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 Score/Rating. A rating of 4 or 5 can be cosnidered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered nuetral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it is 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 wil be set to "positive". Otherwise, it will be set to "negative".

In [1]:

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


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler
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
from scipy.sparse import csr_matrix
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
```

```
D:\AAnaconda\lib\site-packages\gensim\utils.py:1212: UserWarning: detected Windows; aliasing chunkize t
o chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [2]:

```
# using 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 100000""", con)

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negative rating(0).
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 (100000, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976
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	B005ZBZLT4	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ESG	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B005ZBZLT4	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ESG	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBEV0	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	B001ATMQK2	undertheshrine "undertheshrine"	1296691200	5	I bought this 6 pack because for the price tha...	5

In [6]:

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

Out[6]:

```
393063
```

[2] Exploratory Data Analysis

[2.1] 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 it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that

ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8)

ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

In [8]:

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

In [9]:

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

Out[9]:

```
(87775, 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]:

```
87.775
```

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

In [11]:

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

display.head()
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
0	64422	B000MIDROQ	A161DK06JJMCYF	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)
```

```
print(final.shape)

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

```
(87773, 10)
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

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 [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)
```

```
I bought a few of these after my apartment was infested with fruit flies. After only a few hours, the t
rap had &quot;attracted&quot; many flies and within a few days they were practically gone. This may not
be a long term  solution, but if flies are driving you crazy, consider buying this. One  caution- the s
urface is very sticky, so try to avoid touching it.
==================================================
I have made these brownies for family and for a den of cub scouts and no one would have known they were
gluten free and everyone asked for seconds!  These brownies have a fudgy texture and have bits of choco
late chips in them which are delicious.  I would say the mix is very thick and a little difficult to wo
rk with.  The cooked brownies are slightly difficult to cut into very neat edges as the edges tend to c
rumble a little and I would also say that they make a slightly thinner layer of brownies than most of t
he store brand gluten containing but they taste just as good, if not better.  Highly recommended!<br />
<br />(For those wondering, this mix requires 2 eggs OR 4 egg whites and 7 tbs melted butter to prepare
.  They do have suggestions for lactose free and low fat preparations)
==================================================
I love Pretzels and have to say that after trying my way through many different kinds, these are The BE
ST.<br /><br />The taste great, are REALLY crunchy - a key requirement for me - and have just the right
amount of salt. The Newman's Rounds are just as good - maybe even better.<br /><br />And as an added bo
nus, Paul Newmann donates all his after tax profits from the sale of his products to charity - an unbea
table combination in my book!
==================================================
Last fall I bought a slew of different protein bar brands to see which I liked the most.  Of the many b
rands I tried, most all were roughly the same in taste, texture, etc.  But these Honey Stingers stood o
ut to me.  They have a much smoother texture, and have a great sweetness to them that really makes them
taste great.  I'm a huge fan... gonna by a bunch more boxes for snacks and workouts.
==================================================
```

```
--------------------------------------------------
```

In [15]:

```python
# 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)
```

```
I bought a few of these after my apartment was infested with fruit flies. After only a few hours, the t
rap had &quot;attracted&quot; many flies and within a few days they were practically gone. This may not
be a long term  solution, but if flies are driving you crazy, consider buying this. One  caution- the s
urface is very sticky, so try to avoid touching it.
```

In [16]:

```python
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-an-elem
ent
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)
```

```
I bought a few of these after my apartment was infested with fruit flies. After only a few hours, the t
rap had "attracted" many flies and within a few days they were practically gone. This may not be a long
term  solution, but if flies are driving you crazy, consider buying this. One  caution- the surface is
very sticky, so try to avoid touching it.
==================================================
I have made these brownies for family and for a den of cub scouts and no one would have known they were
gluten free and everyone asked for seconds!  These brownies have a fudgy texture and have bits of choco
late chips in them which are delicious.  I would say the mix is very thick and a little difficult to wo
rk with.  The cooked brownies are slightly difficult to cut into very neat edges as the edges tend to c
rumble a little and I would also say that they make a slightly thinner layer of brownies than most of t
he store brand gluten containing but they taste just as good, if not better.  Highly recommended!(For t
hose wondering, this mix requires 2 eggs OR 4 egg whites and 7 tbs melted butter to prepare.  They do h
ave suggestions for lactose free and low fat preparations)
==================================================
I love Pretzels and have to say that after trying my way through many different kinds, these are The BE
ST.The taste great, are REALLY crunchy - a key requirement for me - and have just the right amount of s
alt. The Newman's Rounds are just as good - maybe even better.And as an added bonus, Paul Newmann donat
es all his after tax profits from the sale of his products to charity - an unbeatable combination in my
book!
==================================================
Last fall I bought a slew of different protein bar brands to see which I liked the most.  Of the many b
rands I tried, most all were roughly the same in taste, texture, etc.  But these Honey Stingers stood o
ut to me.  They have a much smoother texture, and have a great sweetness to them that really makes them
taste great.  I'm a huge fan... gonna by a bunch more boxes for snacks and workouts.
```

In [17]:

```python
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
```

```
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [18]:

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

```
I love Pretzels and have to say that after trying my way through many different kinds, these are The BE
ST.<br /><br />The taste great, are REALLY crunchy - a key requirement for me - and have just the right
amount of salt. The Newman is Rounds are just as good - maybe even better.<br /><br />And as an added b
onus, Paul Newmann donates all his after tax profits from the sale of his products to charity - an unbe
atable combination in my book!
==================================================
```

In [19]:

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

```
I bought a few of these after my apartment was infested with fruit flies. After only a few hours, the t
rap had &quot;attracted&quot; many flies and within a few days they were practically gone. This may not
be a long term  solution, but if flies are driving you crazy, consider buying this. One  caution- the s
urface is very sticky, so try to avoid touching it.
```

In [20]:

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

```
I love Pretzels and have to say that after trying my way through many different kinds these are The BES
T br br The taste great are REALLY crunchy a key requirement for me and have just the right amount of s
alt The Newman is Rounds are just as good maybe even better br br And as an added bonus Paul Newmann do
nates all his after tax profits from the sale of his products to charity an unbeatable combination in m
y book
```

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 revmoved in the 1st step

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 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())
final['CleanedText'] = preprocessed_reviews
```

```
100%|██████████| 87773/87773 [00:40<00:00, 2175.74it/s]
```

In [23]:

```
preprocessed_reviews[1500]
```

Out[23]:

```
'love pretzels say trying way many different kinds best taste great really crunchy key requirement righ
t amount salt newman rounds good maybe even better added bonus paul newmann donates tax profits sale pr
oducts charity unbeatable combination book'
```

[3.2] Preprocessing Review Summary

In [90]:

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

[4] Featurization

[4.1] BAG OF WORDS

In [26]:

```
#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 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', 'aaa', 'aaaa', 'aaaaa', 'aaaaaaaaaaaa', 'aaaaaaaaaaaaaaa', 'aaaaaaahhhhhh', 'aaaaaaarrrrrggghhh', 'aaaaaawwwwwwwwww', 'aaaaah']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (87773, 54904)
the number of unique words  54904
```

[4.2] Bi-Grams and n-Grams.

In [27]:

```
#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  (87773, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[4.3] TF-IDF

In [0]:

```
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 [153]:

```
# Word2Vec model for train/test and cv dataset
i=0
list_of_sent=[]
for sent in X_train['CleanedText'].values:
    list_of_sent.append(sent.split())
print(X_tr['CleanedText'].values[0])

print("*****************************************************************")
print(list_of_sent[0])
```

```
# Word2Vec model for test
i=0
list_of_sent_test=[]
for sent in X_test['CleanedText'].values:
    list_of_sent_test.append(sent.split())
print(X_test['CleanedText'].values[0])
print("*****************************************************************")
print(list_of_sent_test[0])

# Word2Vec model for CV
i=0
list_of_sent_cv=[]
for sent in X_cv['CleanedText'].values:
    list_of_sent_cv.append(sent.split())
print(X_test['CleanedText'].values[0])
print("*****************************************************************")
print(list_of_sent_cv[0])

w2v_model_train=Word2Vec(list_of_sent,min_count=5,size=50, workers=5)
w2v_model_test=Word2Vec(list_of_sent_test,min_count=5,size=50, workers=5)
w2v_words = list(w2v_model_train.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
different opinion first reviewer think triple berry granola simply great tasty full whole grain goodnes
s may need berries though sometimes get bag no berries try see like way full fiber good
*****************************************************************
['excited', 'gum', 'not', 'artificial', 'sweeteners', 'bought', 'two', 'boxes', 'glee', 'gum', 'pepperm
int', 'triple', 'berry', 'arrived', 'tore', 'open', 'box', 'popped', 'pieces', 'disappointment', 'began
', 'minutes', 'later', 'flavor', 'quickly', 'dissipated', 'took', 'couple', 'pieces', 'flavor', 'return
ed', 'disappointed', 'later', 'went', 'away', 'money', 'better', 'spent', 'sucking', 'lollipops', 'eati
ng', 'found', 'gum', 'not', 'artificial', 'sweeteners', 'dentine', 'gum', 'cinnamon', 'never', 'ending'
, 'disappointment', 'never', 'ending', 'calories', 'try', 'keep', 'flavor']
disappointed gerber added dha green beans son used love gerber organic green beans unwittingly tried fe
eding new formulation added dha kept refusing notice consistency different usual gel like texture read
ingredients saw added tuna oil today learned another favorites gerber organic carrots dha added frustra
ted looks like making baby food
*****************************************************************
['disappointed', 'gerber', 'added', 'dha', 'green', 'beans', 'son', 'used', 'love', 'gerber', 'organic'
, 'green', 'beans', 'unwittingly', 'tried', 'feeding', 'new', 'formulation', 'added', 'dha', 'kept', 'r
efusing', 'notice', 'consistency', 'different', 'usual', 'gel', 'like', 'texture', 'read', 'ingredients
', 'saw', 'added', 'tuna', 'oil', 'today', 'learned', 'another', 'favorites', 'gerber', 'organic', 'car
rots', 'dha', 'added', 'frustrated', 'looks', 'like', 'making', 'baby', 'food']
disappointed gerber added dha green beans son used love gerber organic green beans unwittingly tried fe
eding new formulation added dha kept refusing notice consistency different usual gel like texture read
ingredients saw added tuna oil today learned another favorites gerber organic carrots dha added frustra
ted looks like making baby food
*****************************************************************
['become', 'favorite', 'stash', 'tea', 'liked', 'earl', 'grey', 'double', 'bergamot', 'earl', 'grey', '
twice', 'good', 'whether', 'not', 'tastes', 'like', 'areal', 'earl', 'grey', 'say', 'little', 'differen
t', 'like', 'earl', 'grey', 'definitely', 'worth', 'try']
number of words that occured minimum 5 times  15733
sample words  ['excited', 'gum', 'not', 'artificial', 'sweeteners', 'bought', 'two', 'boxes', 'glee', '
peppermint', 'triple', 'berry', 'arrived', 'tore', 'open', 'box', 'popped', 'pieces', 'disappointment',
'began', 'minutes', 'later', 'flavor', 'quickly', 'dissipated', 'took', 'couple', 'returned', 'disappoi
nted', 'went', 'away', 'money', 'better', 'spent', 'sucking', 'lollipops', 'eating', 'found', 'cinnamon
', 'never', 'ending', 'calories', 'try', 'keep', 'great', 'product', 'definitely', 'buy', 'looking', 'c
ontained']
```

In [125]:

```
# 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=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 ")
```

```
[('fantastic', 0.8497572541236877), ('awesome', 0.8415793180465698), ('good', 0.8220787644386292), ('te
rrific', 0.8089662194252014), ('excellent', 0.793988823890686), ('wonderful', 0.7904127836227417), ('pe
rfect', 0.7855398654937744), ('amazing', 0.7499324083328247), ('nice', 0.712704598903656), ('fabulous',
0.6920047402381897)]
==================================================
[('greatest', 0.8216128945350647), ('best', 0.7184992432594299), ('nastiest', 0.7126660346984863), ('ta
stiest', 0.7060397267341614), ('coolest', 0.6580905914306641), ('closest', 0.6341551542282104), ('disgu
sting', 0.618719756603241), ('experienced', 0.6100812554359436), ('horrible', 0.598488450050354), ('cry
', 0.5878204107284546)]
```

In [0]:

```
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', '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 Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

In [0]:

```
# 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 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 [00:03<00:
```

```
100%|██████████| 4986/4986 [00:03<00:00, 1330.47it/s]
```

```
4986
50
```

In [129]:

```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_sent): # 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_train.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]))



sent_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sent_test): # 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_train.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_test.append(sent_vec)
print(len(sent_vectors_test))
print(len(sent_vectors_test[0]))
```

```
100%|██████████| 70218/70218 [02:29<00:00, 469.41it/s]
```

```
70218
50
```

```
100%|██████████| 17555/17555 [00:38<00:00, 461.20it/s]
```

```
17555
50
```

In [154]:

```python
sent_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sent_cv): # 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_train.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_cv.append(sent_vec)
print(len(sent_vectors_cv))
print(len(sent_vectors_cv[0]))
```

```
100%|██████████| 14044/14044 [00:33<00:00, 420.09it/s]
```

```
14044
50
```

[4.4.1.2] TFIDF weighted W2v

In [130]:

```python
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X_train['CleanedText'].values)
# 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 [131]:

```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_sent): # 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_train.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


# 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_test = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sent_test): # 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_train.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_test.append(sent_vec)
    row += 1
```

```
100%|██████████| 70218/70218 [52:31<00:00, 22.28it/s]
100%|██████████| 17555/17555 [13:54<00:00, 21.03it/s]
```

In [155]:

```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_cv = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sent_cv): # 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_train.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_cv.append(sent_vec)
    row += 1
```

```
100%|██████████| 14044/14044 [10:15<00:00, 22.83it/s]
```

[5] Assignment 5: Apply Logistic Regression

1. **Apply Logistic Regression on these feature sets**
    - SET 1:Review text, preprocessed one converted into vectors using (BOW)
    - SET 2:Review text, preprocessed one converted into vectors using (TFIDF)
    - SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)
    - SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)

2. **Hyper paramter tuning (find best hyper parameters corresponding the algorithm that you choose)**
    - Find the best hyper parameter which will give the maximum AUC value
    - Find the best hyper paramter using k-fold cross validation or simple cross validation data
    - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

3. **Pertubation Test**
    - Get the weights W after fit your model with the data X i.e Train data.
    - Add a noise to the X (X' = X + e) and get the new data set X' (if X is a sparse matrix, X.data+=e)
    - Fit the model again on data X' and get the weights W'
    - Add a small eps value(to eliminate the divisible by zero error) to W and W’ i.e W=W+10^-6 and W’ = W’+10^-6
    - Now find the % change between W and W' (| (W-W') / (W) |)*100)
    - Calculate the 0th, 10th, 20th, 30th, ...100th percentiles, and observe any sudden rise in the values of percentage_change_vector
    - Ex: consider your 99th percentile is 1.3 and your 100th percentiles are 34.6, there is sudden rise from 1.3 to 34.6, now calculate the 99.1, 99.2, 99.3,..., 100th percentile values and get the proper value after which there is sudden rise the values, assume it is 2.5
    - Print the feature names whose % change is more than a threshold x(in our example it's 2.5)

4. **Sparsity**
    - Calculate sparsity on weight vector obtained after using L1 regularization

    NOTE: Do sparsity and multicollinearity for any one of the vectorizers. Bow or tf-idf is recommended.

5. **Feature importance**
    - Get top 10 important features for both positive and negative classes separately.

6. **Feature engineering**

- To increase the performance of your model, you can also experiment with with feature engineering like :
  - Taking length of reviews as another feature.
  - Considering some features from review summary as well.

7. **Representation of results**

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.
- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
- Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

8. **Conclusion**

- You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link.

Applying Logistic Regression

Splitting into train and test dataset

In [24]:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn import model_selection

#X_train, X_test= np.split(final, [int(0.80 *len(final))])
X_train, X_test, y_train, y_test = train_test_split(final, y, test_size=0.2, shuffle=False)
#y_train = X_train['Score']
#y_test = X_test['Score']
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

#Splitting train data into train and cv(60:20)
X_tr, X_cv, y_tr, y_cv = train_test_split(X_train, y_train, test_size=0.2, shuffle=False)
print(X_tr.shape, y_tr.shape)
print(X_cv.shape, y_cv.shape)
```

```
(70218, 11) (70218,)
(17555, 11) (17555,)
(56174, 11) (56174,)
(14044, 11) (14044,)
```

In [27]:

```
#Applying BoW
count_vect = CountVectorizer()
count_vect.fit(X_train['CleanedText'])
train_bow = count_vect.transform(X_tr['CleanedText'])
cv_bow = count_vect.transform(X_cv['CleanedText'])
```

```
test_bow = count_vect.transform(X_test['CleanedText'])
print(test_bow.shape)
print(cv_bow.shape)
print(train_bow.shape)
```

```
(17555, 49066)
(14044, 49066)
(56174, 49066)
```

[5.1] Logistic Regression on BOW, SET 1

[5.1.1] Applying Logistic Regression with L1 regularization on BOW, SET 1

In [28]:

```
#https://machinelearningmastery.com/how-to-tune-algorithm-parameters-with-scikit-learn/
#Applying GridSearch to find the best hyperparameter C

tuned_parameter = [{'C': [0.00001, 0.0001, 0.001, 0.01, 1, 10, 100, 1000, 10000]}]
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(class_weight='balanced')
scoring = {'AUC': 'roc_auc', 'Accuracy': 'accuracy'}
grid = GridSearchCV(estimator=clf, param_grid = tuned_parameter ,scoring = scoring, refit = 'AUC')
grid.fit(train_bow, y_tr)
print(grid)
# summarize the results of the grid search
print(grid.best_score_)
print(grid.best_estimator_)
print(grid.score(cv_bow,y_cv))
results = grid.cv_results_
#print(results)
#print(grid.confusion_matrix)
```

```
C:\Users\Dell\AppData\Roaming\Python\Python36\site-packages\sklearn\svm\base.py:929: ConvergenceWarning
: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\Dell\AppData\Roaming\Python\Python36\site-packages\sklearn\svm\base.py:929: ConvergenceWarning
: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\Dell\AppData\Roaming\Python\Python36\site-packages\sklearn\svm\base.py:929: ConvergenceWarning
: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\Dell\AppData\Roaming\Python\Python36\site-packages\sklearn\svm\base.py:929: ConvergenceWarning
: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\Dell\AppData\Roaming\Python\Python36\site-packages\sklearn\svm\base.py:929: ConvergenceWarning
: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\Dell\AppData\Roaming\Python\Python36\site-packages\sklearn\svm\base.py:929: ConvergenceWarning
: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\Dell\AppData\Roaming\Python\Python36\site-packages\sklearn\svm\base.py:929: ConvergenceWarning
: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
             estimator=LogisticRegression(C=1.0, class_weight='balanced',
                                          dual=False, fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='warn',
                                          n_jobs=None, penalty='l2',
                                          random_state=None, solver='warn',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'C': [1e-05, 0.0001, 0.001, 0.01, 1, 10, 100, 1000,
                                10000]}],
             pre_dispatch='2*n_jobs', refit='AUC', return_train_score=False,
             scoring={'AUC': 'roc_auc', 'Accuracy': 'accuracy'}, verbose=0)
0.9265321931582181
```

```
LogisticRegression(C=0.01, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
0.9362469080203264
```

[5.1.1.1] Calculating sparsity on weight vector obtained using L1 regularization on BOW, SET 1

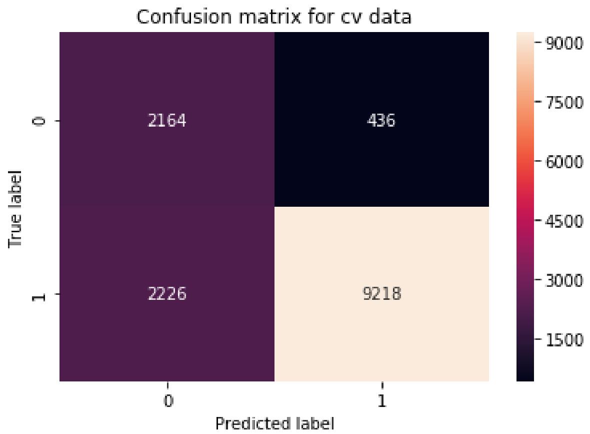
In [32]:

```
# Please write all the code with proper documentation

clf = LogisticRegression(C=0.01, penalty ='l1',class_weight='balanced');
cclf=clf.fit(train_bow, y_tr).predict(cv_bow)
w = clf.coef_
print("Sparsity:",np.count_nonzero(w))

#cclf=clf.fit(train_bow, y_tr).predict(cv_bow)
pred_cv = clf.predict_proba(cv_bow)[:,1]
fpr, tpr, thresholds = roc_curve(y_cv,pred_cv)
roc_auc_cv = auc(fpr, tpr)
print('Area under the ROC curve : %f', + roc_auc_cv)
#Plotting confusion matrix
import seaborn as sns
conf_mat = confusion_matrix(y_cv, cclf)
print(conf_mat)
#conf_normalized = conf_mat.astype('int') / conf_mat.sum(axis=1)[:, np.newaxis]
sns.heatmap(conf_mat, annot=True, fmt ='g')
plt.title('Confusion matrix for cv data')
plt.ylabel('True label')
plt.xlabel('Predicted label')

#clf = LogisticRegression(C=0.1, penalty ='l1');
#clf.fit(train_bow, y_tr)
#w = clf.coef_
#print(np.count_nonzero(w))
```

```
Sparsity: 177
Area under the ROC curve : %f 0.8975796016723576
[[2164  436]
 [2226 9218]]
```

Out[32]:

```
Text(0.5, 15.0, 'Predicted label')
```



In [33]:

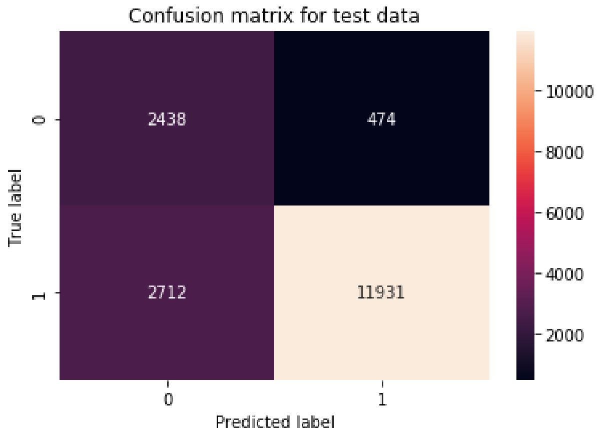
```
#predicting test data
cclf=clf.fit(train_bow, y_tr).predict(test_bow)
w = clf.coef_
print("Sparsity:",np.count_nonzero(w))

#cclf=clf.fit(train_bow, y_tr).predict(cv_bow)
pred_test = clf.predict_proba(test_bow)[:,1]
```

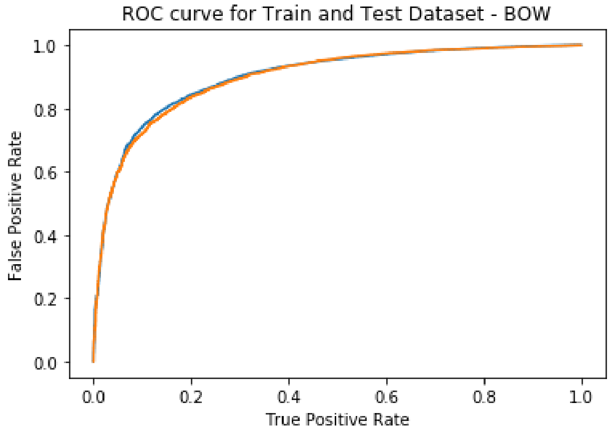
```
fpr, tpr, thresholds = roc_curve(y_test,pred_test)
roc_auc_test = auc(fpr, tpr)
print('Area under the ROC curve : %f', + roc_auc_test)
#Plotting confusion matrix
import seaborn as sns
conf_mat = confusion_matrix(y_test, cclf)
print(conf_mat)
#conf_normalized = conf_mat.astype('int') / conf_mat.sum(axis=1)[:, np.newaxis]
sns.heatmap(conf_mat, annot=True, fmt ='g')
plt.title('Confusion matrix for test data')
plt.ylabel('True label')
plt.xlabel('Predicted label')


#Plot ROC Curve
plt.figure(0).clf()
fpr, tpr, thresholds = roc_curve(y_test,pred_test)
roc_auc_test = auc(fpr, tpr)
plt.plot(fpr,tpr,label="Test Data, auc="+str(roc_auc_test))
fpr, tpr, thresh = roc_curve(y_cv, pred_cv)
roc_auc_cv = auc(fpr, tpr)
plt.plot(fpr,tpr,label="Train Data, auc="+str(roc_auc_cv))
plt.title('ROC curve for Train and Test Dataset - BOW')
plt.xlabel('True Positive Rate')
plt.ylabel('False Positive Rate')
```

```
Sparsity: 177
Area under the ROC curve : %f 0.9004300872674412
[[ 2438   474]
 [ 2712 11931]]
```

Out[33]:

```
Text(0, 0.5, 'False Positive Rate')
```





[5.1.2] Applying Logistic Regression with L2 regularization on BOW, SET 1

In [34]:

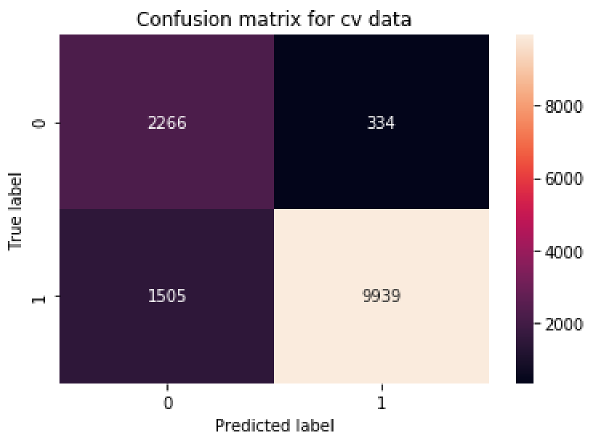
```
# Please write all the code with proper documentation
```

```
clf = LogisticRegression(C=0.01, penalty ='l2',class_weight='balanced');
cclf=clf.fit(train_bow, y_tr).predict(cv_bow)
W = clf.coef_
print("Sparsity:",np.count_nonzero(w))

#cclf=clf.fit(train_bow, y_tr).predict(cv_bow)
pred_cv = clf.predict_proba(cv_bow)[:,1]
fpr, tpr, thresholds = roc_curve(y_cv,pred_cv)
roc_auc_cv = auc(fpr, tpr)
print('Area under the ROC curve : %f', + roc_auc_cv)
#Plotting confusion matrix
import seaborn as sns
conf_mat = confusion_matrix(y_cv, cclf)
print(conf_mat)
#conf_normalized = conf_mat.astype('int') / conf_mat.sum(axis=1)[:, np.newaxis]
sns.heatmap(conf_mat, annot=True, fmt ='g')
plt.title('Confusion matrix for cv data')
plt.ylabel('True label')
plt.xlabel('Predicted label')

#clf = LogisticRegression(C=0.1, penalty ='l1');
#clf.fit(train_bow, y_tr)
#w = clf.coef_
#print(np.count_nonzero(w))
```

```
Sparsity: 177
Area under the ROC curve : %f 0.9362469080203264
[[2266  334]
 [1505 9939]]
```

Out[34]:

```
Text(0.5, 15.0, 'Predicted label')
```



In [35]:

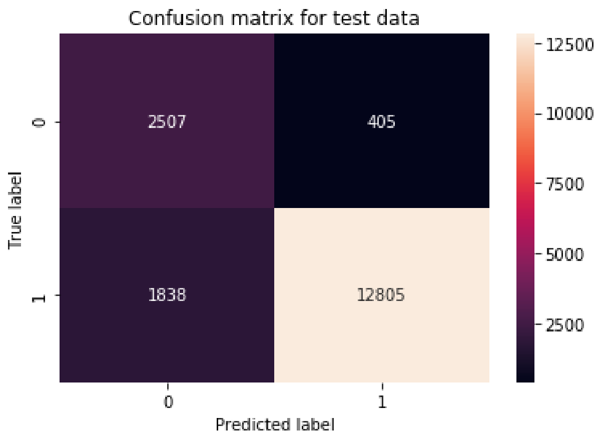
```
#predicting test data
cclf=clf.fit(train_bow, y_tr).predict(test_bow)
W = clf.coef_
print("Sparsity:",np.count_nonzero(w))

#cclf=clf.fit(train_bow, y_tr).predict(cv_bow)
pred_test = clf.predict_proba(test_bow)[:,1]
fpr, tpr, thresholds = roc_curve(y_test,pred_test)
roc_auc_test = auc(fpr, tpr)
print('Area under the ROC curve : %f', + roc_auc_test)
#Plotting confusion matrix
import seaborn as sns
conf_mat = confusion_matrix(y_test, cclf)
print(conf_mat)
#conf_normalized = conf_mat.astype('int') / conf_mat.sum(axis=1)[:, np.newaxis]
sns.heatmap(conf_mat, annot=True, fmt ='g')
plt.title('Confusion matrix for test data')
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

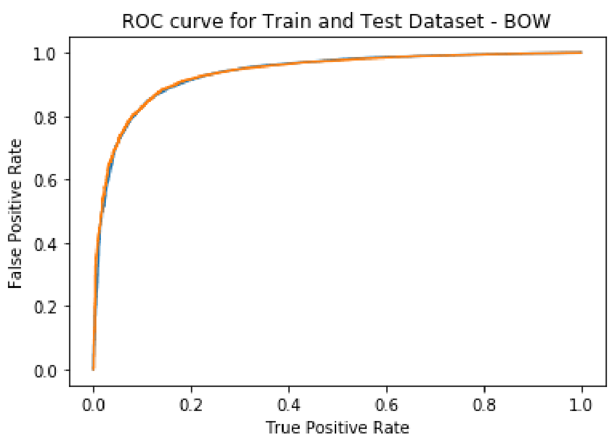
```
#Plot ROC Curve
plt.figure(0).clf()
fpr, tpr, thresholds = roc_curve(y_test,pred_test)
roc_auc_test = auc(fpr, tpr)
plt.plot(fpr,tpr,label="Test Data, auc="+str(roc_auc_test))
fpr, tpr, thresh = roc_curve(y_cv, pred_cv)
roc_auc_cv = auc(fpr, tpr)
plt.plot(fpr,tpr,label="Train Data, auc="+str(roc_auc_cv))
plt.title('ROC curve for Train and Test Dataset - BOW')
plt.xlabel('True Positive Rate')
plt.ylabel('False Positive Rate')
```

```
Sparsity: 177
Area under the ROC curve : %f 0.9347828360773968
[[ 2507   405]
 [ 1838 12805]]
```

Out[35]:

```
Text(0, 0.5, 'False Positive Rate')
```





[5.1.2.1] Performing pertubation test (multicollinearity check) on BOW, SET 1

In [36]:

```
def collinear_features_fun(vectorizer, w):
    feature_names = vectorizer.get_feature_names()
    topn_class = sorted(zip(w, feature_names), reverse=True)[:]
    features_list = []
    for coef, feat in topn_class:
        if coef != 0.0 :
            features_list.append(feat)
    collinear_features = features_list;
    return collinear_features;
```

In [73]:

```
Clf = LogisticRegression(C=1, penalty='l1',class_weight='balanced')
```

```
Clf.fit(train_bow, y_tr)
X = Clf.coef_
print(X)
print("Sparsity with actual data",np.count_nonzero(X))


train_bow_noise=train_bow
train_bow_noise.data+=np.random.normal(loc=0,scale=0.0001,size=train_bow_noise.data.shape)
#noise = np.random.normal(0,1,train_bow.shape)
#train_bow_noise=train_bow+noise

print(train_bow_noise.shape)

clf_Noise=LogisticRegression(penalty='l1',C=1,class_weight='balanced')
clf_Noise.fit(train_bow_noise,y_tr)
X_noise=clf_Noise.coef_
print(X_noise)
print("Sparsity with noise data",np.count_nonzero(X_noise))
```

```
[[-0.11917375  0.          0.         ...  0.          0.
   0.        ]]
Sparsity with actual data 5306
(56174, 49066)
[[-0.11935698  0.          0.         ...  0.          0.
   0.        ]]
Sparsity with noise data 5301
```

In [74]:

```
X=X[0]+10**-6
X_noise=X_noise[0]+10**-6
print(X)
print(X_noise)
#w = list(X)
print(len(X))
print(len(X_noise))
```

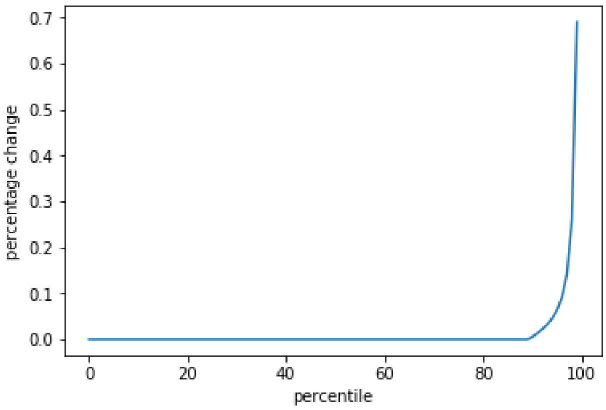
```
[-1.19172747e-01  1.00000000e-06  1.00000000e-06 ...  1.00000000e-06
  1.00000000e-06  1.00000000e-06]
[-1.19355984e-01  1.00000000e-06  1.00000000e-06 ...  1.00000000e-06
  1.00000000e-06  1.00000000e-06]
49066
49066
```

In [75]:

```
#to eliminate divisible by zero error we will add 10^-6 to W_before and W_after
change_vector_percentage = []
for i in tqdm(range(0,len(X))):
    change_vector = 0
    change_vector=(abs((X[i]-(X_noise[i]))/(X[i])))*100
    change_vector_percentage.append(change_vector)
#per_vector=[]
#print(change_vector_percentage)
percentile_value =[]
percentile = []
i=0
while i<100:
    percentile.append(i)
    percentile_value.append(np.percentile(change_vector_percentage,i))
    i = i+1;

plt.plot(percentile, percentile_value)
#print(percentile_value)
plt.xlabel('percentile')
plt.ylabel('percentage change')
plt.show()
#for i in range(len(W[0])):
    #val=W_noise[0][i]-W[0][i]
    #val/=W[0][i]
    #per_vector.append(val)

#original_per_vect=np.absolute(per_vector)
#per_vector=sorted(np.absolute(per_vector))[::-1]
```

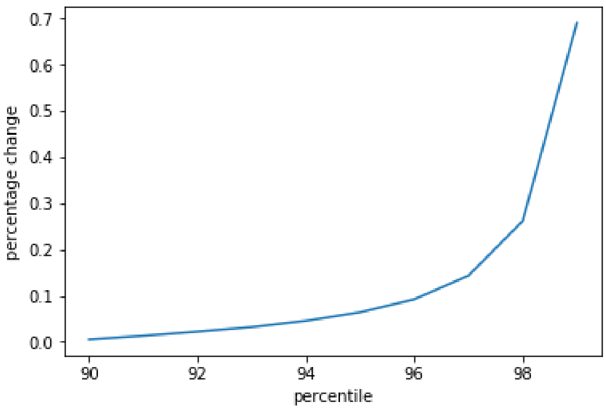
```
#percentage change in vectors

#per_vector[ :10]
```

```
100%|██████████| 49066/49066 [00:00<00:00, 262104.27it/s]
```



In [80]:

```python
ninty_percentile_value =[]
ninty_percentile = []
j=90
while j<100:
    ninty_percentile.append(j)
    ninty_percentile_value.append(np.percentile(change_vector_percentage,j))
    j = j+1;

plt.plot(ninty_percentile, ninty_percentile_value)
#print(percentile_value)
plt.xlabel('percentile')
plt.ylabel('percentage change')
plt.show()

print(ninty_percentile)
print(ninty_percentile_value)
```

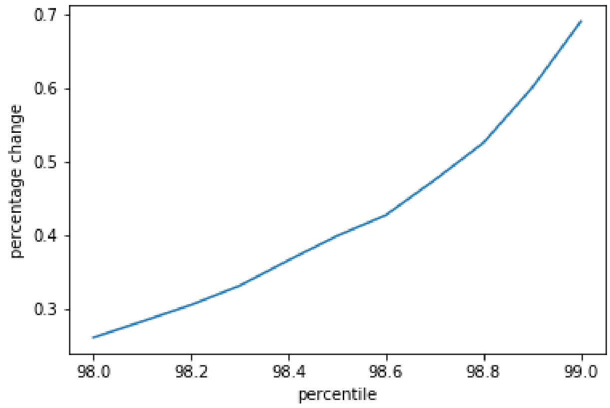



```
[90, 91, 92, 93, 94, 95, 96, 97, 98, 99]
[0.005355582997842501, 0.013357155937351987, 0.022396385534470965, 0.03221516776804075, 0.0457184095285
1004, 0.06424666922694774, 0.09220428964926967, 0.14341919634109088, 0.26138061176601085, 0.68996922587
21515]
```

In [84]:

```python
nint_percentile_value =[]
nint_percentile = []
k=98
while k<99:
    nint_percentile.append(k)
    nint_percentile_value.append(np.percentile(change_vector_percentage,k))
    k = k+0.1;
```

```
plt.plot(nint_percentile, nint_percentile_value)
#print(percentile_value)
plt.xlabel('percentile')
plt.ylabel('percentage change')
plt.show()

print(nint_percentile)
print(nint_percentile_value)
```




```
[98, 98.1, 98.19999999999999, 98.29999999999998, 98.39999999999998, 98.49999999999997, 98.5999999999999
7, 98.69999999999996, 98.79999999999995, 98.89999999999995, 98.99999999999994]
[0.26138061176601085, 0.28326339409667756, 0.3056699743110664, 0.33172067259023524, 0.36623717343449136
, 0.3992151998721237, 0.42745232763129903, 0.47503207078281867, 0.5253464597665197, 0.600058267403584,
0.6899692258719867]
```

In [85]:

```
#At percentile 98.5, the value is 0.427 which is the elbow point. The features/words whose percentage c
hange is greater than
#this threshold value 0.427 are affected by noise and referred as multicollinear features
w_threshold=[]
count = 0;
for i in range(0,len(change_vector_percentage)):
    if change_vector_percentage[i] > 0.427:
        count = count+1;
        w_threshold.append(X[i])
    else:
        w_threshold.append(0.0)
print(count)
```

```
689
```

In [86]:

```
#multicolinear features
features_threshold = collinear_features_fun(count_vect,w_threshold)
print(features_threshold)
```

```
['volvic', 'usb', 'computer', 'lapsang', 'kudos', 'bumblebee', 'activities', 'gently', 'upc', 'wherever
', 'fuzzy', 'space', 'conventional', 'petite', 'chelated', 'gripe', 'picaridin', 'paupa', 'seldom', 'cu
isine', 'taint', 'boil', 'recognized', 'grapeseed', 'earthborn', 'smack', 'connection', 'record', 'deep
er', 'electrolytes', 'counter', 'acana', 'heavenly', 'heated', 'bro', 'freshens', 'receives', 'channels
', 'proteins', 'organix', 'happybaby', 'leakage', 'zoe', 'distribution', 'wt', 'thoroughly', 'weary', '
stands', 'cares', 'account', 'experimenting', 'unbelievably', 'impressive', 'refrigerate', 'follow', 's
pain', 'indomie', 'excess', 'refrigerated', 'hopper', 'gulp', 'power', 'moisture', 'aware', 'lasagna',
'reusable', 'workable', 'focused', 'jose', 'traps', 'brilliant', 'solution', 'holistic', 'rinsing', 'so
urcing', 'dogswell', 'denver', 'methods', 'luckily', 'bakery', 'posted', 'discovering', 'reorder', 'eye
brow', 'differences', 'shots', 'highland', 'place', 'flip', 'creamier', 'souchong', 'shared', 'completl
y', 'discontinue', 'generation', 'thus', 'purebites', 'dreaming', 'girardelli', 'die', 'toast', 'profil
e', 'adequate', 'additives', 'twizzlers', 'handfuls', 'marco', 'stop', 'simmered', 'guarana', 'puked',
'wall', 'el', 'grooves', 'detox', 'hey', 'environment', 'alcohols', 'beyond', 'gobbled', 'kefir', 'hmmm
', 'system', 'demerara', 'las', 'fashioned', 'bird', 'antioxidant', 'banana', 'everlasting', 'consumer'
, 'absent', 'fenugreek', 'keebler', 'directly', 'cornstarch', 'gym', 'roasting', 'begged', 'scones', 's
hredded', 'class', 'tast', 'fewer', 'brandy', 'came', 'ceramic', 'einstein', 'rave', 'dick', 'newmans',
'shampoo', 'distiller', 'alert', 'geisha', 'excellence', 'god', 'filet', 'manufactured', 'quest', 'labs
', 'milk', 'square', 'leash', 'bahlsen', 'scrambled', 'fromm', 'inflammatory', 'photos', 'pb', 'fillers
', 'beach', 'talked', 'nation', 'know', 'phase', 'balsamic', 'dissolves', 'anyways', 'finer', 'western'
```

```
', 'beach', 'talked', 'nation', 'know', 'phase', 'balsamic', 'dissolves', 'anyways', 'liner', 'western'
, 'peels', 'cola', 'porch', 'oil', 'capsules', 'tubes', 'strongly', 'diet', 'lowfat', 'toppings', 'samp
led', 'overnight', 'retail', 'meeting', 'recommends', 'require', 'advantages', 'comes', 'pesto', 'atten
tion', 'cooking', 'pleasing', 'swedish', 'perishable', 'grande', 'successful', 'exceptionally', 'size',
'flow', 'papaya', 'pretzels', 'stairs', 'jolly', 'marshmellow', 'category', 'bottled', 'stovetop', 'ben
gal', 'prevent', 'dents', 'goodie', 'nerds', 'miniature', 'va', 'roommate', 'cook', 'akita', 'powdering
', 'shavings', 'personally', 'gatorade', 'providing', 'bothered', 'compare', 'green', 'chais', 'types',
'colors', 'entirely', 'tricks', 'believe', 'swiftly', 'poops', 'mine', 'traces', 'occasional', 'weekend
', 'anymore', 'plants', 'handles', 'watched', 'reminds', 'heal', 'ms', 'canned', 'summer', 'tree', 'com
plain', 'roland', 'boiled', 'subscribing', 'dirty', 'cooling', 'ride', 'sodas', 'sugarworks', 'fluffy',
'reaches', 'seal', 'steal', 'toxic', 'biodegradable', 'giannini', 'yuccky', 'worsethey', 'winebaskets',
'spicybut', 'shutting', 'reeks', 'prairie', 'ovenand', 'maxes', 'harrogate', 'guys', 'degermed', 'compa
rtment', 'one', 'rare', 'number', 'finger', 'debbiednorvell', 'ot', 'consumption', 'tablets', 'batches'
, 'send', 'chery', 'clorox', 'gould', 'first', 'brewed', 'alike', 'mouth', 'damaged', 'owners', 'autho
r', 'public', 'deflated', 'gels', 'avoderm', 'break', 'unnamed', 'guanabana', 'appears', 'arrowhead', '
outside', 'byzantine', 'coffe', 'misrepresentation', 'zico', 'revealed', 'butt', 'almond', 'accidentall
y', 'present', 'lavazza', 'hours', 'deet', 'sickly', 'overpower', 'shrunk', 'timothy', 'substantial', '
watch', 'market', 'honestly', 'cherimoyas', 'inside', 'wintergreens', 'filming', 'online', 'ringer', 'j
oe', 'see', 'rocky', 'folgers', 'peel', 'outta', 'soymilk', 'refunded', 'refreshment', 'obligate', 'ice
breakers', 'teh', 'gevalia', 'stinking', 'cats', 'age', 'grind', 'hmm', 'larvae', 'distance', 'evidence
d', 'thekeurig', 'manufacturer', 'slopped', 'soups', 'saddest', 'reviewers', 'jif', 'different', 'news'
, 'investigation', 'zi', 'renton', 'kinds', 'vitamin', 'costco', 'gizmo', 'amarillos', 'content', 'effe
ct', 'thing', 'gluten', 'shoe', 'ship', 'strips', 'chug', 'false', 'seasoning', 'strip', 'real', 'cruch
y', 'experiencing', 'hundreds', 'caloriesdeceptive', 'pastabut', 'urge', 'philippines', 'selected', 'di
rectionsfrom', 'sacrifices', 'impression', 'guava', 'leaving', 'murchie', 'late', 'brendan', 'pastas',
'daughters', 'article', 'wide', 'hulless', 'altoids', 'research', 'bulky', 'oro', 'driveway', 'noted',
'whip', 'learn', 'saucepan', 'burn', 'sold', 'camouflage', 'calamari', 'contributing', 'herbal', 'milde
wy', 'candle', 'told', 'adult', 'accept', 'tray', 'learning', 'selections', 'thru', 'akin', 'ness', 'wh
is', 'caloric', 'craxker', 'msds', 'vegit', 'introduce', 'insist', 'creepy', 'pecan', 'feather', 'waste
ful', 'cento', 'kohl', 'wrapping', 'honeydew', 'freeze', 'eb', 'beagle', 'believes', 'harney', 'veinier
', 'fromthe', 'choc', 'solofil', 'point', 'meyers', 'straws', 'casseroles', 'managed', 'animal', 'fenne
l', 'bodied', 'mass', 'inevitable', 'hardened', 'coworker', 'website', 'frito', 'continues', 'sweetleaf
', 'couch', 'grey', 'flush', 'otehres', 'petowners', 'escape', 'crap', 'compromised', 'mesquite', 'bunn
y', 'fed', 'hersey', 'paw', 'paralyzed', 'finished', 'consumers', 'unacceptable', 'monotone', 'higher',
'bombshells', 'lea', 'jan', 'newville', 'gree', 'survive', 'maine', 'jasmine', 'vernors', 'st', 'unwitt
ingly', 'beginning', 'redwoods', 'constant', 'approx', 'watermark', 'tping', 'lolthese', 'jumps', 'comm
enter', 'kernelly', 'phillip', 'destroys', 'calcified', 'panic', 'itme', 'cfl', 'displayed', 'wiped', '
lay', 'attempted', 'gnaw', 'poisoning', 'perched', 'guar', 'bergamont', 'havi', 'caring', 'deader', 'un
aware', 'ahmad', 'parts', 'lactic', 'gummybears', 'spaghettiit', 'deleted', 'wrinkly', 'esp', 'sucker'
, 'faking', 'carolyn', 'david', 'pete', 'ing', 'ingested', 'route', 'badi', 'kah', 'researched', 'pumpi
ng', 'pucker', 'proactive', 'bronx', 'regretted', 'respect', 'lattes', 'ter', 'immortal', 'breville', '
queazy', 'fong', 'washing', 'emptor', 'noone', 'regatta', 'wle', 'abnormal', 'paper', 'bab', 'audacity'
, 'latervia', 'cylindrical', 'remotly', 'legal', 'itfor', 'aguave', 'episode', 'cite', 'walloping', 'in
conclusion', 'baffled', 'amaretto', 'inclusive', 'walkers', 'sex', 'duds', 'alive', 'hocker', 'diminish
es', 'wean', 'bitten', 'continental', 'hoppin', 'sorting', 'relive', 'ppm', 'disapointing', 'dop', 'aso
ft', 'grocwery', 'rolands', 'irritate', 'linux', 'equipment', 'showing', 'todd', 'mugged', 'referred',
'hugely', 'visable', 'drippy', 'intro', 'insightbb', 'spoonful', 'smallfrom', 'supports', 'thankx', 'mi
ntiness', 'agoraphobic', 'seek', 'morgue', 'nocks', 'dissaponited', 'johns', 'shopped', 'colour', 'yuck
iness', 'zapp', 'rio', 'earthquakes', 'taxidermist', 'wll', 'pillows', 'popovers', 'stamped', 'awfulonl
y', 'plug', 'yuckky', 'mopped', 'newer', 'torched', 'tech', 'mayorga', 'sneaky', 'windows', 'atis', 'ca
ps', 'chao', 'officemate', 'insouth', 'stabilizes', 'cos', 'aforementioned', 'perrins', 'dumpster', 'sa
lesman', 'battles', 'embarassed', 'laptop', 'chuao', 'havebut', 'anywayz', 'microphone', 'huy', 'distil
led', 'shouldnt', 'isbn', 'technicians', 'taylors', 'stolen', 'yadayadayada', 'storebrand', 'elegible',
'minimial', 'akg', 'emerging', 'hapkido', 'embodies', 'toot', 'saquin']
```

In [ ]:

```
#Trying different C values to predict Sparsity
```

In [116]:

```
clf = LogisticRegression(C=100, penalty='l1')
clf.fit(train_bow, y_train)

pred = clf.predict(test_bow)
ac1 = accuracy_score(y_test, pred) * 100
er1 = np.around(100 - ac1, decimals = 2)

w = clf.coef_
s1 = np.count_nonzero(w)
print("Sparsity with C = 100:", s1)

clf = LogisticRegression(C=10, penalty='l1')
clf.fit(train_bow, y_train)

pred = clf.predict(test_bow)
```

```
pred = clf.predict(test_bow)
ac1 = accuracy_score(y_test, pred) * 100
er1 = np.around(100 - ac1, decimals = 2)

w = clf.coef_
s1 = np.count_nonzero(w)
print("Sparsity with C = 10:", s1)

clf = LogisticRegression(C=1, penalty='l1')
clf.fit(train_bow, y_train)

pred = clf.predict(test_bow)
ac1 = accuracy_score(y_test, pred) * 100
er1 = np.around(100 - ac1, decimals = 2)

w = clf.coef_
s1 = np.count_nonzero(w)
print("Sparsity with C = 1:", s1)


clf = LogisticRegression(C=0.1, penalty='l1')
clf.fit(train_bow, y_train)

pred = clf.predict(test_bow)
ac1 = accuracy_score(y_test, pred) * 100
er1 = np.around(100 - ac1, decimals = 2)

w = clf.coef_
s1 = np.count_nonzero(w)
print("Sparsity with C = 0.1:", s1)


clf = LogisticRegression(C=0.01, penalty='l1')
clf.fit(train_bow, y_train)

pred = clf.predict(test_bow)
ac1 = accuracy_score(y_test, pred) * 100
er1 = np.around(100 - ac1, decimals = 2)

w = clf.coef_
s1 = np.count_nonzero(w)
print("Sparsity with C = 0.01:", s1)
```

```
Sparsity with C = 100: 12884
Sparsity with C = 10: 10758
Sparsity with C = 1: 5075
Sparsity with C = 0.1: 984
Sparsity with C = 0.01: 145
```

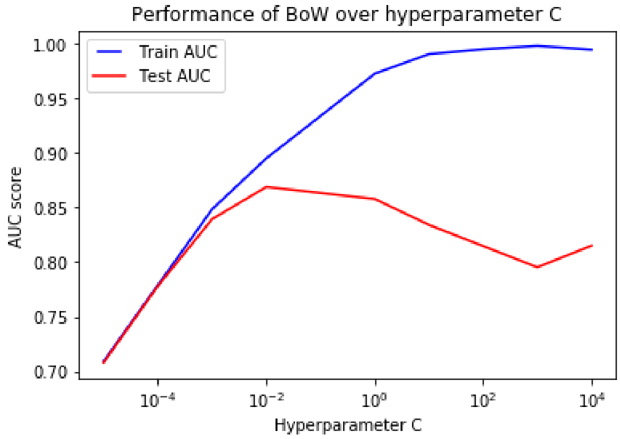
From above results, it is found that as the C value decreases, sparsity also decreases.

In [87]:

```
C = [0.00001, 0.0001, 0.001, 0.01, 1, 10, 100, 1000, 10000]
train_results = []
test_results = []
for i in C:
    clf = LogisticRegression(C=i,class_weight='balanced')
    clf.fit(train_bow, y_tr)
    train_pred = clf.predict(train_bow)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_tr, train_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    train_results.append(roc_auc)
    y_pred = clf.predict(test_bow)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    test_results.append(roc_auc)
from matplotlib.legend_handler import HandlerLine2D
ax = plt.gca()
ax.set_xscale('log')
line1, = ax.plot(C, train_results, 'b', label="Train AUC")
line2, = ax.plot(C, test_results, 'r', label="Test AUC")
plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
plt.title("Performance of BoW over hyperparameter C")
plt.ylabel('AUC score')
```

```
plt.xlabel('Hyperparameter C')
plt.show()
```

```
C:\Users\Dell\AppData\Roaming\Python\Python36\site-packages\sklearn\svm\base.py:929: ConvergenceWarning
: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\Dell\AppData\Roaming\Python\Python36\site-packages\sklearn\svm\base.py:929: ConvergenceWarning
: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:\Users\Dell\AppData\Roaming\Python\Python36\site-packages\sklearn\svm\base.py:929: ConvergenceWarning
: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```



[5.1.3] Feature Importance on BOW, SET 1

[5.1.3.1] Top 10 important features of positive class from SET 1

In [103]:

```
# Please write all the code with proper documentation

weight=clf_l2.coef_
pos_indx=np.argsort(weight)[:,::-1]

neg_indx=np.argsort(weight)

print('Top 10 positive features :')
for i in list(pos_indx[0][0:10]):
    print(all_features[i])
```

```
Top 10 positive features :
pleasantly
worried
satisfied
beat
excellent
welcome
delicious
amazing
hooked
complaint
```

[5.1.3.2] Top 10 important features of negative class from SET 1

In [104]:

```
# Please write all the code with proper documentation
print('Top 10 negative features :')
for i in list(neg_indx[0][:10]):
    print(all_features[i])
```

```
Top 10 negative features :
```

```
worst
cancelled
disappointing
undrinkable
terrible
rip
tasteless
sounded
disappointment
flavorless
```

[5.2] Logistic Regression on TFIDF, SET 2

[5.2.1] Applying Logistic Regression with L1 regularization on TFIDF, SET 2

In [91]:

```
# Please write all the code with proper documentation
tfidf_vect = TfidfVectorizer(ngram_range=(1,2),min_df=10)
tfidf_vect.fit(X_tr['CleanedText'])
train_tfidf = tfidf_vect.transform(X_tr['CleanedText'])
cv_tfidf = tfidf_vect.transform(X_cv['CleanedText'])
test_tfidf = tfidf_vect.transform(X_test['CleanedText'])
print(test_tfidf.shape)
print(cv_tfidf.shape)
print(train_tfidf.shape)
```

```
(17555, 33089)
(14044, 33089)
(56174, 33089)
```

In [120]:

```
#https://machinelearningmastery.com/how-to-tune-algorithm-parameters-with-scikit-learn/
#Applying GridSearch to find the best hyperparameter C

tuned_parameter = [{'C': [0.00001, 0.0001, 0.001, 0.01, 1, 10, 100, 1000, 10000]}]
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(penalty='l1',class_weight='balanced')
scoring = {'AUC': 'roc_auc', 'Accuracy': 'accuracy'}
grid = GridSearchCV(estimator=clf, param_grid = tuned_parameter ,scoring = scoring, refit = 'AUC')
grid.fit(train_tfidf, y_train)
print(grid)
# summarize the results of the grid search
print(grid.best_score_)
print(grid.best_estimator_)
print(grid.score(test_tfidf,y_test))
results = grid.cv_results_
#print(results)
#print(grid.confusion_matrix)
```

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='warn',
                                          n_jobs=None, penalty='l1',
                                          random_state=None, solver='warn',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'C': [1e-05, 0.0001, 0.001, 0.01, 1, 10, 100, 1000,
                                10000]}],
             pre_dispatch='2*n_jobs', refit='AUC', return_train_score=False,
             scoring={'AUC': 'roc_auc', 'Accuracy': 'accuracy'}, verbose=0)
0.9541666203800235
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l1',
```

```
                    random_state=None, solver='warn', tol=0.0001, verbose=0,
                    warm_start=False)
0.9609677727400501
```

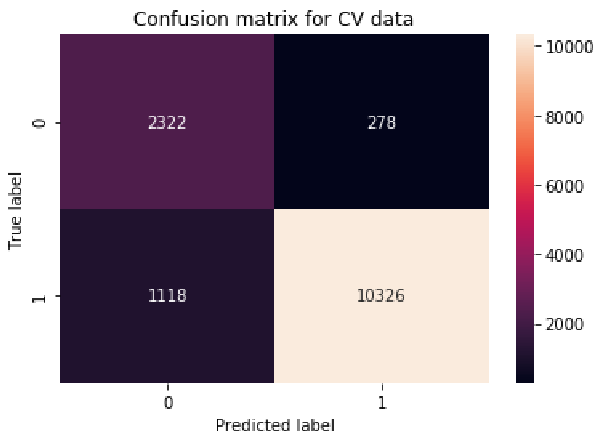
In [92]:

```python
clf = LogisticRegression(C=1, penalty ='l1',class_weight='balanced');
clf.fit(train_tfidf, y_tr)
#conf_matrix = confusion_matrix(y_test,pred)
cclf=clf.predict(cv_tfidf)
pred_cv = clf.predict_proba(cv_tfidf)[:,1]
#print('alpha value = ',1)
#acc = accuracy_score(y_test,pred_test)*100
#print("Accuracy",acc)
fpr, tpr, thresholds = roc_curve(y_cv,pred_cv)
roc_auc_cv = auc(fpr, tpr)
print('Area under the ROC curve : %f', + roc_auc_cv)
#Plotting confusion matrix
import seaborn as sns
conf_mat = confusion_matrix(y_cv, cclf)
print(conf_mat)
#conf_normalized = conf_mat.astype('int') / conf_mat.sum(axis=1)[:, np.newaxis]
sns.heatmap(conf_mat, annot=True, fmt ='g')
plt.title('Confusion matrix for CV data')
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

```
Area under the ROC curve : %f 0.9616383459253087
[[ 2322   278]
 [ 1118 10326]]
```

Out[92]:

```
Text(0.5, 15.0, 'Predicted label')
```



In [93]:

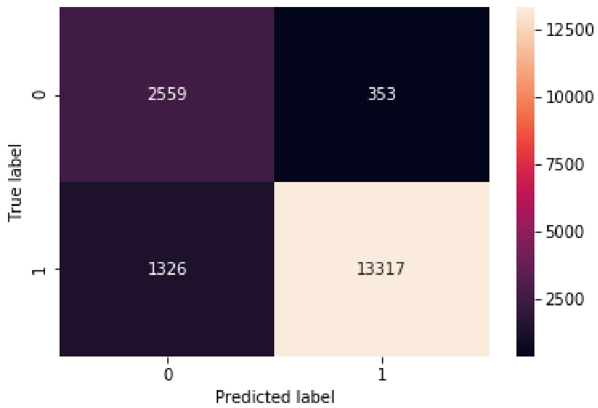
```python
clf = LogisticRegression(C=1, penalty ='l1',class_weight='balanced');
clf.fit(train_tfidf, y_tr)
#w = clf.coef_
#print("Sparsity:",np.count_nonzero(w))

#conf_matrix = confusion_matrix(y_test,pred)
cclf=clf.predict(test_tfidf)
pred_test = clf.predict_proba(test_tfidf)[:,1]
#print('alpha value = ',1)
#acc = accuracy_score(y_test,pred_test)*100
#print("Accuracy",acc)
fpr, tpr, thresholds = roc_curve(y_test,pred_test)
roc_auc_test = auc(fpr, tpr)
print('Area under the ROC curve : %f', + roc_auc_test)
#Plotting confusion matrix
import seaborn as sns
conf_mat = confusion_matrix(y_test, cclf)
print(conf_mat)
#conf_normalized = conf_mat.astype('int') / conf_mat.sum(axis=1)[:, np.newaxis]
sns.heatmap(conf_mat, annot=True, fmt ='g')
```

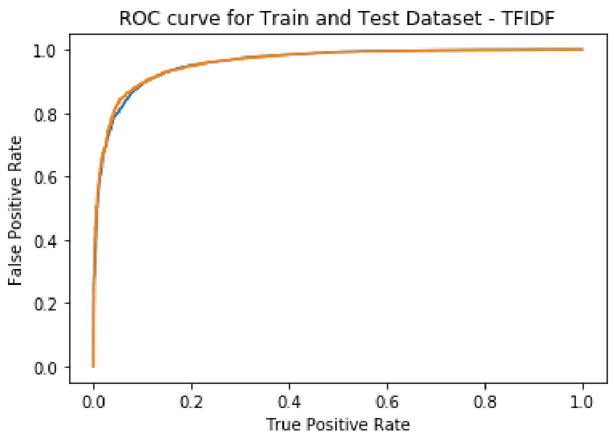
```
plt.ylabel('True label')
plt.xlabel('Predicted label')


#Plot ROC Curve
plt.figure(0).clf()
fpr, tpr, thresholds = roc_curve(y_test,pred_test)
roc_auc_test = auc(fpr, tpr)
plt.plot(fpr,tpr,label="Test Data, auc="+str(roc_auc_test))
fpr, tpr, thresh = roc_curve(y_cv, pred_cv)
roc_auc_cv = auc(fpr, tpr)
plt.plot(fpr,tpr,label="Train Data, auc="+str(roc_auc_cv))
plt.title('ROC curve for Train and Test Dataset - TFIDF')
plt.xlabel('True Positive Rate')
plt.ylabel('False Positive Rate')
```

```
Area under the ROC curve : %f 0.9600346066042132
[[ 2559   353]
 [ 1326 13317]]
```

Out[93]:

```
Text(0, 0.5, 'False Positive Rate')
```





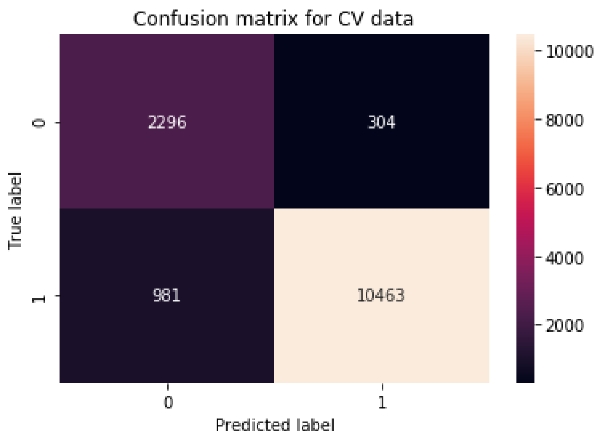
[5.2.2] Applying Logistic Regression with L2 regularization on TFIDF, SET 2

In [94]:

```
clf = LogisticRegression(C=1, penalty ='l2',class_weight='balanced');
clf.fit(train_tfidf, y_tr)
#conf_matrix = confusion_matrix(y_test,pred)
cclf=clf.predict(cv_tfidf)
pred_cv = clf.predict_proba(cv_tfidf)[:,1]
#print('alpha value = ',1)
#acc = accuracy_score(y_test,pred_test)*100
#print("Accuracy",acc)
fpr, tpr, thresholds = roc_curve(y_cv,pred_cv)
roc_auc_cv = auc(fpr, tpr)
print('Area under the ROC curve : %f', + roc_auc_cv)
#Plotting confusion matrix
import seaborn as sns
```

```
conf_mat = confusion_matrix(y_cv, cclf)
print(conf_mat)
#conf_normalized = conf_mat.astype('int') / conf_mat.sum(axis=1)[:, np.newaxis]
sns.heatmap(conf_mat, annot=True, fmt ='g')
plt.title('Confusion matrix for CV data')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

```
Area under the ROC curve : %f 0.9622826539940311
[[ 2296   304]
 [  981 10463]]
```



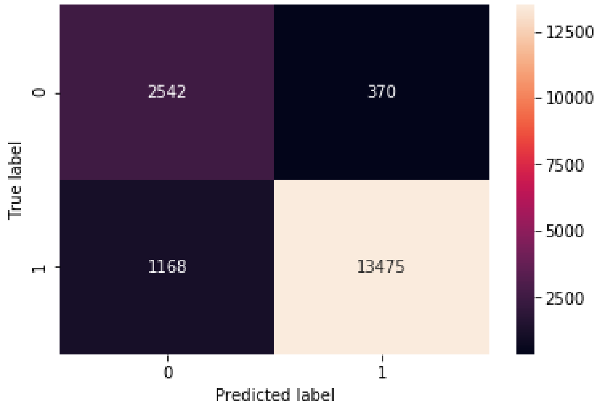
In [95]:

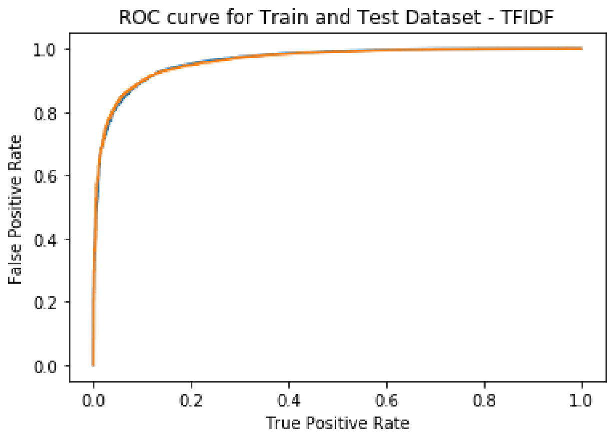
```
clf = LogisticRegression(C=1, penalty ='l2',class_weight='balanced');
clf.fit(train_tfidf, y_tr)
#w = clf.coef_
#print("Sparsity:",np.count_nonzero(w))

#conf_matrix = confusion_matrix(y_test,pred)
cclf=clf.predict(test_tfidf)
pred_test = clf.predict_proba(test_tfidf)[:,1]
#print('alpha value = ',1)
#acc = accuracy_score(y_test,pred_test)*100
#print("Accuracy",acc)
fpr, tpr, thresholds = roc_curve(y_test,pred_test)
roc_auc_test = auc(fpr, tpr)
print('Area under the ROC curve : %f', + roc_auc_test)
#Plotting confusion matrix
import seaborn as sns
conf_mat = confusion_matrix(y_test, cclf)
print(conf_mat)
#conf_normalized = conf_mat.astype('int') / conf_mat.sum(axis=1)[:, np.newaxis]
sns.heatmap(conf_mat, annot=True, fmt ='g')
plt.ylabel('True label')
plt.xlabel('Predicted label')


#Plot ROC Curve
plt.figure(0).clf()
fpr, tpr, thresholds = roc_curve(y_test,pred_test)
roc_auc_test = auc(fpr, tpr)
plt.plot(fpr,tpr,label="Test Data, auc="+str(roc_auc_test))
fpr, tpr, thresh = roc_curve(y_cv, pred_cv)
roc_auc_cv = auc(fpr, tpr)
plt.plot(fpr,tpr,label="Train Data, auc="+str(roc_auc_cv))
plt.title('ROC curve for Train and Test Dataset - TFIDF')
plt.xlabel('True Positive Rate')
plt.ylabel('False Positive Rate')
```

```
Area under the ROC curve : %f 0.9625707919922731
[[ 2542   370]
 [ 1168 13475]]
```

Out[95]:

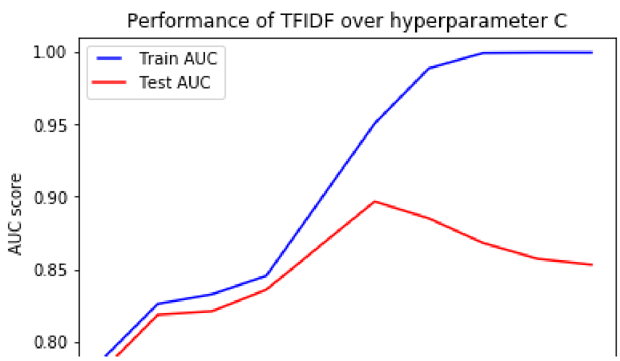
Text(0, 0.5, 'False Positive Rate')

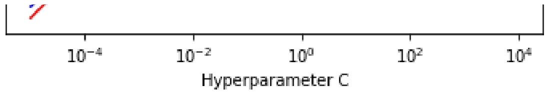




In [97]:

```
C = [0.00001, 0.0001, 0.001, 0.01, 1, 10, 100, 1000, 10000]
train_results = []
test_results = []
for i in C:
    clf = LogisticRegression(C=i,class_weight='balanced')
    clf.fit(train_tfidf, y_tr)
    train_pred = clf.predict(train_tfidf)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_tr, train_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    train_results.append(roc_auc)
    y_pred = clf.predict(test_tfidf)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    test_results.append(roc_auc)
from matplotlib.legend_handler import HandlerLine2D
ax = plt.gca()
ax.set_xscale('log')
line1, = ax.plot(C, train_results, 'b', label="Train AUC")
line2, = ax.plot(C, test_results, 'r', label="Test AUC")
plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
plt.title("Performance of TFIDF over hyperparameter C")
plt.ylabel('AUC score')
plt.xlabel('Hyperparameter C')
plt.show()
```





[5.2.3] Feature Importance on TFIDF, SET 2

[5.2.3.1] Top 10 important features of positive class from SET 2

In [123]:

```
# Please write all the code with proper documentation
import matplotlib.pyplot as plt
def plot_coefficients(classifier, feature_names, top_features=20):
    coef = classifier.coef_.ravel()
    top_positive_coefficients = np.argsort(coef)[-top_features:]
    top_negative_coefficients = np.argsort(coef)[:top_features]
    top_coefficients = np.hstack([top_negative_coefficients, top_positive_coefficients])
    feature_names = np.array(feature_names)
    print("Positive coefficients:",feature_names[top_positive_coefficients])
    print("Negative coefficients:",feature_names[top_negative_coefficients])

plot_coefficients(clf, tfidf_vect.get_feature_names(), top_features=20)
```

```
Positive coefficients: ['without' 'yummy' 'definitely' 'tasty' 'awesome' 'happy' 'easy'
 'favorite' 'amazing' 'not disappointed' 'wonderful' 'nice' 'excellent'
 'loves' 'love' 'perfect' 'good' 'delicious' 'best' 'great']
Negative coefficients: ['disappointed' 'not' 'worst' 'disappointing' 'not worth' 'terrible'
 'not good' 'awful' 'horrible' 'not recommend' 'not buy' 'disappointment'
 'unfortunately' 'bad' 'threw' 'stale' 'weak' 'return' 'money' 'bland']
```

[5.2.3.2] Top 10 important features of negative class from SET 2

In [0]:

```
# Please write all the code with proper documentation
```

[5.3] Logistic Regression on AVG W2V, SET 3

[5.3.1] Applying Logistic Regression with L1 regularization on AVG W2V SET 3

In [133]:

```
# Please write all the code with proper documentation
#https://machinelearningmastery.com/how-to-tune-algorithm-parameters-with-scikit-learn/
#Applying GridSearch to find the best hyperparameter C

tuned_parameter = [{'C': [0.00001, 0.0001, 0.001, 0.01, 1, 10, 100, 1000, 10000]}]
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(penalty='l1',class_weight='balanced')
scoring = {'AUC': 'roc_auc', 'Accuracy': 'accuracy'}
grid = GridSearchCV(estimator=clf, param_grid = tuned_parameter ,scoring = scoring, refit = 'AUC')
grid.fit(sent_vectors, y_train)
print(grid)
# summarize the results of the grid search
print(grid.best_score_)
print(grid.best_estimator_)
print(grid.score(sent_vectors_test,y_test))
results = grid.cv_results_
#print(results)
#print(grid.confusion_matrix)
```

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
```

```
                                          max_iter=100, multi_class='warn',
                                          n_jobs=None, penalty='l1',
                                          random_state=None, solver='warn',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'C': [1e-05, 0.0001, 0.001, 0.01, 1, 10, 100, 1000,
                                10000]}],
             pre_dispatch='2*n_jobs', refit='AUC', return_train_score=False,
             scoring={'AUC': 'roc_auc', 'Accuracy': 'accuracy'}, verbose=0)
0.9080226568236914
LogisticRegression(C=100, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l1',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
0.904455887944814
```

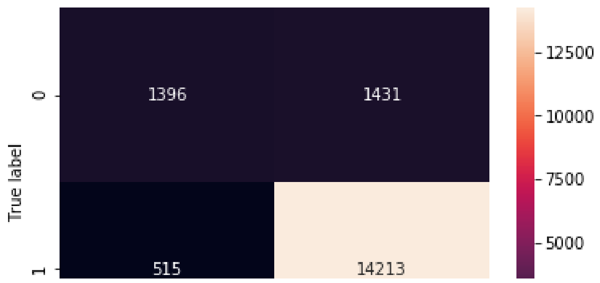
In [136]:

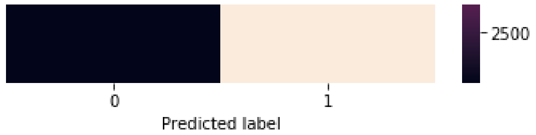
```python
clf = LogisticRegression(C=100, penalty ='l1');
clf.fit(sent_vectors, y_train)
#w = clf.coef_
#print("Sparsity:",np.count_nonzero(w))

#conf_matrix = confusion_matrix(y_test,pred)
cclf=clf.predict(sent_vectors_test)
pred_test = clf.predict_proba(sent_vectors_test)[:,1]
#print('alpha value = ',1)
#acc = accuracy_score(y_test,pred_test)*100
#print("Accuracy",acc)
fpr, tpr, thresholds = roc_curve(y_test,pred_test)
roc_auc_cv = auc(fpr, tpr)
print('Area under the ROC curve : %f', + roc_auc_cv)
#Plotting confusion matrix
import seaborn as sns
conf_mat = confusion_matrix(y_test, cclf)
print(conf_mat)
#conf_normalized = conf_mat.astype('int') / conf_mat.sum(axis=1)[:, np.newaxis]
sns.heatmap(conf_mat, annot=True, fmt ='g')
plt.ylabel('True label')
plt.xlabel('Predicted label')

#Plot ROC Curve
plt.figure(0).clf()
fpr, tpr, thresholds = roc_curve(y_test,pred_test)
roc_auc_test = auc(fpr, tpr)
plt.plot(fpr,tpr,label="Test Data, auc="+str(roc_auc_test))
fpr, tpr, thresh = roc_curve(y_test, pred_test)
roc_auc_cv = auc(fpr, tpr)
plt.plot(fpr,tpr,label="Train Data, auc="+str(roc_auc_cv))
plt.title('ROC curve for Train and Test Dataset - Tfidf')
plt.xlabel('True Positive Rate')
plt.ylabel('False Positive Rate')
plt.legend(loc=0)
```

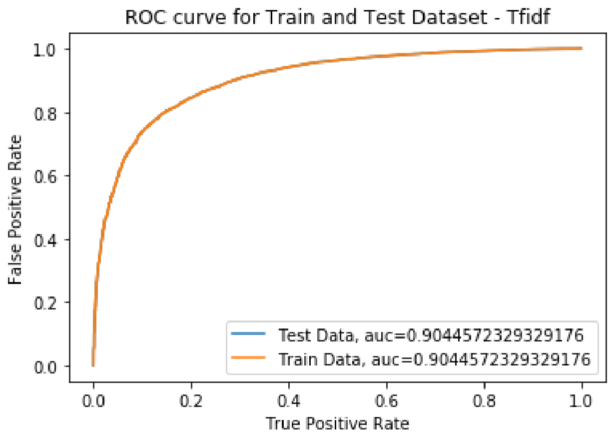
```
Area under the ROC curve : %f 0.9044572329329176
[[ 1396  1431]
 [  515 14213]]
```

Out[136]:

```
<matplotlib.legend.Legend at 0x64453d30>
```







[5.3.2] Applying Logistic Regression with L2 regularization on AVG W2V, SET 3

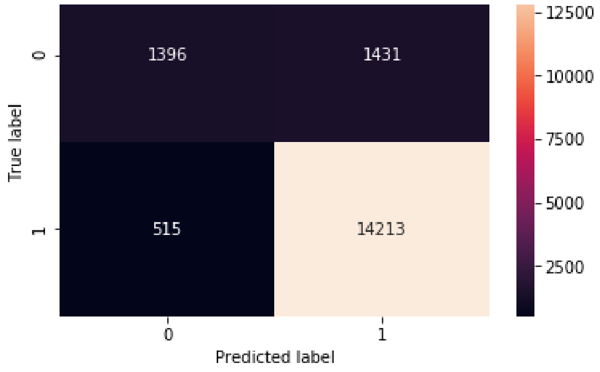
In [137]:

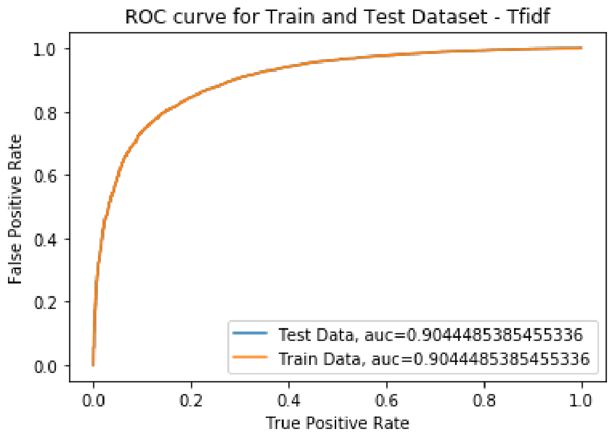
```
# Please write all the code with proper documentation
clf = LogisticRegression(C=100, penalty ='l2');
clf.fit(sent_vectors, y_train)
#w = clf.coef_
#print("Sparsity:",np.count_nonzero(w))

#conf_matrix = confusion_matrix(y_test,pred)
cclf=clf.predict(sent_vectors_test)
pred_test = clf.predict_proba(sent_vectors_test)[:,1]
#print('alpha value = ',1)
#acc = accuracy_score(y_test,pred_test)*100
#print("Accuracy",acc)
fpr, tpr, thresholds = roc_curve(y_test,pred_test)
roc_auc_cv = auc(fpr, tpr)
print('Area under the ROC curve : %f', + roc_auc_cv)
#Plotting confusion matrix
import seaborn as sns
conf_mat = confusion_matrix(y_test, cclf)
print(conf_mat)
#conf_normalized = conf_mat.astype('int') / conf_mat.sum(axis=1)[:, np.newaxis]
sns.heatmap(conf_mat, annot=True, fmt ='g')
plt.ylabel('True label')
plt.xlabel('Predicted label')

#Plot ROC Curve
plt.figure(0).clf()
fpr, tpr, thresholds = roc_curve(y_test,pred_test)
roc_auc_test = auc(fpr, tpr)
plt.plot(fpr,tpr,label="Test Data, auc="+str(roc_auc_test))
fpr, tpr, thresh = roc_curve(y_test, pred_test)
roc_auc_cv = auc(fpr, tpr)
plt.plot(fpr,tpr,label="Train Data, auc="+str(roc_auc_cv))
plt.title('ROC curve for Train and Test Dataset - Tfidf')
plt.xlabel('True Positive Rate')
plt.ylabel('False Positive Rate')
plt.legend(loc=0)
```

```
Area under the ROC curve : %f 0.9044485385455336
[[ 1396  1431]
 [  515 14213]]
```

Out[137]:

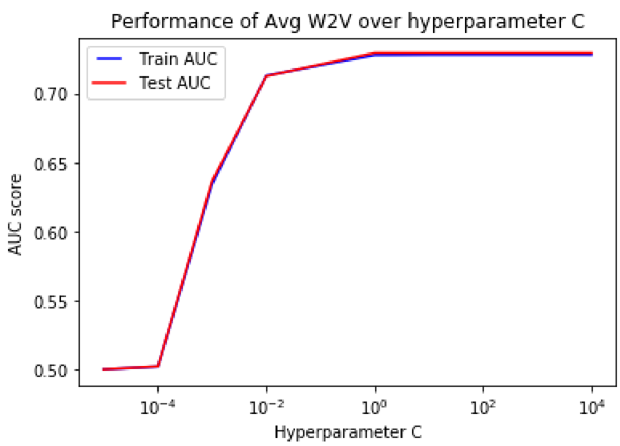
```
<matplotlib.legend.Legend at 0x645a3f28>
```





In [142]:

```
C = [0.00001, 0.0001, 0.001, 0.01, 1, 10, 100, 1000, 10000]
train_results = []
test_results = []
for i in C:
    clf = LogisticRegression(C=i)
    clf.fit(sent_vectors, y_train)
    train_pred = clf.predict(sent_vectors)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, train_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    train_results.append(roc_auc)
    y_pred = clf.predict(sent_vectors_test)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    test_results.append(roc_auc)
from matplotlib.legend_handler import HandlerLine2D
ax = plt.gca()
ax.set_xscale('log')
line1, = ax.plot(C, train_results, 'b', label="Train AUC")
line2, = ax.plot(C, test_results, 'r', label="Test AUC")
plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
plt.title("Performance of Avg W2V over hyperparameter C")
plt.ylabel('AUC score')
plt.xlabel('Hyperparameter C')
plt.show()
```



[5.4] Logistic Regression on TFIDF W2V, SET 4

[5.4.1] Applying Logistic Regression with L1 regularization on TFIDF W2V, SET 4

In [143]:

```
#https://machinelearningmastery.com/how-to-tune-algorithm-parameters-with-scikit-learn/
#Applying GridSearch to find the best hyperparameter C

tuned_parameter = [{'C': [0.00001, 0.0001, 0.001, 0.01, 1, 10, 100, 1000, 10000]}]
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(penalty='l1')
scoring = {'AUC': 'roc_auc', 'Accuracy': 'accuracy'}
grid = GridSearchCV(estimator=clf, param_grid = tuned_parameter ,scoring = scoring, refit = 'AUC')
grid.fit(tfidf_sent_vectors, y_train)
print(grid)
# summarize the results of the grid search
print(grid.best_score_)
print(grid.best_estimator_)
print(grid.score(tfidf_sent_vectors_test,y_test))
results = grid.cv_results_
#print(results)
#print(grid.confusion_matrix)
```

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='warn',
                                          n_jobs=None, penalty='l1',
                                          random_state=None, solver='warn',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'C': [1e-05, 0.0001, 0.001, 0.01, 1, 10, 100, 1000,
                                10000]}],
             pre_dispatch='2*n_jobs', refit='AUC', return_train_score=False,
             scoring={'AUC': 'roc_auc', 'Accuracy': 'accuracy'}, verbose=0)
0.8841530041400345
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l1',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
0.8799224883355906
```

In [144]:

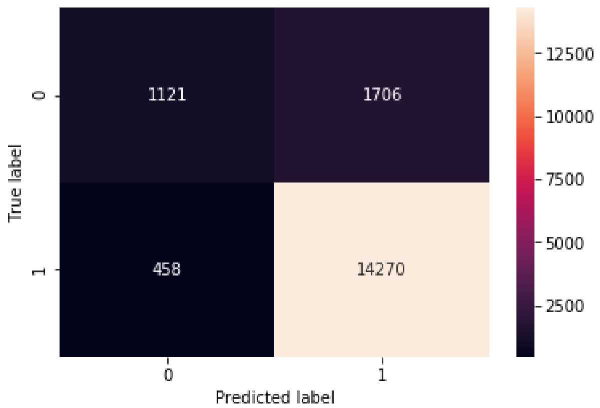
```
clf = LogisticRegression(C=1, penalty ='l1');
clf.fit(tfidf_sent_vectors, y_train)
#w = clf.coef_
#print("Sparsity:",np.count_nonzero(w))

#conf_matrix = confusion_matrix(y_test,pred)
cclf=clf.predict(tfidf_sent_vectors_test)
pred_test = clf.predict_proba(tfidf_sent_vectors_test)[:,1]
#print('alpha value = ',1)
#acc = accuracy_score(y_test,pred_test)*100
#print("Accuracy",acc)
fpr, tpr, thresholds = roc_curve(y_test,pred_test)
roc_auc_cv = auc(fpr, tpr)
print('Area under the ROC curve : %f', + roc_auc_cv)
#Plotting confusion matrix
import seaborn as sns
conf_mat = confusion_matrix(y_test, cclf)
print(conf_mat)
#conf_normalized = conf_mat.astype('int') / conf_mat.sum(axis=1)[:, np.newaxis]
sns.heatmap(conf_mat, annot=True, fmt ='g')
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

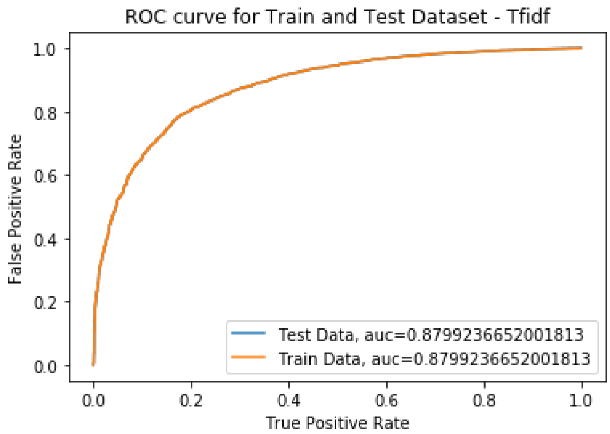
```
#Plot ROC Curve
plt.figure(0).clf()
fpr, tpr, thresholds = roc_curve(y_test,pred_test)
roc_auc_test = auc(fpr, tpr)
plt.plot(fpr,tpr,label="Test Data, auc="+str(roc_auc_test))
fpr, tpr, thresh = roc_curve(y_test, pred_test)
roc_auc_cv = auc(fpr, tpr)
plt.plot(fpr,tpr,label="Train Data, auc="+str(roc_auc_cv))
plt.title('ROC curve for Train and Test Dataset - Tfidf')
plt.xlabel('True Positive Rate')
plt.ylabel('False Positive Rate')
plt.legend(loc=0)
```

```
Area under the ROC curve : %f 0.8799236652001813
[[ 1121  1706]
 [  458 14270]]
```

Out[144]:

```
<matplotlib.legend.Legend at 0x6926d9b0>
```





[5.4.2] Applying Logistic Regression with L2 regularization on TFIDF W2V, SET 4

In [145]:

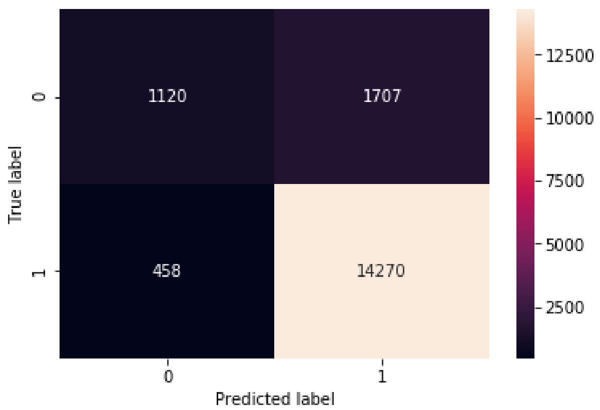
```
# Please write all the code with proper documentation
clf = LogisticRegression(C=1, penalty ='l2');
clf.fit(tfidf_sent_vectors, y_train)
#w = clf.coef_
#print("Sparsity:",np.count_nonzero(w))

#conf_matrix = confusion_matrix(y_test,pred)
cclf=clf.predict(tfidf_sent_vectors_test)
pred_test = clf.predict_proba(tfidf_sent_vectors_test)[:,1]
#print('alpha value = ',1)
#acc = accuracy_score(y_test,pred_test)*100
#print("Accuracy",acc)
fpr, tpr, thresholds = roc_curve(y_test,pred_test)
roc_auc_cv = auc(fpr, tpr)
print('Area under the ROC curve : %f', + roc_auc_cv)
#Plotting confusion matrix
```

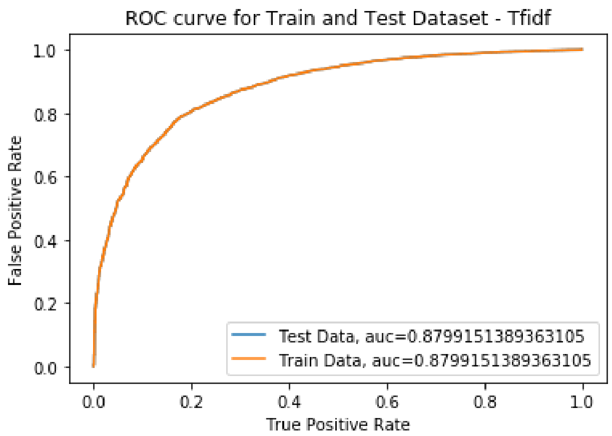
```
#Plotting confusion matrix
import seaborn as sns
conf_mat = confusion_matrix(y_test, cclf)
print(conf_mat)
#conf_normalized = conf_mat.astype('int') / conf_mat.sum(axis=1)[:, np.newaxis]
sns.heatmap(conf_mat, annot=True, fmt ='g')
plt.ylabel('True label')
plt.xlabel('Predicted label')

#Plot ROC Curve
plt.figure(0).clf()
fpr, tpr, thresholds = roc_curve(y_test,pred_test)
roc_auc_test = auc(fpr, tpr)
plt.plot(fpr,tpr,label="Test Data, auc="+str(roc_auc_test))
fpr, tpr, thresh = roc_curve(y_test, pred_test)
roc_auc_cv = auc(fpr, tpr)
plt.plot(fpr,tpr,label="Train Data, auc="+str(roc_auc_cv))
plt.title('ROC curve for Train and Test Dataset - Tfidf')
plt.xlabel('True Positive Rate')
plt.ylabel('False Positive Rate')
plt.legend(loc=0)
```

```
Area under the ROC curve : %f 0.8799151389363105
[[ 1120  1707]
 [  458 14270]]
```

Out[145]:

```
<matplotlib.legend.Legend at 0x692d5b70>
```

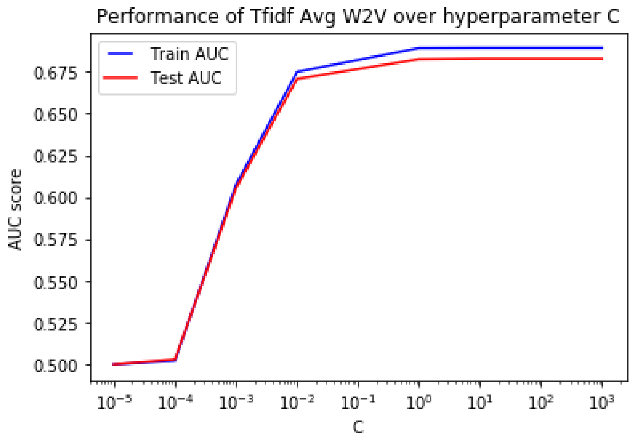




In [149]:

```
#Performance of Tfidf Avg w2v over hyperparameter C
C = [0.00001, 0.0001, 0.001, 0.01, 1, 10, 100, 1000]
train_results = []
test_results = []
for i in C:
    clf = LogisticRegression(C=i)
    clf.fit(tfidf_sent_vectors, y_train)
    train_pred = clf.predict(tfidf_sent_vectors)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, train_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
```

```
    train_results.append(roc_auc)
    y_pred = clf.predict(tfidf_sent_vectors_test)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    test_results.append(roc_auc)
from matplotlib.legend_handler import HandlerLine2D
ax = plt.gca()
ax.set_xscale('log')
line1, = ax.plot(C, train_results, 'b', label="Train AUC")
line2, = ax.plot(C, test_results, 'r', label="Test AUC")
plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
plt.title("Performance of Tfidf Avg W2V over hyperparameter C")
plt.ylabel('AUC score')
plt.xlabel('C')
plt.show()
```



[6] Conclusions

In [151]:

```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
table = PrettyTable(["model","C value","Test AUC"])
table.add_row(["LR using BoW", "1",0.934])
table.add_row(["LR using TFIDF","1",0.964])
table.add_row(["LR using AVG W2V","100",0.904])
table.add_row(["LR using TFIDF AVG W2V","1",0.8799])
print(table)
```

```
+------------------------+---------+----------+
|         model          | C value | Test AUC |
+------------------------+---------+----------+
|      LR using BoW      |    1    |  0.934   |
|     LR using TFIDF     |    1    |  0.964   |
|    LR using AVG W2V    |   100   |  0.904   |
| LR using TFIDF AVG W2V |    1    |  0.8799  |
+------------------------+---------+----------+
```

1. Logistic Regression is applied on Amazon food review dataset with four different vectorization techniques.
2. The hyperparameter C is tuned using GridSearchCV and the test results are predicted.
3. Various C values are used to predict the sparsity in BoW with L1 regularizer. It is found that the sparsity(number of non zero vectors)decreases as C value decreases. L1 and L2 regularisations are used.
4. Perturbation technique is applied to check for multicollinearity.
5. From the above table, AUC score for BoW and TFIDF are better than the rest.