

# 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 - unique 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 considered a positive review. A review of 1 or 2 could be considered negative. A review of 3 is neutral 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 is above 3, then the recommendation will be set to "positive". Otherwise, it will be set to "negative".

In [1]:

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

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
```

```

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

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

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

from tqdm import tqdm
import os

```

## [1]. Reading Data

In [2]:

```

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

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

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

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

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

```

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

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	T
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017

In [3]:

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

In [4]:

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

(80668, 7)

Out[4]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXJB9	B005HG9ET0	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B007Y59HVM	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ET0	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBE1U	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [5]:

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

Out[5]:

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

In [6]:

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

Out[6]:

393063

# Exploratory Data Analysis

## [2] Data Cleaning: Deduplication

## [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 [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	HelpfulnessDenominator	Score	
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995'
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995'
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995'
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995'
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995'

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 delete 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 calculations

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	HelpfulnessDenominator	Score	
0	64422	B000MIDROQ	A161DK06JMCYF	J. E. Stephens "Jeanne"	3	1	5	12248
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	12128

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
```

### [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 observed to be better than Porter Stemming)

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

In [14]:

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

=====

I recently tried this flavor/brand and was surprised at how delicious these 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 with family and friends. I am a little disappointed that there are not, so far, very many brown chips in these bags, but the flavor is still very good. 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 haven't eaten Kettle chips before, I recommend that you try a bag before buying 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 were ordering; the other wants crispy cookies. Hey, I'm sorry; but these reviews do nobody any good beyond reminding us to look before ordering.<br /><br />These are chocolate-oatmeal cookies. If you don't like that combination, don't order this type of cookie. I find the combo quite nice, really. 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 tastes differ; so, I've given my opinion.<br /><br />Then, these are soft, chewy cookies -- as advertised. They are not "crispy" cookies, or the blurb would say "crispy," rather than "chewy." I happen to like raw cookie dough; 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 together. 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 somewhat sweet.<br /><br />So, if you want something hard and crisp, I suggest Nabisco's Ginger Snaps. If you want a cookie that's soft, chewy and tastes like 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 [15]:

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

In [16]:

```
# 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)
```

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

=====

I recently tried this flavor/brand and was surprised at how delicious these 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 with family and friends. I am a little disappointed that there are not, so far, very many brown chips in these bags, but the flavor is still very good. 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 haven't eaten Kettle chips before, I recommend that you try a bag before buying 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 were ordering; the other wants crispy cookies. Hey, I'm sorry; but these reviews do nobody any good beyond reminding us to look before ordering. These are chocolate-oatmeal cookies. If you don't like that combination, don't order this type of cookie. I find the combo quite nice, really. 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 tastes differ; so, I've given my opinion. Then, these are soft, chewy cookies -- as advertised. They are not "crispy" cookies, or the blurb would say "crispy," rather than "chewy." I happen to like raw cookie dough; 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 together. 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 somewhat sweet. So, if you want something hard and crisp, I suggest Nabisco's Ginger Snaps. If you want a cookie that's soft, chewy and tastes like 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. This k cup is great coffee. dcaf is very good as well

In [17]:

```
# 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 [18]:

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

Wow. So far, two two-star reviews. One obviously had no idea what they were ordering; the other wants crispy cookies. Hey, I am sorry; but these reviews do nobody any good beyond reminding us to look before ordering. These are chocolate-oatmeal cookies. If you do not like that combination, do not order this type of cookie. I find the combo quite nice, really. The oatmeal sort of "calms" the rich chocolate flavor and gives the cookie sort of a coconut-type consistency. Now let is also remember that tastes differ; so, I have given my opinion. Then, these are soft, chewy cookies -- as advertised. They are not "crispy" cookies, or the blurb would say "crispy," rather than "chewy." I happen to like raw cookie dough; however, I do not see where these taste like raw cookie dough. Both are soft, however, so is this the confusion? And, yes, they stick together. Soft cookies tend to do that. They are not individually wrapped, which would add to the cost. Oh yeah, chocolate chip cookies tend to be somewhat sweet. So, if you want something hard and crisp, I suggest Nabisco is Ginger Snaps. If you want a cookie that is soft, chewy and tastes like a combination of chocolate and oatmeal, give these a try. I am here to place my second order.

=====

In [19]:

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

Why is this \$[...] when the same product is available for \$[...] here? The Victor and traps are unreal, of course -- total fly genocide. Pretty stinky, but only right nearby.

In [20]:

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

Wow So far two two star reviews One obviously had no idea what they were ordering the other wants crisp y cookies Hey I am sorry but these reviews do nobody any good beyond reminding us to look before ordering br br These are chocolate oatmeal cookies If you do not like that combination do not order this type of cookie I find the combo quite nice really The oatmeal sort of calms the rich chocolate flavor and gives the cookie sort of a coconut type consistency Now let is also remember that tastes differ so I have given my opinion br br Then these are soft chewy cookies as advertised They are not crispy cookies or the blurb would say crispy rather than chewy I happen to like raw cookie dough however I do not see where these taste like raw cookie dough Both are soft however so is this the confusion And yes they stick together Soft cookies tend to do that They are not individually wrapped which would add to the cost Oh yeah chocolate chip cookies tend to be somewhat sweet br br So if you want something hard and crisp I suggest Nabisco is Ginger Snaps If you want a cookie that is soft chewy and tastes like a combination of chocolate and oatmeal give these a try I am here to place my second order

In [21]:

```
# 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 been removed in the 1st step
```



```

# instead of set, we have list, these edge words have been removed in the stop word
stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you'r
e", "you've",\
               "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself'
, \
               'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 't
heir',\
               'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these',
'those', \
               'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'd
o', 'does', \
               'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'whil
e', 'of', \
               'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'bef
ore', 'after',\
               'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'a
gain', 'further',\
               'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', '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", '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', "isn't", 'ma', 'mightn', "mightn't",
'mustn',\
               "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't",
'weren', "weren't", \
               'won', "won't", 'wouldn', "wouldn't"])

```

In [22]:

```

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

```

100%|██████████| 4986/4986 [00:04<00:00, 1113.52it/s]

In [23]:

```
preprocessed_reviews[1500]
```

Out[23]:

'wow far two two star reviews one obviously no idea ordering wants crispy cookies hey sorry reviews nob  
ody good beyond reminding us look ordering chocolate oatmeal cookies not like combination not order typ  
e cookie find combo quite nice really oatmeal sort calms rich chocolate flavor gives cookie sort coconu  
t type consistency let also remember tastes differ given opinion soft chewy cookies advertised not cris  
py cookies blurb would say crispy rather chewy happen like raw cookie dough however not see taste like  
raw cookie dough soft however confusion yes stick together soft cookies tend not individually wrapped w  
ould add cost oh yeah chocolate chip cookies tend somewhat sweet want something hard crisp suggest nabi  
so ginger snaps want cookie soft chewy tastes like combination chocolate oatmeal give try place second  
order'

## [3.2] Preprocess Summary

### Objective

In Summary Preprocessing phase we will perform the following steps:-

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 observed to be better than Porter Stemming)

## Basic Information about Summary

In [24]:

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

```
#checking the shape of final
print("Shape: ",final.shape)
rows,cols=final.shape
print("Total rows: ",rows)
print("Total columns: ",cols)
```

```
Shape: (4986, 10)
Total rows: 4986
Total columns: 10
```

In [25]:

```
# description of final
final.describe()
```

Out[25]:

	Id	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
count	4986.000000	4986.000000	4986.000000	4986.000000	4.986000e+03
mean	2715.507220	1.574609	2.018452	0.837946	1.295019e+09
std	1568.270918	5.057161	5.622785	0.368537	4.765973e+07
min	1.000000	0.000000	0.000000	0.000000	9.617184e+08
25%	1349.250000	0.000000	0.000000	1.000000	1.270944e+09
50%	2720.500000	0.000000	1.000000	1.000000	1.309349e+09
75%	4069.750000	2.000000	2.000000	1.000000	1.330387e+09
max	5427.000000	165.000000	168.000000	1.000000	1.351210e+09

In [26]:

```
# printing first 4 rows of final dataset
final.head(4)
```

Out[26]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
2546	2774	B00002NCJC	A196AJHU9EASJN	Alex Chaffee	0	0	1	128
2547	2775	B00002NCJC	A13RRPGE79XFFH	reader48	0	0	1	128

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
1145	1244	B00002Z754	A3B8RCEI0FXFI6	B G Chase	10	10	1	962
1146	1245	B00002Z754	A29Z5PI9BW2PU3	Robbie	7	7	1	961

In [27]:

```
# checking few Summaries
summary=final['Summary']
print('Summary:\n\n')
summ_0=summary.values[0]
print(summ_0)
print('-'*80)

summ_1000=summary.values[1000]
print(summ_1000)
print('-'*80)

summ_2000=summary.values[2000]
print(summ_2000)
print('-'*80)

summ_3000=summary.values[3000]
print(summ_3000)
print('-'*80)

summ_4000=summary.values[4000]
print(summ_4000)
print('-'*80)

#printing the last summary
summ_last=summary.values[len(summary)-1]
print(summ_last)
```

Summary:

thirty bucks?

-----  
Best sour cream & onion chip I've had

-----  
This is great stuff

-----  
Love it

-----  
Pleasantly surprised

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

## Step 1: Removing URL & HTML tags

In [28]:

```
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
summ_0 = re.sub(r"http\S+", "", summ_0)
summ_1000 = re.sub(r"http\S+", "", summ_1000)
summ_2000 = re.sub(r"http\S+", "", summ_2000)
summ_3000 = re.sub(r"http\S+", "", summ_3000)
summ_4000 = re.sub(r"http\S+", "", summ_4000)
```

```
print("After removing url:\t",summ_0)

def cleanhtml(raw_html):
    cleanr = re.compile('<.*>')
    cleantext = re.sub(cleanr, ' ', raw_html)
    cleantext= re.sub(r"http\S+", "", cleantext)
    return cleantext

print("After cleaning html tags:",cleanhtml (summ_0))
```

After removing url: thirty bucks?  
 After cleaning html tags: thirty bucks?

## Step 2: Remove any punctuations

In [29]:

```
# removingspecial symbols
# https://stackoverflow.com/questions/9662346/python-code-to-remove-html-tags-from-a-string
def remv(txt):
    cleantext=re.sub('\?', ' ',txt)
    cleantext=re.sub(r'[$.\[\]-]', '',cleantext)
    cleantext=re.sub(' ', ' ',cleantext)
    return cleantext
s0=summ_0
print("Before cleaning:",s0)
print("After cleaning:\t",remv(s0))
```

Before cleaning: thirty bucks?  
 After cleaning: thirty bucks

## 3. Removing Special Character and Numbers

In [30]:

```
# Removing numbers and special character and keeping the text only
num=summary[2700]
text_only=re.sub('[^A-Za-z]+', ' ', num)
print("Before cleaning:",num)
print("After cleaning:\t",text_only)
```

Before cleaning: All 3 of my cats refused to eat it  
 After cleaning: All of my cats refused to eat it

In [31]:

```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
s0 = re.sub("\S*\d\S*", "", num).strip()
print("Before cleaning:",num)
print("After cleaning:\t",s0)
```

Before cleaning: All 3 of my cats refused to eat it  
 After cleaning: All of my cats refused to eat it

## 4. Checking length of the word is greater than 2

In [32]:

```
#https://stackoverflow.com/questions/29689516/find-words-of-length-4-using-regular-expression
print(s0)
print('\n Printing all the words having length greater than 2:\n')
print(re.sub(r'\b[a-zA-Z]{1,2}\b', '', s0))
```

All of my cats refused to eat it

Printing all the words having length greater than 2:

All    cats refused   eat

## 5. Convert the word to lowercase

In [33]:

```
summ_lower = ' '.join(e.lower() for e in summ_4000.split() )
print('Before converting:',summ_4000)
print('After converting:',summ_lower)
```

Before converting: Pleasantly surprised

After converting: pleasantly surprised

## 6. Remove Stopwords

In [34]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you'r
e", "you've", \
                "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself'
, \
                'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 't
heir', \
                'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these',
'those', \
                'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'd
o', 'does', \
                'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'whil
e', 'of', \
                'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'bef
ore', 'after', \
                'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'a
gain', 'further', \
                'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', '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", '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', "isn't", 'ma', 'mightn', "mightn't",
'mustn', \
                "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't",
'weren', "weren't", \
                'won', "won't", 'wouldn', "wouldn't"])
```

In [35]:

```
new_txt = ' '.join(e for e in text_only.split() if e not in stopwords)
print('With stopword:\t',text_only)
print('Without stopword:',new_txt)
```

With stopword: All of my cats refused to eat it

Without stopword: All cats refused eat

## 7. Snowball Stemming the word

In [36]:

```
# http://www.nltk.org/howto/stem.html
```

```
#importing snowball stemmer
```

```

from nltk.stem.snowball import SnowballStemmer

# print(" ".join(SnowballStemmer.languages))
# danish dutch english finnish french german hungarian italian
# norwegian porter portuguese romanian russian spanish swedish
# Create a new instance of a language specific subclass.

stemmer = SnowballStemmer("english")
print(s0)
s1=re.sub(' ', '', s0)
word=s1.split(' ')
print("\n Before stemming:", word)
lst=[]
for w in word:
    lst.append(stemmer.stem(w))
print("\n After stemming:", lst)

print("\n Example:\n Before stemming: running")
print(" After stemming:", stemmer.stem('running'))

```

All of my cats refused to eat it

Before stemming: ['Allof', 'my', 'cats', 'refused', 'to', 'eat', 'it']

After stemming: ['allof', 'my', 'cat', 'refus', 'to', 'eat', 'it']

Example:

Before stemming: running

After stemming: run

In [37]:

```

# creating function for snowball stemming
print(s0)
senl=s0
def stemmings(snt):
    sn=snt.split(' ')
    list1=[]
    for w in sn:
        list1.append(stemmer.stem(w))
    return ' '.join(list1)

print(stemmings(senl))

```

All of my cats refused to eat it

all of my cat refus to eat it

In [38]:

```

# Processing all the above steps

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

```

100% |██████████| 4986/4986 [00:00<00:00, 5510.71it/s]

In [39]:

```
preprocessed_summary[0:10]
```

Out [39]:

```
['thirti buck',
 'fli begon',
 'wow make islick',
 'great product',
 'good stuff',
 'premium qualiti dog food',
 'cat love',
 'nice big piec big almond flavor',
 'summer treat fat free guilt free',
 'not buy product unless look shred coconut']
```

## Visualising the Summary

### Positive Summary

In [40]:

```
neg = final[final['Score']<3]
neg=neg[0:1000]
print(neg.shape)

pos=pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score > 3 LIMIT 1000""", con)
print(pos.head(4))
```

```
(1000, 10)
   Id  ProductId  UserId  ProfileName \
0   1  B001E4KFG0  A3SGXH7AUHU8GW  delmartian
1   3  B000LQOCH0  ABXLMWJIXXAIN  Natalia Corres "Natalia Corres"
2   5  B006K2ZZ7K  A1UQRSCLF8GW1T  Michael D. Bigham "M. Wassir"
3   6  B006K2ZZ7K  ADT0SRK1MG0EU  Twoapennything

   HelpfulnessNumerator  HelpfulnessDenominator  Score  Time \
0                      1                      1      5  1303862400
1                      1                      1      4  1219017600
2                      0                      0      5  1350777600
3                      0                      0      4  1342051200

           Summary  Text
0  Good Quality Dog Food  I have bought several of the Vitality canned d...
1  "Delight" says it all  This is a confection that has been around a fe...
2      Great taffy  Great taffy at a great price. There was a wid...
3      Nice Taffy  I got a wild hair for taffy and ordered this f...
```

### Word Cloud

In [41]:

```
# https://www.datacamp.com/community/tutorials/wordcloud-python

# importing Word Cloud
from wordcloud import WordCloud, ImageColorGenerator
text = []
for val in pos['Summary']:
    text.append(val)
#print(text)
conv=str(text)

# Create and generate a word cloud image:
wordcloud_pos = WordCloud(max_words=50, stopwords=stopwords, background_color="white").generate(conv)

# Display the generated image:
plt.imshow(wordcloud_pos, interpolation='bilinear')
plt.axis("off")
plt.title('Word Cloud of Positive Summary\n')
plt.show()
```

Word Cloud of Positive Summary





```
# create coloring from image
image_colors = ImageColorGenerator(mask)
plt.figure(figsize=[7,7])
plt.imshow(wordcloud_usa.recolor(color_func=image_colors), interpolation="bilinear")
plt.axis("off")
```

$$(-0.5, 799.5, 799.5, -0.5)$$


```
# https://www.datacamp.com/community/tutorials/wordcloud-python
# Generate a word cloud image
mask = np.array(Image.open("love.png"))
wordcloud_usa = WordCloud(stopwords=stopwords, background_color="white", mode="RGBA", max_words=1000, mask=mask).generate(conv)

# create coloring from image
image_colors = ImageColorGenerator(mask)
plt.figure(figsize=[7,7])
plt.imshow(wordcloud_usa.recolor(color_func=image_colors), interpolation="bilinear")
plt.axis("off")
```

$$(-0.5, 1999.5, 1842.5, -0.5)$$






```
) [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
```

In [45]:

```
print("some bigram feature names ", count_vect.get_feature_names()[0:10])
print('='*50)
```

```
some bigram feature names ['ability', 'able', 'able find', 'able get', 'absolute', 'absolutely', 'absolutely delicious', 'absolutely love', 'absolutely no', 'according']
=====
```

## [4.3] TF-IDF

In [46]:

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

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

```
some sample features(unique words in the corpus) ['ability', 'able', 'able find', 'able get', 'absolute', 'absolutely', 'absolutely delicious', 'absolutely 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] Word2Vec

In [47]:

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

In [48]:

```
# 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 variable 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 occurred at least 5 times
    w2v_model=Word2Vec(list_of_sentence,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 google's word2vec file, keep want_to_train_w2v = True, to train your own
w2v ")

```

```

[('alternative', 0.9921382069587708), ('snack', 0.9916977286338806), ('want', 0.9910628795623779), ('ta
sty', 0.9907669425010681), ('ahoy', 0.9906185865402222), ('enjoy', 0.9905602931976318), ('describe', 0.
9903753995895386), ('care', 0.9903718829154968), ('salty', 0.9902743101119995), ('satisfying', 0.990234
2557907104)]

```

```

[('together', 0.9994702339172363), ('form', 0.9994691610336304), ('popcorn', 0.9994255900382996), ('sta
rt', 0.9994062185287476), ('cook', 0.9994022250175476), ('dinner', 0.9993957281112671), ('bacon', 0.999
3876218795776), ('stand', 0.9993842840194702), ('lots', 0.9993777275085449), ('minute', 0.9993722438812
256)]

```

In [49]:

```

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

```

```

number of words that occurred minimum 5 times 3817
sample words ['product', 'available', 'course', 'total', 'pretty', 'stinky', 'right', 'nearby', 'used'
, 'ca', 'not', 'beat', 'great', 'received', 'shipment', 'could', 'hardly', 'wait', 'try', 'love', 'call
', 'instead', 'removed', 'easily', 'daughter', 'designed', 'printed', 'use', 'car', 'windows', 'beautif
ully', 'shop', 'program', 'going', 'lot', 'fun', 'everywhere', 'like', 'tv', 'computer', 'really', 'goo
d', 'idea', 'final', 'outstanding', 'window', 'everybody', 'asks', 'bought', 'made']

```

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

### [4.4.1.1] Avg W2v

In [50]:

```

# 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_sentence): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 3
00 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 [00:08<00:00, 584.73it/s]

#### [4.4.1.2] TFIDF weighted W2v

In [51]:

```
# 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 [52]:

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

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentence): # 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 corpus
            # sent.count(word) = tf value 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 [00:48<00:00, 102.38it/s]

## [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

In [217]:

```
# https://github.com/pavlin-polcar/fastTSNE you can try this also, this version is little faster than
sklearn
import numpy as np
from sklearn.manifold import TSNE
from sklearn import datasets
import pandas as pd
import matplotlib.pyplot as plt
from numpy.random import rand

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

tsne = TSNE(n_components=2, perplexity=30, 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'])
colors = {0:'red', 1:'blue', 2:'green'}

# https://matplotlib.org/gallery/lines_bars_and_markers/scatter_with_legend.html
n = 750
scale = 200.0 * rand(n)

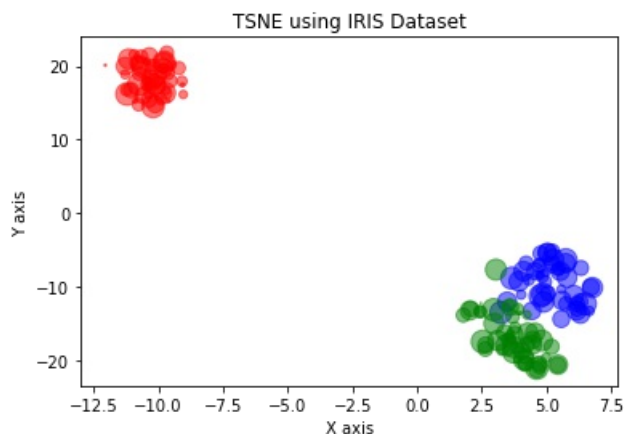
plt.scatter(for_tsne_df['Dimension_x'], for_tsne_df['Dimension_y'], s=scale, alpha=0.5, c=for_tsne_df['Score'].apply(lambda x: colors[x]))

plt.xlabel('X axis')
plt.ylabel('Y axis')

plt.title('TSNE using IRIS Dataset')

plt.show()

```



## Observation:

Red points are well separated from blue and green points.  
Blue and green points are overlapping to some extent.

## [5.1] Applying TNSE on Text BOW vectors

Please write all the code with proper documentation, and proper titles for each subsection when you plot any graph make sure you use

- Title, that describes your plot, this will be very helpful to the reader
- Legends if needed
- X-axis label
- Y-axis label

## plotting using 5000 points

In [54]:

```

#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', 'aahhs', 'aback', 'abandon', 'abates', 'abbott', 'abby', 'abdominal', 'abid  
inal', 'ability']

```
ing, '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
```

In [322]:

```
print(X_embedding1.shape)
pfinal=np.hstack((X_embedding1,final['Score'].reshape(-1,1)))
ptsne = pd.DataFrame(data=pfinal, columns=['Dimension_x','Dimension_y','Score'])
colors = {0:'red', 1:'green'}

# https://matplotlib.org/gallery/lines_bars_and_markers/scatter_with_legend.html
n = 750
scale = 200.0 * rand(n)

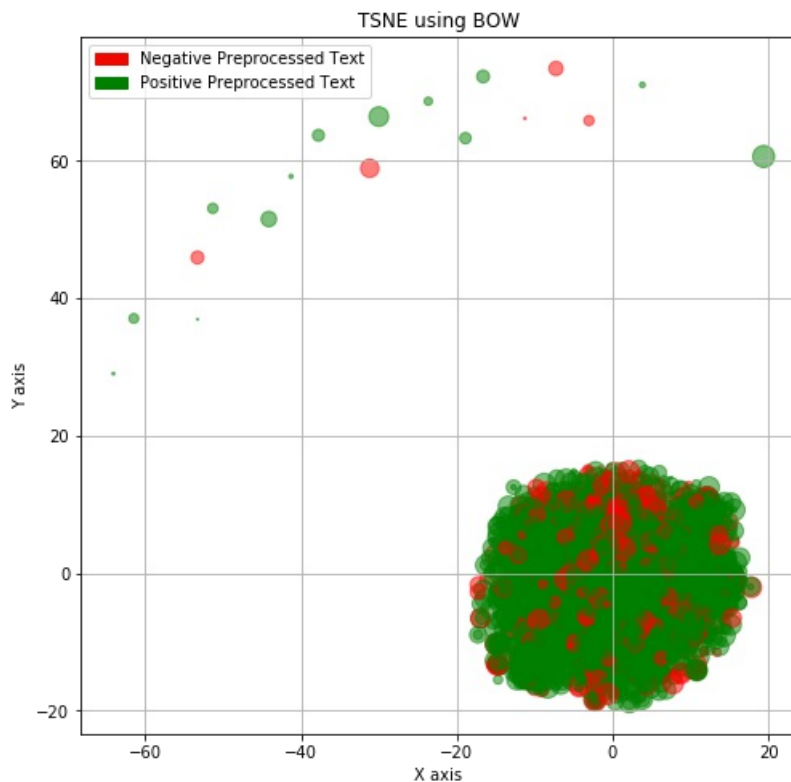
clr=ptsne['Score'].apply(lambda x: colors[x])
lbl=(set(clr))

plt.figure(figsize=(8,8))
labels = ['Negative Preprocessed Text', 'Positive preprocessed Text']
plt.scatter(ptsne['Dimension_x'], ptsne['Dimension_y'],s=scale, alpha=0.5,c=clr, label='%s\n data' % cl
r.unique())

# plt.legend(['Bond Ret','Equity Ret'], loc=0)
import matplotlib.patches as mpatches

red_patch = mpatches.Patch(color='red', label='Negative Preprocessed Text')
green_patch = mpatches.Patch(color='green', label='Positive Preprocessed Text')
plt.legend(handles=[red_patch,green_patch])
plt.grid(True)
# plt.legend()
plt.title('TSNE using BOW')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.show()
```

(4986, 2)



**Observation:**



Most of the points are very close to origin.

Only few points are far away from the group of dense points.

In [64]:

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

#removing stop words like "not" should be avoided before building n-grams
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation
# http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
# you can choose these numebrs min_df=10, max_features=5000, of your choice
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=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_counts.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

In [ ]:

```
xgram=final_bigram_counts.toarray()

tsne=TSNE(n_components=2,perplexity=30,learning_rate=200)
X_embeddings = tsne.fit_transform(xgram)
```

In [329]:

```
print(X_embeddings.shape)
pfinal1=np.hstack((X_embeddings,final['Score'].reshape(-1,1)))
ptsnel = pd.DataFrame(data=pfinal1, columns=['Dimension_x','Dimension_y','Score'])
colors = {0:'red', 1:'green'}

# https://matplotlib.org/gallery/lines_bars_and_markers/scatter_with_legend.html
n = 750
scale = 200.0 * rand(n)

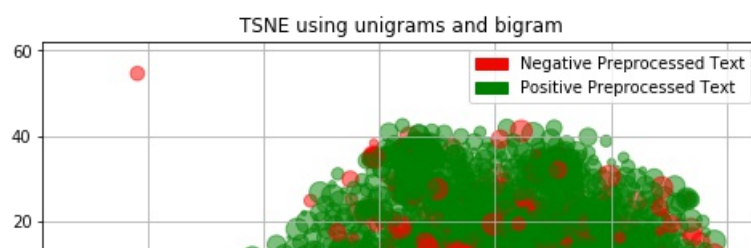
plt.figure(figsize=(8,8))
#labels = ['Negative Preprocessed Text', 'Positive preprocessed Text']
plt.scatter(ptsnel['Dimension_x'], ptsnel['Dimension_y'],s=scale, alpha=0.5,c=ptsnel['Score'].apply(lambda
bda x: colors[x]),marker='o')

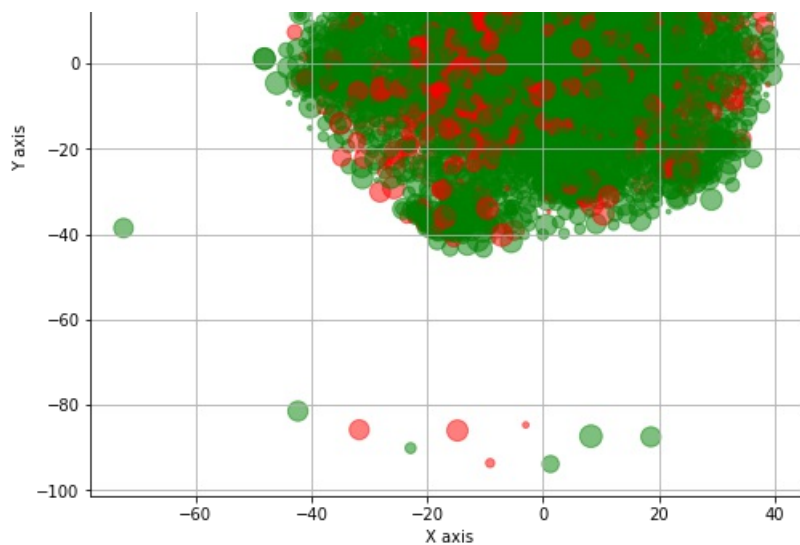
# https://matplotlib.org/users/legend_guide.html
import matplotlib.patches as mpatches

red_patch = mpatches.Patch(color='red', label='Negative Preprocessed Text')
green_patch = mpatches.Patch(color='green', label='Positive Preprocessed Text')
plt.legend(handles=[red_patch,green_patch])
plt.grid(True)

plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.title('TSNE using unigrams and bigram')
plt.show()
```

(4986, 2)





### Observation:

Most of the points are very close to origin.

Only few points are far away from the group of dense points.

## [5.2] Applying TNSE on Text TFIDF vectors

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

In [69]:

```
# taking 5000 points only

tf_idf_vect1 = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect1.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect1.get_feature_names()[0:10])
print('='*50)

final_tf_idf1 = tf_idf_vect1.transform(preprocessed_reviews)

print("the shape of out text TFIDF vectorizer ",final_tf_idf1.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_tf_idf1.get_shape()[1])
```

some sample features(unique words in the corpus) ['ability', 'able', 'able find', 'able get', 'absolute', 'absolutely', 'absolutely delicious', 'absolutely love', 'absolutely no', 'according']

=====

the shape of out text TFIDF vectorizer (4986, 3144)

the number of unique words including both unigrams and bigrams 3144

In [70]:

```
from sklearn.manifold import TSNE

tfidf3=final_tf_idf1.todense()

tsne3=TSNE(n_components=2,perplexity=30,learning_rate=200)
X_embedding3 = tsne3.fit_transform(tfidf3)

print(X_embedding3.shape)
```

(4986, 2)

In [330]:

```
pfinal2=np.hstack((X_embedding3,final['Score'].reshape(-1,1)))
```

```
ptsne2 = pd.DataFrame(data=pfinal2, columns=['Dimension_x', 'Dimension_y', 'Score'])
colors = {0:'red', 1:'blue'}

# https://matplotlib.org/gallery/lines_bars_and_markers/scatter_with_legend.html
n = 750
scale = 200.0 * rand(n)

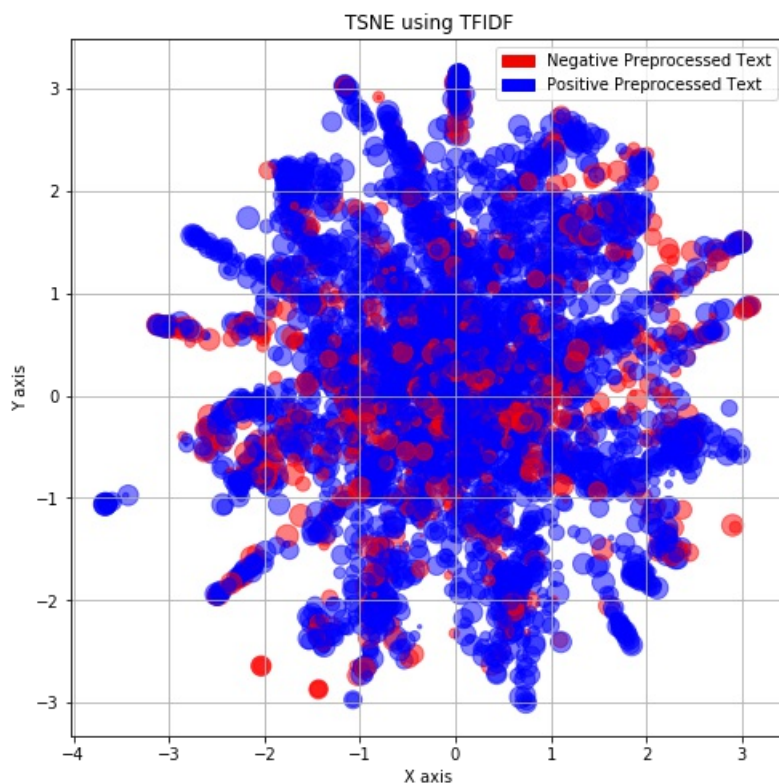
plt.figure(figsize=(8,8))
#labels = ['Negative Preprocessed Text', 'Positive preprocessed Text']
plt.scatter(ptsne2['Dimension_x'], ptsne2['Dimension_y'], s=scale, alpha=0.5, c=ptsne2['Score'].apply(lambda x: colors[x]))

# https://matplotlib.org/users/legend_guide.html
import matplotlib.patches as mpatches

red_patch = mpatches.Patch(color='red', label='Negative Preprocessed Text')
blue_patch = mpatches.Patch(color='blue', label='Positive Preprocessed Text')
plt.legend(handles=[red_patch, blue_patch])

plt.grid(True)

plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.title('TSNE using TFIDF')
plt.show()
```



### Observation:

Most of the blue and red points are overlapping with each other.  
Most of the points lies much close to origin.

## [5.3] Applying TNSE on Text Avg W2V vectors

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

In [237]:

```
# 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/0B7XkCwpI5KDYNlNUTTlSS2lpQmM/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 variable 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 occurred atleast 5 times
    w2v_model=Word2Vec(list_of_sentence,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_train_w2v = True, to train your own
w2v ")

```

```

[('regular', 0.9939302802085876), ('alternative', 0.9932116270065308), ('calorie', 0.9923950433731079),
('ahoy', 0.9920624494552612), ('healthier', 0.9919491410255432), ('snack', 0.991835355758667), ('tasty'
, 0.9917206764221191), ('healthy', 0.991540253162384), ('care', 0.9915015697479248), ('hide', 0.9914788
007736206)]
=====

```

In [240]:

```

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

```

```

number of words that occurred minimum 5 times 3817
sample words ['product', 'available', 'course', 'total', 'pretty', 'stinky', 'right', 'nearby', 'used'
, 'ca', 'not', 'beat', 'great', 'received', 'shipment', 'could', 'hardly', 'wait', 'try', 'love', 'call'
, 'instead', 'removed', 'easily', 'daughter', 'designed', 'printed', 'use', 'car', 'windows', 'beautif
ully', 'shop', 'program', 'going', 'lot', 'fun', 'everywhere', 'like', 'tv', 'computer', 'really', 'goo
d', 'idea', 'final', 'outstanding', 'window', 'everybody', 'asks', 'bought', 'made']

```

In [241]:

```

# average Word2Vec
# compute average word2vec for each review.
sent_vectors1 = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentence): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 3
00 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_vectors1.append(sent_vec)
print(len(sent_vectors1))
print(len(sent_vectors1[0]))

```

100%|██████████| 4986/4986 [00:09<00:00, 551.37it/s]

4986

50

In [242]:

```
#w2vec_model=Word2Vec(list_of_sentence,min_count=5,size=50, workers=4)
```

```
w2v_tsne1=TSNE(n_components=2,perplexity=30,learning_rate=200)
X_w2v1 = w2v_tsne1.fit_transform(sent_vectors1)
```

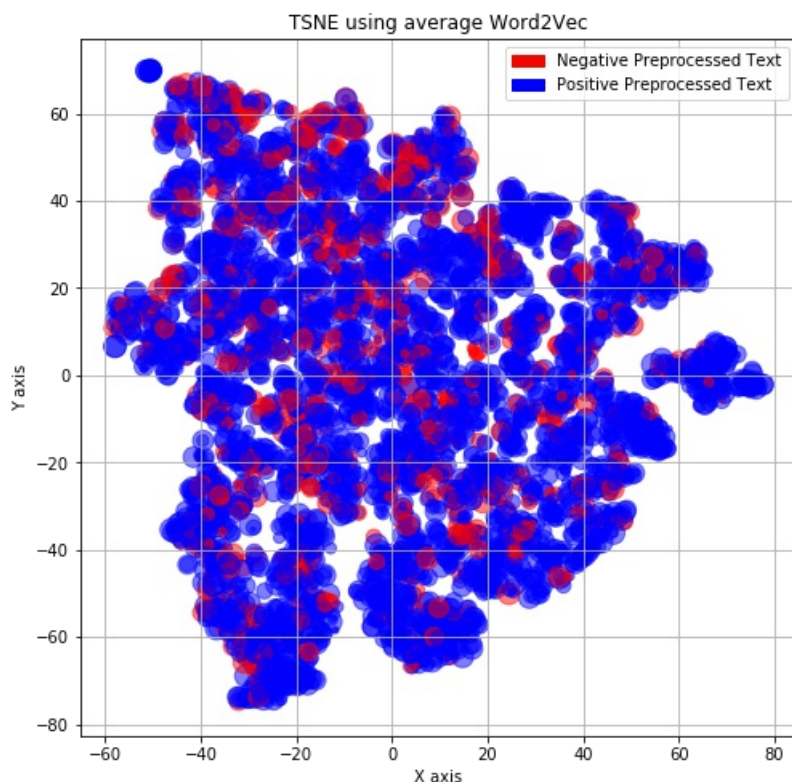
In [331]:

```
pfinal3=np.hstack((X_w2v1,final['Score'].reshape(-1,1)))
ptsne3 = pd.DataFrame(data=pfinal3, columns=['Dimension_x','Dimension_y','Score'])
colors = {0:'red', 1:'blue'}

# https://matplotlib.org/gallery/lines_bars_and_markers/scatter_with_legend.html
n = 750
scale = 200.0 * rand(n)

plt.figure(figsize=(8,8))
#labels = ['Negative Preprocessed Text', 'Positive preprocessed Text']
plt.scatter(ptsne3['Dimension_x'], ptsne3['Dimension_y'],s=scale, alpha=0.5,c=ptsne3['Score'].apply(lambda x: colors[x]))
plt.grid(True)
# https://matplotlib.org/users/legend_guide.html
import matplotlib.patches as mpatches

red_patch = mpatches.Patch(color='red', label='Negative Preprocessed Text')
blue_patch = mpatches.Patch(color='blue', label='Positive Preprocessed Text')
plt.legend(handles=[red_patch,blue_patch])
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.title('TSNE using average Word2Vec')
plt.show()
```



Observation:

Both red and blue points overlapping each other.  
Almost all the point lies close to origin.

## [5.4] Applying TNSE on Text TFIDF weighted W2V vectors

In [0]:

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

In [244]:

```
# 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 [245]:

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

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentence): # 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 corpus
            # sent.count(word) = tf value 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 [00:56<00:00, 88.34it/s]

In [247]:

```
tfidf_tsne=TSNE(n_components=2,perplexity=30,learning_rate=200)
X_tfidf = tfidf_tsne.fit_transform(tfidf_sent_vectors)
```

In [332]:

```
pfinal4=np.hstack((X_tfidf,final['Score'].reshape(-1,1)))
ptsne4 = pd.DataFrame(data=pfinal4, columns=['Dimension_x','Dimension_y','Score'])
colors = {0:'red', 1:'blue'}

# https://matplotlib.org/gallery/lines_bars_and_markers/scatter_with_legend.html
n = 750
scale = 200.0 * rand(n)

plt.figure(figsize=(8,8))
#labels = ['Negative Preprocessed Text', 'Positive preprocessed Text']
cs=plt.scatter(ptsne4['Dimension_x'], ptsne4['Dimension_y'],s=scale, alpha=0.5,c=ptsne4['Score'].apply(
    lambda x: colors[x]))
```



```

lambda x: colors[x]))
# https://matplotlib.org/users/legend_guide.html
import matplotlib.patches as mpatches

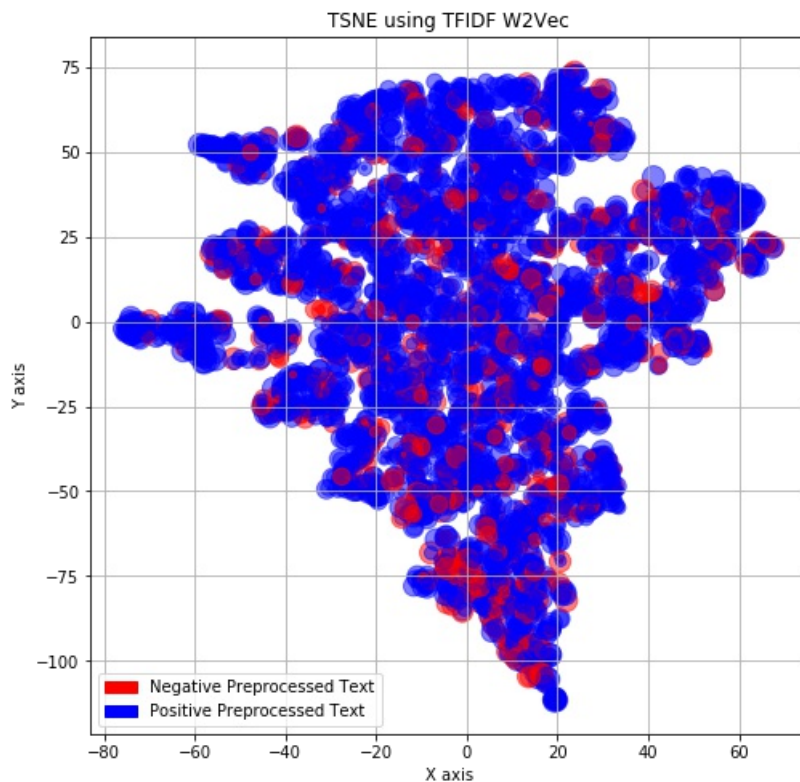
red_patch = mpatches.Patch(color='red', label='Negative Preprocessed Text')
blue_patch = mpatches.Patch(color='blue', label='Positive Preprocessed Text')
plt.legend(handles=[red_patch,blue_patch])

plt.grid(True)

plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.title('TSNE using TFIDF W2Vec')

plt.show()

```



### Observation:

Both red and blue points overlapping each others.  
Almost all the point lies close to origin.

## [6] Conclusions

Write few sentence about the results that you got and observation that you did from the analysis

- TSNE plots are highly useful in visualising high dimensional data.
- We have plotted tsne plots for BOW, TFIDF, AVG W2Vec, TFIDF W2Vec by taking 5000 points.
- In all the plots, Positive and Negative preprocessed text are overlapping with each other. Only few points are far away from the group of dense points.
- In all the TSNE plots, the points are closer to origin.