Amazon Fine Food Reviews Analysis

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

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454
Number of users: 256,059
Number of products: 74,258
Timespan: Oct 1999 - Oct 2012
Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Objective:

Given a review, determine whether the review is positive (Rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative?

[Ans] We could use the Score/Rating. A rating of 4 or 5 could be cosnidered a positive review. A review of 1 or 2 could be considered negative. A review of 3 is nuetral and ignored. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score id above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [209]:

```
%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 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 [210]:

```
# 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""", co
n)
# 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[210]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	130386240
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	134697600

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	121901760(

In [211]:

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

In [212]:

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

```
(80668, 7)
```

Out[212]:

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

In [213]:

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

Out[213]:

	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 [214]:

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

Out[214]:

```
393063
```

Exploratory Data Analysis

[2] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

In [215]:

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

Out[215]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(

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 [216]:

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='qui
cksort', na_position='last')
```

In [217]:

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

Out[217]:

```
(4986, 10)
```

In [218]:

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

Out[218]:

```
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 [219]:

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

display.head()
```

Out[219]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892ε
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	12128832

In [220]:

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

In [221]:

```
#Before starting the next phase of preprocessing lets see the number of entries left
print(final.shape)
label=final['Score']

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

```
(4986, 10)
```

Out[221]:

```
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 obsereved to be better than Porter Stemming)

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

In [222]:

```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 genocide. Pretty stinky, but only right nearby.
==================================================
I recently tried this flavor/brand and was surprised at how delicious these chips are.  The best t
hing was that there were a lot of "brown" chips in the bsg (my favorite), so I bought some more th
rough amazon and shared with family and friends.  I am a little disappointed that there are not, s
o 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 flav
or is better.  If you haven't eaten Kettle chips before, I recommend that you try a bag before buy
ing 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 com
bination, 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 li
ke raw cookie dough.  Both are soft, however, so is this the confusion?  And, yes, they stick toge
ther.  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 Nabiso's Ginger Snaps.  If you want a cookie that's soft, chew
y 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.  d
caf is very good as well
==================================================
```

In [223]:

```
# 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 M3
80 and M502 traps are unreal, of course -- total fly genocide. Pretty stinky, but only right nearb
y.
```

In [224]:

```
# 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 t
hing was that there were a lot of "brown" chips in the bsg (my favorite), so I bought some more th
rough amazon and shared with family and friends.  I am a little disappointed that there are not, s
o 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 flav
or is better.  If you haven't eaten Kettle chips before, I recommend that you try a bag before buy
ing 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, do
n'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 te
nd 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 Nabiso'
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 [225]:

```
# 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 [226]:

```
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 t
o 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 a
re soft, chewy cookies -- as advertised.  They are not "crispy" cookies, or the blurb would say "c
rispy," 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 st
ick together.  Soft cookies tend to do that.  They are not individually wrapped, which would add t
o the cost.  Oh yeah, chocolate chip cookies tend to be somewhat sweet.<br /><br />So, if you want
something hard and crisp, I suggest Nabiso is Ginger Snaps.  If you want a cookie that is soft, ch
ewy and tastes like a combination of chocolate and oatmeal, give these a try.  I am here to place
my second order.
==================================================
```

In [227]:

```
#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  a
nd  traps are unreal, of course -- total fly genocide. Pretty stinky, but only right nearby.
```

In [228]:

```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^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
crispy cookies Hey I am sorry but these reviews do nobody any good beyond reminding us to look bef
ore 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 ch
ocolate flavor and gives the cookie sort of a coconut type consistency Now let is also remember th
at 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 c
ookie 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 indivi
dually 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 Nabiso 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 [229]:

```
# 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', 'ourselves', 'you', "y
ou'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', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', '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', "do
esn'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 [230]:

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

```
100%|██████████████████████████████████████████████████| 4986/4986
[00:04<00:00, 1148.42it/s]
```

In [231]:

```
preprocessed_reviews[250]
```

Out[231]:

```
'definitely best snack mix ever got buddies many movie nights deployed iraq'
```

[3.2] Preprocess Summary

In [119]:

```
## preprocessing for review summary.
## using the same above code for performing preprocessing of review summary
preprocessed_reviews = []
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 stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|████████████████████████████████████████████████████████████████| 4986/4986
[00:02<00:00, 1725.33it/s]
```

In [120]:

```
#printing summary review after performing preprocessing
preprocessed_reviews[100]
```

Out[120]:

```
'not minced ground beef'
```

[4] Featurization

[4.1] BAG OF WORDS

In [232]:

```
#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_bow = count_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_counts_bow))
print("the shape of out text BOW vectorizer ",final_counts_bow.get_shape())
print("the number of unique words ", final_counts_bow.get_shape()[1])
```

```
some feature names  ['aa', 'aahhhs', 'aback', 'abandon', 'abates', 'abbott', '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 [233]:

```
#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_s
hape()[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 [234]:

```
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 [235]:

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

In [236]:

```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=Tr
ue)
        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 ")
```

```
[('excellent', 0.9955097436904907), ('overall', 0.9951634407043457), ('alternative',
0.995011568069458), ('amazing', 0.9949752688407898), ('watching', 0.9948762059211731),
('terrific', 0.9948509931564331), ('thicker', 0.9947086572647095), ('looking',
0.9946579933166504), ('satisfying', 0.9946224689483643), ('wanting', 0.9945143461227417)]
```

```
==================================================
[('turned', 0.9995012283325195), ('level', 0.9994671940803528), ('device', 0.99940425157547),
('varieties', 0.9993860125541687), ('popcorn', 0.9993807077407837), ('note', 0.9993668794631958),
('wow', 0.9993412494659424), ('must', 0.9993234276771545), ('nearly', 0.9993224143981934),
('beef', 0.999319851398468)]
```

In [237]:

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

```
number of words that occured minimum 5 times  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stinky', 'right', 'nearby', '
used', 'ca', 'not', 'beat', 'great', 'received', 'shipment', 'could', 'hardly', 'wait', 'try', 'lo
ve', 'call', 'instead', 'removed', 'easily', 'daughter', 'designed', 'printed', 'use', 'car', 'win
dows', 'beautifully', 'shop', 'program', 'going', 'lot', 'fun', 'everywhere', 'like', 'tv',
'computer', 'really', 'good', 'idea', 'final', 'outstanding', 'window', 'everybody', 'asks',
'bought', 'made']
```

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

[4.4.1.1] Avg W2v

In [238]:

```python
# 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
[00:08<00:00, 595.57it/s]
```

```
4986
50
```

[4.4.1.2] TFIDF weighted W2v

In [239]:

```python
# 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 [240]:

```python
# 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_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
:53<00:00, 93.39it/s]
```

In [241]:

```
type(tfidf_sent_vectors)
```

Out[241]:

```
list
```

[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 [242]:

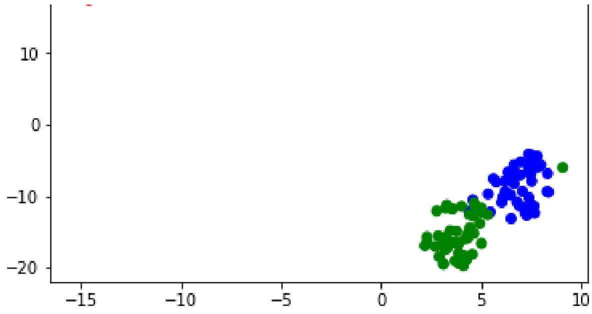
```
# https://github.com/pavlin-policar/fastTSNE you can try this also, this version is little faster
than sklearn
import numpy as np
from sklearn.manifold import TSNE
from sklearn import datasets
import pandas as pd
import matplotlib.pyplot as plt

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'}
plt.scatter(for_tsne_df['Dimension_x'], for_tsne_df['Dimension_y'], c=for_tsne_df['Score'].apply(la
mbda x: colors[x]))
plt.show()
```



[5.1] Applying TNSE on Text BOW vectors

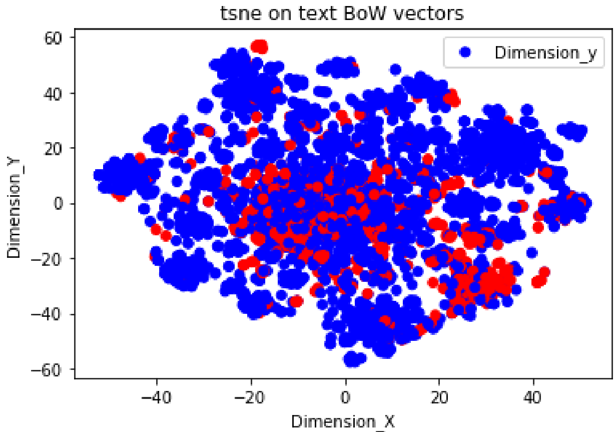
In [243]:

```
#initial label is of type series ,convert it to array type so that we can add it with X_embedding
below
label=np.array(label)
```

In [123]:

```
#As the team applied tsne on iris dataset ,i will apply it in the same way on text review after co
nverting it to vectors using BoW
#importing all reuired libraries/modules
import numpy as np
from sklearn.manifold import TSNE
from sklearn import datasets
import pandas as pd
import matplotlib.pyplot as plt
#creating a tsne model with number of number of components and perplexity
tsne = TSNE(n_components=2, perplexity=30, learning_rate=200)
#final_counts is sparse matrix converting it to dense matrix
X_embedding = tsne.fit_transform(final_counts_bow.toarray())
# 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,label.reshape(-1,1)))
for_tsne_df = pd.DataFrame(data=for_tsne, columns=['Dimension_x','Dimension_y','Score'])
colors = {0:'red',1:'blue'}
plt.scatter(for_tsne_df['Dimension_x'], for_tsne_df['Dimension_y'], c=for_tsne_df['Score'].apply(la
mbda x: colors[x]))
plt.title('tsne on text BoW vectors')
plt.legend(loc='best')
plt.xlabel('Dimension_X')
plt.ylabel('Dimension_Y')
plt.show()
```



[5.1] Applying TNSE on Text TFIDF vectors

In [131]:

```
type(final_tf_idf)#check type of matrix
```

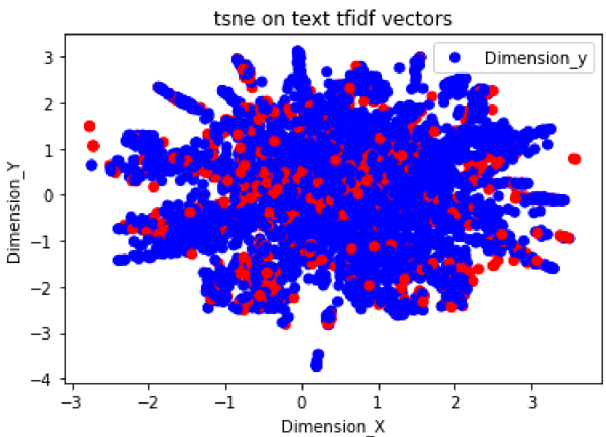
Out[131]:

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

In [208]:

```
#As the team applied tsne on iris dataset ,i will apply it in the same way on text review after co
nverting it to vectors using TF-IDF
#importing all reuired libraries/modules
import numpy as np
from sklearn.manifold import TSNE
from sklearn import datasets
import pandas as pd
import matplotlib.pyplot as plt
#creating a tsne model with number of number of components and perplexity
tsne = TSNE(n_components=2, perplexity=30, learning_rate=200)
#final_tf_idf is a sparse matrix converting it to dense matrix
X_embedding = tsne.fit_transform(final_tf_idf.toarray())
# 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,label.reshape(-1,1)))
for_tsne_df = pd.DataFrame(data=for_tsne, columns=['Dimension_x','Dimension_y','Score'])
colors = {0:'red',1:'blue'}
plt.scatter(for_tsne_df['Dimension_x'], for_tsne_df['Dimension_y'], c=for_tsne_df['Score'].apply(la
mbda x: colors[x]))
plt.title('tsne on text tfidf vectors')
plt.legend(loc='best')
plt.xlabel('Dimension_X')
plt.ylabel('Dimension_Y')
plt.show()
```



[5.3] Applying TNSE on Text Avg W2V vectors

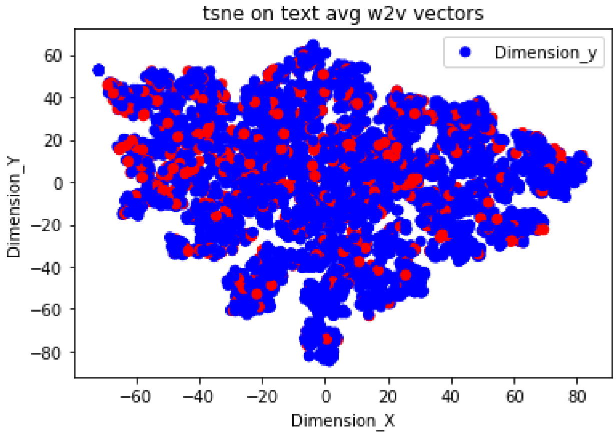
In [202]:

```
sent_vectors=np.array(sent_vectors)
label=np.array(label)
```

In [206]:

```
#As the team applied tsne on iris dataset ,i will apply it in the same way on text review after co
nverting it to vectors using BoW
#importing all reuired libraries/modules
import numpy as np
from sklearn.manifold import TSNE
from sklearn import datasets
import pandas as pd
import matplotlib.pyplot as plt
#creating a tsne model with number of number of components and perplexity
tsne = TSNE(n_components=2, perplexity=30, learning_rate=200)
#final_counts is sparse matrix converting it to dense matrix
X_embedding = tsne.fit_transform(sent_vectors)
# 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,label.reshape(-1,1)))
for_tsne_df = pd.DataFrame(data=for_tsne, columns=['Dimension_x','Dimension_y','Score'])
colors = {0:'red',1:'blue'}
plt.scatter(for_tsne_df['Dimension_x'], for_tsne_df['Dimension_y'], c=for_tsne_df['Score'].apply(la
mbda x: colors[x]))
plt.title('tsne on text avg w2v vectors')
plt.legend(loc='best')
plt.xlabel('Dimension_X')
plt.ylabel('Dimension_Y')
plt.show()
```



[5.4] Applying TNSE on Text TFIDF weighted W2V vectors

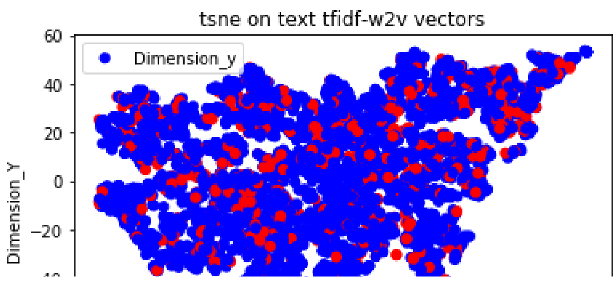
In [ ]:

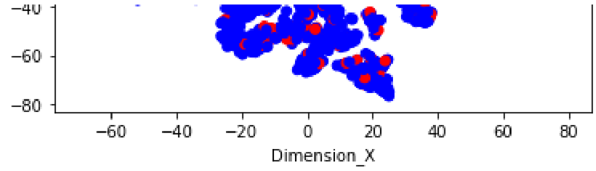
```
tfidf_sent_vectors=np.array(tfidf_sent_vectors)
```

In [205]:

```
#As the team applied tsne on iris dataset ,i will apply it in the same way on text review after co
nverting it to vectors using BoW
#importing all reuired libraries/modules
import numpy as np
from sklearn.manifold import TSNE
from sklearn import datasets
import pandas as pd
import matplotlib.pyplot as plt
#creating a tsne model with number of number of components and perplexity
tsne = TSNE(n_components=2, perplexity=30, learning_rate=200)
#final_counts is sparse matrix converting it to dense matrix
X_embedding = tsne.fit_transform(tfidf_sent_vectors)
# 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,label.reshape(-1,1)))
for_tsne_df = pd.DataFrame(data=for_tsne, columns=['Dimension_x','Dimension_y','Score'])
colors = {0:'red',1:'blue'}
plt.scatter(for_tsne_df['Dimension_x'], for_tsne_df['Dimension_y'], c=for_tsne_df['Score'].apply(la
mbda x: colors[x]))
plt.title('tsne on text tfidf-w2v vectors')
plt.legend(loc='best')
plt.xlabel('Dimension_X')
plt.ylabel('Dimension_Y')
plt.show()
```





[6] Conclusions

1. -Above we have 4 plots of tsne on review text of amazon data with only 5000 data points
2. -first 5000 data set is divided as positive 4178 and negetive 808 thats why we have more blue points and less red points in all four plots
3. -four plots when text converted to vector using BoW,TFIDF,AVG-W2V and TFIDF-W2V
4. -each of the plot has perplexity value of 30
5. -In the first plot which is with BoW,blue points are clustured together and there is gap in clusters and red points formed small clusters and overlapped with blue points
6. -plot with tfidf, most of the blue points are in middle region and there is no gap left, and at border it is like straight line type clusters and red points formed small small clusters and spread all over blue points
7. -plot with avg w2v,points are somewhat at top side and there is little little gaps between blue points and red points formed small clusters and spread/overlapped with blue points but on upper side
8. -plot with tfidf-w2v,there is no much difference compared to third plot only the shape is different otherwise it seems like point distribution is almost same
9. -these osbservation made with only 5k data set and running tsne algorithm only few times thats why we are not getting results properly , but when we work with large data sets on computers with good ram capabilities we will get better results