf_idf.todense()
```

In [71]:

```
type(final_tf_idf)
```

Out[71]:

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

t-SNE of tf-idf with perplexity = 30 and n_iter = 1000

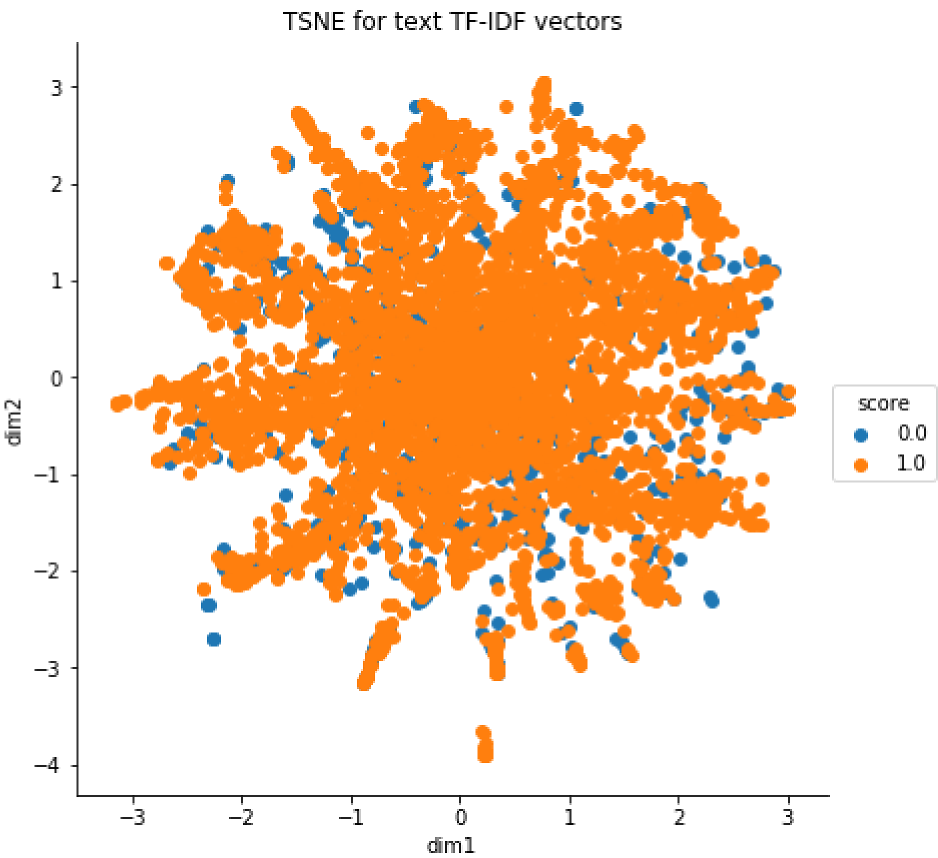
In [75]:

```
from sklearn.manifold import TSNE
model = TSNE(n_components=2, random_state=0, perplexity = 30, n_iter = 1000)
# configuring the parameteres
# the number of components = 2
# default perplexity = 30
# default learning rate = 200
# default Maximum number of iterations for the optimization = 1000

tsne_datAtfidf = model.fit_transform(final_tf_idf)


# creating a new data frame which help us in ploting the result data
tsne_datAtfidf = np.vstack((tsne_datAtfidf.T, final['Score'])).T
tsnetfidf_df = pd.DataFrame(data=tsne_datAtfidf, columns=("dim1", "dim2", "score"))

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



OBSERVATION:-

1> From above plots it is observed that around (85-90)% points are overlapping so it is very difficult to classify the polarity of the reviews i.e positive or negative.

2> It is hard to classify points using a simple linear model.

[5.2] Applying TNSE on Text Avg W2V vectors

t-SNE of avg W2V with perplexity = 30 and n_iter = 1000

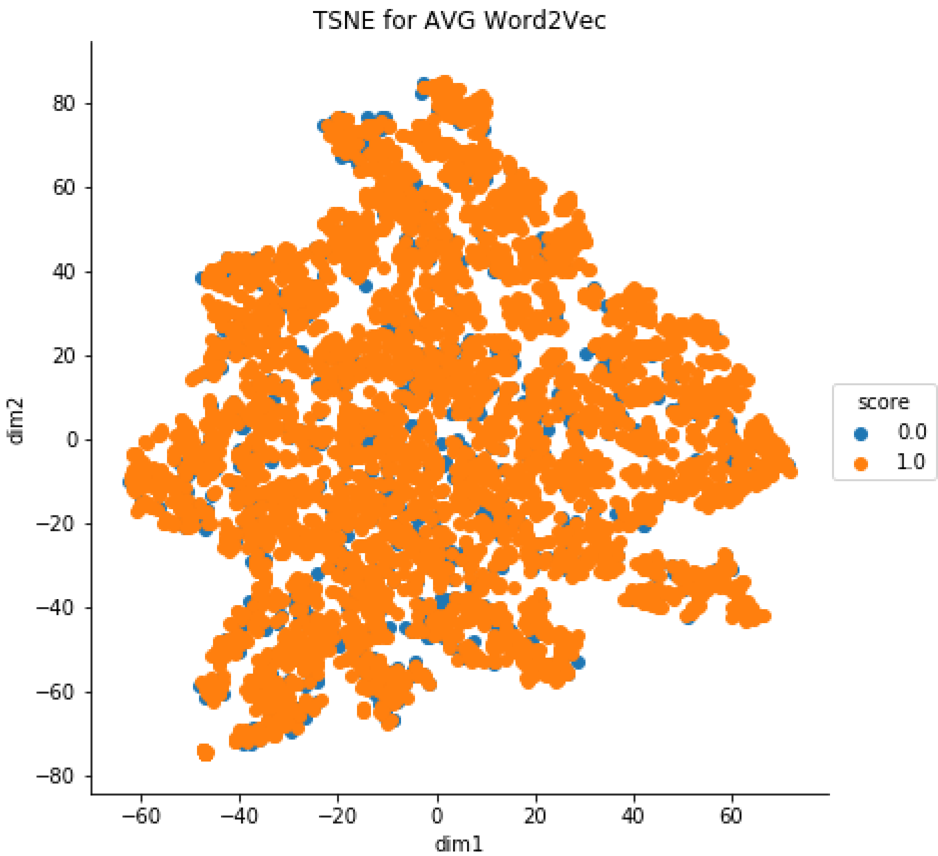
In [74]:

```
from sklearn.manifold import TSNE
model = TSNE(n_components=2, random_state=0, perplexity = 30, n_iter = 1000)
# configuring the parameteres
# the number of components = 2
# default perplexity = 30
# default learning rate = 200
# default Maximum number of iterations for the optimization = 1000

tsne_datAwv = model.fit_transform(sent_vectors)


# creating a new data frame which help us in ploting the result data
tsne_datAwv = np.vstack((tsne_datAwv.T, final['Score'])).T
tsneAwv_df = pd.DataFrame(data=tsne_datAwv, columns=("dim1", "dim2", "score"))

# Ploting the result of tsne
sns.FacetGrid(tsneAwv_df, hue="score", size=6).map(plt.scatter, 'dim1', 'dim2').add_leg
end()
plt.title("TSNE for AVG Word2Vec")
plt.show()
```



OBSERVATION:-

1> From above plots it is observed that around 90% and above points are overlapping so it is very difficult to classify the polarity of the reviews i.e positive or negative.

2> It is hard to classify points using a simple linear line or a plane

[5.3] Applying TNSE on Text TFIDF weighted W2V vectors

t-SNE of tf-idf weighted W2V with perplexity = 30 and n_iter = 1000

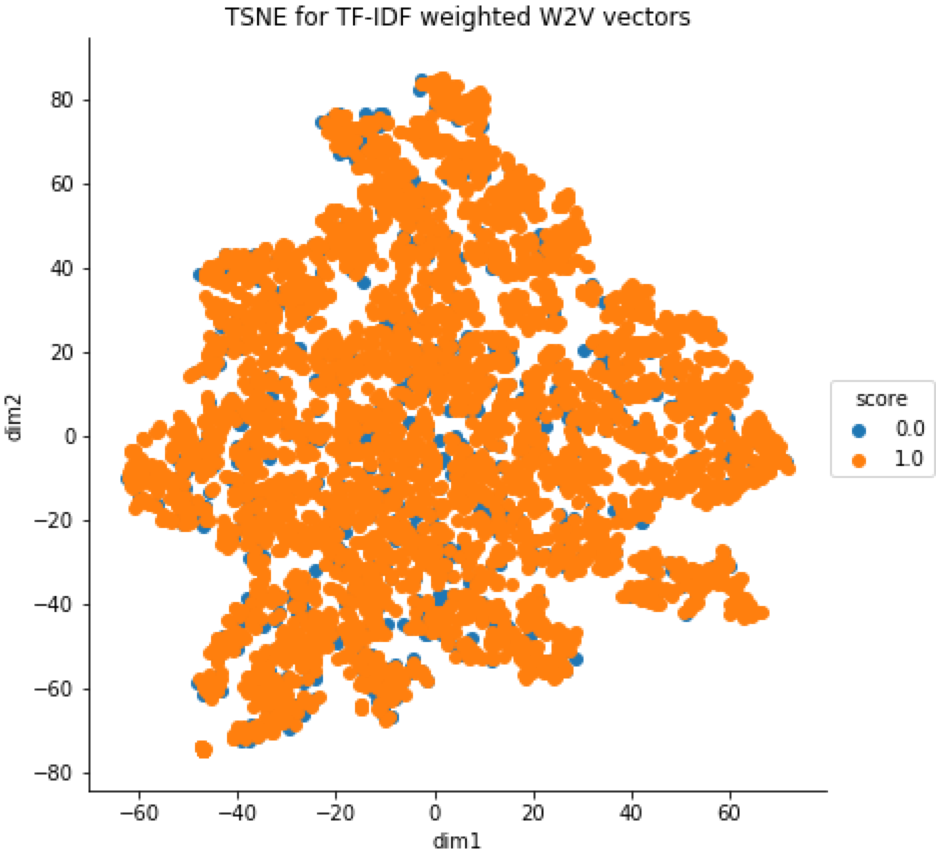
In [76]:

```
from sklearn.manifold import TSNE
model = TSNE(n_components=2, random_state=0, perplexity = 30, n_iter = 1000)
# configuring the parameteres
# the number of components = 2
# default perplexity = 30
# default learning rate = 200
# default Maximum number of iterations for the optimization = 1000

tsne_datfidfw2v = model.fit_transform(tfidf_sent_vectors)


# creating a new data frame which help us in ploting the result data
tsne_datfidfw2v = np.vstack((tsne_datfidfw2v.T, final['Score'])).T
tsnefidfw2v_df = pd.DataFrame(data=tsne_datfidfw2v, columns=("dim1", "dim2", "score"))

# Ploting the result of tsne
sns.FacetGrid(tsneAwv_df, hue="score", size=6).map(plt.scatter, 'dim1', 'dim2').add_leg
end()
plt.title("TSNE for TF-IDF weighted W2V vectors")
plt.show()
```



OBSERVATION:-

1> From above plots it is observed that around 90% and above points are overlapping so it is very difficult to classify the polarity of the reviews i.e positive or negative.

2> It is hard to classify points using a simple linear model.

[6] Conclusions

As we can see all above 4 t-SNE plots it is clearly shown that the data is highly overlapping due to which it is impossible to classify the positive and negative reviews/points with implementation a linear model or line or a plane.

As it is clear that our data is not linearly saparable so we will make a model which will easily saparate our data.

We have taken 5k points due to low system configration if we will take more points then result can be better.

Due to low system configtation and t-SNE do lot of internal computations which take a lot of time and memory we have plot t-SNE on default perplexity = 30 ,default learning rate = 200,default number of iterations=1000 for better analysis we can plot it for more different parameters.

In [ ]:

```
84.5,83,87,86,70.66.66,73
```