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 [2]:

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



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

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

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

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

from tqdm import tqdm
import os
```

```
C:\Users\nairh\Anaconda3\lib\site-packages\sma
rt_open\ssh.py:34: UserWarning: paramiko missi
ng, opening SSH/SCP/SFTP paths will be disable
d.  `pip install paramiko` to suppress
  warnings.warn('paramiko missing, opening SSH
/SCP/SFTP paths will be disabled.  `pip instal
l paramiko` to suppress')
C:\Users\nairh\Anaconda3\lib\site-packages\gen
sim\utils.py:1197: UserWarning: detected Windo
ws; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing ch
unkize to chunkize_serial")
```

In [3]:

```
# 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 g
ive 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 W
HERE 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 v
ice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shap
e)
filtered_data.head(3)
```

```
Number of data points in our data (100000, 10)
```

Out[3]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Hel
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	

1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1

In [4]:

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

In [5]:

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

```
(80668, 7)
```

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Tex
0	#oc-R115TNMSPFT9I7	B005ZBZLT4	Breyton	1331510400	2	Overall its just OK when considering the price...
1	#oc-R11D9D7SHXIJB9	B005HG9ESG	Louis E. Emory	1342396800	5	My wife ha recurrin extreme

			"hoppy"			muscle spasms, u..
2	#oc-R11DNU2NBKQ23Z	B005ZBZLT4	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...
3	#oc-R11O5J5ZVQE25C	B005HG9ESG	Penguin Chick	1346889600	5	This will be the bottle that you grab from the..
4	#oc-R12KPBODL2B5ZD	B007OSBEV0	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...

In [6]:

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

Out[6]:

	UserId	ProductId	ProfileName	Time	Score	Tex
80638	AZY10LLTJ71NX	B001ATMQK2	undertheshrine "undertheshrine"	1296691200	5	I bought this 6 pack because for the price tha...

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 [8]:

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

Out[8]:

	Id	ProductId	UserId	ProfileName
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan

4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2

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 [9]:

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, as
cending=True, inplace=False, kind='quicksort', na_position='l
ast')
```

In [10]:

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

Out[10]:

```
(87775, 10)
```

In [11]:

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

Out[11]:

```
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 [12]:

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

display.head()
```

Out[12]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator
0				J. E.	

	64422	B000MIDROQ	A161DK06JJMCYF	Stephens "Jeanne"	3
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3

In [13]:

```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDeno
minator]
```

In [14]:

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

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

```
(87773, 10)
```

Out[14]:

```
1    73592
0    14181
Name: Score, dtype: int64
```

In [46]:

```
timesorted_data=final.sort_values('Time') #, axis=0, ascendin
g=True, inplace=False, kind='quicksort', na_position='last')
```

In [53]:

```
print(timesorted_data.shape)
```

```
(87773, 10)
```

[3] Preprocessing

In [ ]:

```

```

[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 [15]:

```
# 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)
```

```python
sent_4900 = final['Text'].values[4900]
print(sent_4900)
print("="*50)
```

```
My dogs loves this chicken but its a product f
rom China, so we wont be buying it anymore.  I
ts very hard to find any chicken products made
 in the USA but they are out there, but this o
ne isnt.  Its too bad too because its a good p
roduct but I wont take any chances till they k
now what is going on with the china imports.
==============================================
====
The Candy Blocks were a nice visual for the Le
go Birthday party but the candy has little tas
te to it.  Very little of the 2 lbs that I bou
ght were eaten and I threw the rest away.  I w
ould not buy the candy again.
==============================================
====
was way to hot for my blood, took a bite and d
id a jig  lol
==============================================
====
My dog LOVES these treats. They tend to have a
 very strong fish oil smell. So if you are afr
aid of the fishy smell, don't get it. But I th
ink my dog likes it because of the smell. Thes
e treats are really small in size. They are gr
eat for training. You can give your dog severa
l of these without worrying about him over eat
ing. Amazon's price was much more reasonable t
han any other retailer. You can buy a 1 pound
bag on Amazon for almost the same price as a 6
 ounce bag at other retailers. It's definitely
 worth it to buy a big bag if your dog eats th
```

```
em a lot.
==================================================
====
```

In [16]:

```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)
```

```
My dogs loves this chicken but its a product f
rom China, so we wont be buying it anymore.  I
ts very hard to find any chicken products made
 in the USA but they are out there, but this o
ne isnt.  Its too bad too because its a good p
roduct but I wont take any chances till they k
now what is going on with the china imports.
```

In [17]:

```python
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-an-element
from bs4 import BeautifulSoup

soup = BeautifulSoup(sent_0, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1000, 'lxml')
text = soup.get_text()
print(text)
print("="*50)
```

```
soup = BeautifulSoup(sent_1500, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_4900, 'lxml')
text = soup.get_text()
print(text)
```

```
My dogs loves this chicken but its a product f
rom China, so we wont be buying it anymore.  I
ts very hard to find any chicken products made
 in the USA but they are out there, but this o
ne isnt.  Its too bad too because its a good p
roduct but I wont take any chances till they k
now what is going on with the china imports.
==============================================
====
The Candy Blocks were a nice visual for the Le
go Birthday party but the candy has little tas
te to it.  Very little of the 2 lbs that I bou
ght were eaten and I threw the rest away.  I w
ould not buy the candy again.
==============================================
====
was way to hot for my blood, took a bite and d
id a jig  lol
==============================================
====
My dog LOVES these treats. They tend to have a
 very strong fish oil smell. So if you are afr
aid of the fishy smell, don't get it. But I th
ink my dog likes it because of the smell. Thes
e treats are really small in size. They are gr
eat for training. You can give your dog severa
l of these without worrying about him over eat
```

```
ing. Amazon's price was much more reasonable t
han any other retailer. You can buy a 1 pound
bag on Amazon for almost the same price as a 6
 ounce bag at other retailers. It's definitely
 worth it to buy a big bag if your dog eats th
em a lot.
```

In [18]:

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

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

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

In [19]:

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

```
was way to hot for my blood, took a bite and d
id a jig  lol
==============================================
====
```

In [20]:

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

```
My dogs loves this chicken but its a product f
rom China, so we wont be buying it anymore.  I
ts very hard to find any chicken products made
 in the USA but they are out there, but this o
ne isnt.  Its too bad too because its a good p
roduct but I wont take any chances till they k
now what is going on with the china imports.
```

In [21]:

```
#remove spacial character: https://stackoverflow.com/a/584354
7/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
was way to hot for my blood took a bite and di
d a jig lol
```

In [22]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', '
nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br
br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have re
vmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we',
 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', '
yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "
it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', '
whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', '
being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', '
if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'b
etween', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in
', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where',
 'why', 'how', 'all', 'any', 'both', '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', "co
uldn'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 [23]:

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

```
100%|██████████| 87773/87773 [00:52<00:00, 165
7.56it/s]
```

In [0]:

```
preprocessed_reviews[1500]
```

Out[0]:

```
'wow far two two star reviews one obviously no
 idea ordering wants crispy cookies hey sorry
reviews nobody good beyond reminding us look o
rdering chocolate oatmeal cookies not like com
bination not order type cookie find combo quit
e nice really oatmeal sort calms rich chocolat
e flavor gives cookie sort coconut type consis
tency let also remember tastes differ given op
inion soft chewy cookies advertised not crispy
 cookies blurb would say crispy rather chewy h
appen like raw cookie dough however not see ta
ste like raw cookie dough soft however confusi
on yes stick together soft cookies tend not in
dividually wrapped would add cost oh yeah choc
olate chip cookies tend somewhat sweet want so
mething hard crisp suggest nabiso ginger snaps
 want cookie soft chewy tastes like combinatio
n chocolate oatmeal give try place second orde
r'
```

In [24]:

```
#Random splitting of data to separate train, cv and test data

Y = final['Score'].values
X = preprocessed_reviews

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, tes
t_size=0.33) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_tr
ain, test_size=0.33) # this is random splitting

print(len(X_train), y_train.shape)
print(len(X_cv), y_cv.shape)
print(len(X_test), y_test.shape)
```

```
39400 (39400,)
19407 (19407,)
28966 (28966,)
```

[3.2] Preprocessing Review Summary

In [0]:

```
## Similartly you can do preprocessing for review summary als
o.
```

[4] Featurization

[4.1] BAG OF WORDS

In [25]:

```
#BoW
count_vect = CountVectorizer(min_df=10)
#count_vect = CountVectorizer(min_df=10, max_features=500) #
for kd tree
 #in scikit-learn
count_vect.fit(X_train)
print("some feature names ", count_vect.get_feature_names()[:
10])
print('='*50)

final_counts_train = count_vect.transform(X_train)
final_counts_cv = count_vect.transform(X_cv)
final_counts_test = count_vect.transform(X_test)
#print("the type of count vectorizer ",type(final_counts))
#print("the shape of out text BOW vectorizer ",final_counts.g
et_shape())
print("the number of unique words ", final_counts_train.get_s
hape()[1])
```

```
some feature names  ['ability', 'able', 'absen
ce', 'absent', 'absolute', 'absolutely', 'abso
lutly', 'absorb', 'absorbed', 'absorption']
==============================================
====
the number of unique words  7739
```

[4.2] Bi-Grams and n-Grams.

In [0]:

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

#removing stop words like "not" should be avoided before buil
ding n-grams
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation http://sci
kit-learn.org/stable/modules/generated/sklearn.feature_extrac
tion.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, ma
x_features=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_r
eviews)
print("the type of count vectorizer ",type(final_bigram_count
s))
print("the shape of out text BOW vectorizer ",final_bigram_co
unts.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.sp
arse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 3
144)
the number of unique words including both unig
rams and bigrams  3144
```

[4.3] TF-IDF

In [26]:

```
#tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10,
max_features=500) # for kd tree
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(X_train)
print("some sample features(unique words in the corpus)",tf_i
df_vect.get_feature_names()[0:10])
print('='*50)

final_tf_idf_train = tf_idf_vect.transform(X_train)
final_tf_idf_cv = tf_idf_vect.transform(X_cv)
final_tf_idf_test = tf_idf_vect.transform(X_test)

#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 an
d bigrams ", final_tf_idf.get_shape()[1])
```

```
some sample features(unique words in the corpu
s) ['ability', 'able', 'able buy', 'able drink
', 'able eat', 'able enjoy', 'able figure', 'a
ble find', 'able finish', 'able get']
==============================================
====
```

[4.4] Word2Vec

In [27]:

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

In [28]:

```
# 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[wo
rd] as values
# To use this code-snippet, download "GoogleNews-vectors-nega
tive300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS
21pQmM/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
```

```python
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('GoogleNe
ws-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 wa
nt_to_train_w2v = True, to train your own w2v ")
```

```
[('fantastic', 0.8449681997299194), ('awesome'
, 0.83317631483078), ('good', 0.82166898250579
83), ('excellent', 0.7990680932998657), ('terr
ific', 0.7915112972259521), ('perfect', 0.7635
703682899475), ('wonderful', 0.763299226760864
3), ('amazing', 0.7176131010055542), ('nice',
0.710258960723877), ('decent', 0.7000181674957
275)]
==============================================
====
[('greatest', 0.8191127777099609), ('best', 0.
7109125852584839), ('tastiest', 0.699251055717
4683), ('disgusting', 0.6615686416625977), ('n
astiest', 0.6208571791648865), ('smoothest', 0
.6163804531097412), ('terrible', 0.59675610065
4602), ('awful', 0.5936288833618164), ('hottes
t', 0.5926538705825806), ('wins', 0.5878374576
568604)]
```

In [29]:

```
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
17386
sample words  ['dogs', 'loves', 'chicken', 'pr
oduct', 'china', 'wont', 'buying', 'anymore',
'hard', 'find', 'products', 'made', 'usa', 'on
e', 'isnt', 'bad', 'good', 'take', 'chances',
'till', 'know', 'going', 'imports', 'love', 's
aw', 'pet', 'store', 'tag', 'attached', 'regar
ding', 'satisfied', 'safe', 'infestation', 'li
terally', 'everywhere', 'flying', 'around', 'k
itchen', 'bought', 'hoping', 'least', 'get', '
rid', 'weeks', 'fly', 'stuck', 'squishing', 'b
uggers', 'success', 'rate']
```

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

[4.4.1.1] Avg W2v

In [30]:

```
# 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/sentenc
e
    sent_vec = np.zeros(50) # as word vectors are of zero len
gth 50, you might need to change this to 300 if you use googl
e's w2v
    cnt_words =0; # num of words with a valid vector in the s
entence/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%|██████████| 87773/87773 [06:35<00:00, 221
.80it/s]
```

```
87773
50
```

[4.4.1.2] TFIDF weighted W2v

In [31]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(preprocessed_reviews)
# we are converting a dictionary with word as a key, and the
idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.i
df_)))
```

In [0]:

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

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/re
view is stored in this list
row=0;
for sent in tqdm(list_of_sentance): # for each review/sentenc
e
    sent_vec = np.zeros(50) # as word vectors are of zero len
gth
    weight_sum =0; # num of words with a valid vector in the
sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#              tf_idf = tf_idf_matrix[row, tfidf_feat.index(wo
rd)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole c
ourpus
            # sent.count(word) = tf valeus of word in this re
view
            tf_idf = dictionary[word]*(sent.count(word)/len(s
ent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|█████████████████████████████████████████
████████████████████████████████████| 4986/498
6 [00:20<00:00, 245.63it/s]
```

[5] Assignment 4: Apply Naive Bayes

1. **Apply Multinomial NaiveBayes 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)
2. **The hyper paramter tuning(find best Alpha)**
    - Find the best hyper parameter which will give the maximum AUC value
    - Consider a wide range of alpha values for hyperparameter tuning, start as low as 0.00001
    - 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. **Feature importance**
    - Find the top 10 features of positive class and top 10 features of negative class for both feature sets Set 1 and Set 2 using values of \`feature_log_prob_\` parameter of MultinomialNB and print their corresponding feature names
4. **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.

5. **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. Here on X-axis you will have alpha values, since they have a wide range, just to represent those alpha values on the graph, apply log function on those alpha values.
  - 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.

6. **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

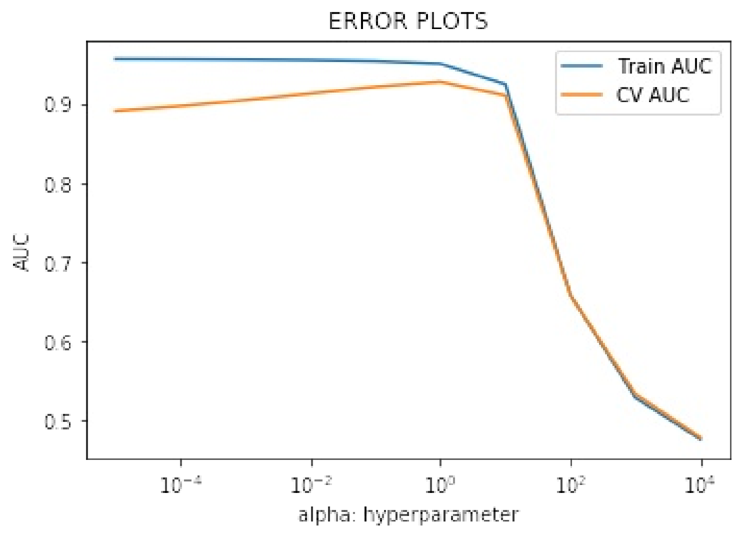
1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply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

Applying Multinomial Naive Bayes

[5.1] Applying Naive Bayes on BOW, SET 1

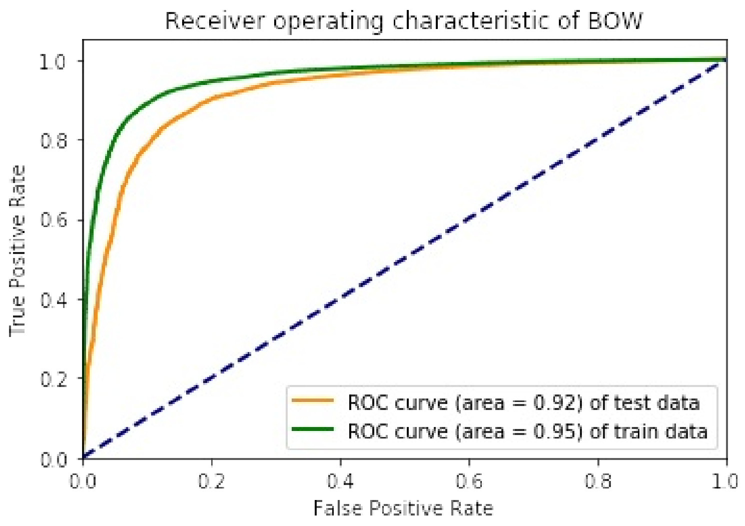
In [32]:

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

train_auc = []
cv_auc = []
alpha_p = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.
0, 1000.0, 10000.0]
for i in alpha_p:
    clf = MultinomialNB(alpha=i)
    clf.fit(final_counts_train, y_train)
    y_train_pred =  clf.predict_proba(final_counts_train)[:,1
]
    y_cv_pred =  clf.predict_proba(final_counts_cv)[:,1]
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.semilogx(alpha_p, train_auc, label='Train AUC')
plt.semilogx(alpha_p, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [34]:

```
from sklearn.metrics import roc_curve, auc
clf = MultinomialNB(alpha=0.1)
clf.fit(final_counts_train, y_train)

y_train_pred =  clf.predict_proba(final_counts_train)[:,1]
y_test_pred =  clf.predict_proba(final_counts_test)[:,1]
fpr_train, tpr_train, _ = roc_curve(y_train, y_train_pred)
roc_auc_train = auc(fpr_train, tpr_train)
fpr_test, tpr_test, _ = roc_curve(y_test, y_test_pred)
roc_auc_test = auc(fpr_test, tpr_test)
plt.figure()
lw = 2
plt.plot(fpr_test, tpr_test, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f) of test data'
 % roc_auc_test)
plt.plot(fpr_train, tpr_train, color='green',
         lw=lw, label='ROC curve (area = %0.2f) of train data
' % roc_auc_train)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic of BOW')
plt.legend(loc="lower right")
plt.show()
```



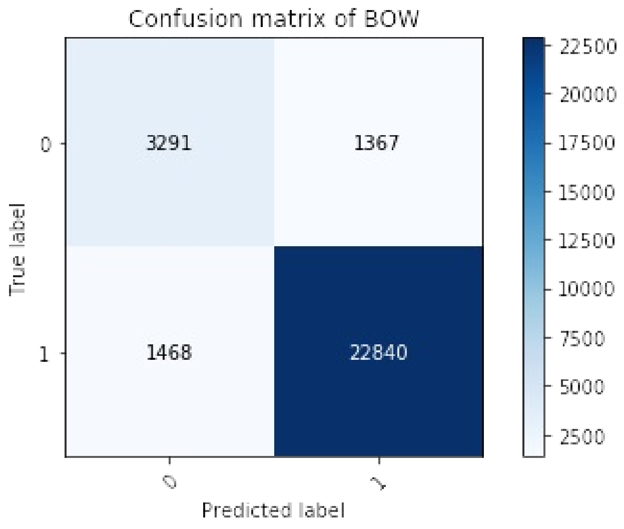
In [35]:

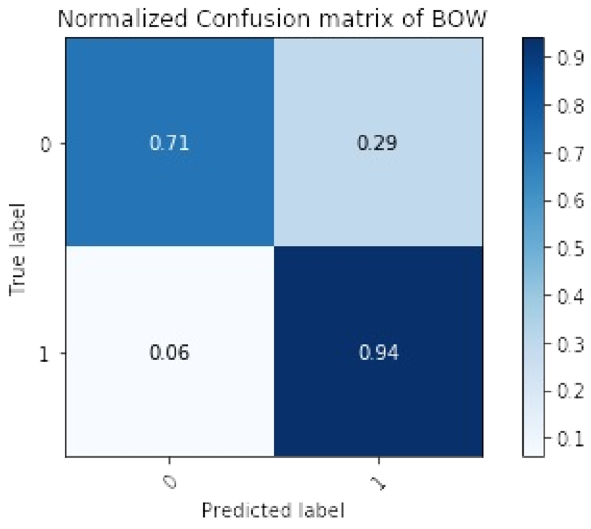
```
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels

clf = MultinomialNB(alpha=0.1)
clf.fit(final_counts_train, y_train)



y_test_pred =  clf.predict(final_counts_test)
class_names = 0,1

def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    cm = confusion_matrix(y_true, y_pred)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxi
s]
```

```
    # Only use the labels that appear in the data
    #classes = classes[unique_labels(y_true, y_pred)]
    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entr
ies
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "
black")
    fig.tight_layout()
    return ax


np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plot_confusion_matrix(y_test, y_test_pred, classes=class_name
s, normalize=False,
                      title='Confusion matrix of BOW')


plot_confusion_matrix(y_test, y_test_pred, classes=class_name
s, normalize=True,
                      title='Normalized Confusion matrix of B
OW')
```

Out[35]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x20
67c814908>
```





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

In [36]:

```
# Please write all the code with proper documentation
import numpy as np
clf = MultinomialNB(alpha=0.1)
clf.fit(final_counts_train, y_train)

positive_class_prob_sorted = clf.feature_log_prob_[1, :].args
ort()[::-1]
print("The log probabilities of top 10 features of positive c
lass", sorted(clf.feature_log_prob_[1, :], reverse=True)[0:10
])
print("                             ")
print("The top 10 features of positive class",np.take(count_v
ect.get_feature_names(),positive_class_prob_sorted[0:10]))
```

```
The log probabilities of top 10 features of po
sitive class [-3.6657699192897795, -4.48463483
1770842, -4.621432053884652, -4.68415891256450
2, -4.8299662808686765, -4.919466944281439, -4
.974899637542597, -4.998627419499165, -5.00703
5922009258, -5.007905530491808]

The top 10 features of positive class ['not' '
like' 'good' 'great' 'one' 'taste' 'coffee' 'l
ove' 'flavor'
 'would']
```

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

In [37]:

```python
# Please write all the code with proper documentation
import numpy as np
clf = MultinomialNB(alpha=0.1)
clf.fit(final_counts_train, y_train)

negative_class_prob_sorted = clf.feature_log_prob_[0, :].args
ort()[::-1]

print("The log probabilities of top 10 features of negative c
lass", sorted(clf.feature_log_prob_[0, :], reverse=True)[0:10
])
print("                             ")
print("The top 10 features of negative class",np.take(count_v
ect.get_feature_names(), negative_class_prob_sorted[0:10]))
```

```
The log probabilities of top 10 features of ne
gative class [-3.231232178287584, -4.329542685
49976, -4.58757981454618, -4.604566301776175,
-4.665211872752491, -4.870015278859359, -4.993
94239970529, -5.068088430287161, -5.0920545884
38736, -5.1481755412909544]

The top 10 features of negative class ['not' '
like' 'taste' 'would' 'product' 'one' 'coffee'
 'good' 'no'
 'flavor']
```

Performing feature engineering (adding length of review) on BOW

In [101]:

```
from sklearn.base import BaseEstimator, TransformerMixin

class ItemSelector(BaseEstimator, TransformerMixin):
    """For data grouped by feature, select subset of data at
a provided key.

    The data is expected to be stored in a 2D data structure,
 where the first
    index is over features and the second is over samples.  i
.e.

    >> len(data[key]) == n_samples

    Please note that this is the opposite convention to sciki
t-learn feature
    matrixes (where the first index corresponds to sample).

    ItemSelector only requires that the collection implement
getitem
    (data[key]).  Examples include: a dict of lists, 2D numpy
 array, Pandas
    DataFrame, numpy record array, etc.

    >> data = {'a': [1, 5, 2, 5, 2, 8],
               'b': [9, 4, 1, 4, 1, 3]}
    >> ds = ItemSelector(key='a')
    >> data['a'] == ds.transform(data)
    ItemSelector is not designed to handle data grouped by sa
mple.  (e.g. a
    list of dicts).  If your data is structured this way, con
sider a
    transformer along the lines of `sklearn.feature_extractio
n.DictVectorizer`.

    Parameters
    ----------
    key : hashable, required
        The key corresponding to the desired value in a mappa
ble.
    """
    def __init__(self, key):
        self.key = key

    def fit(self, x, y=None):
        return self

    def transform(self, data_dict):
        return data_dict[self.key]

class SubjectBodyExtractor(BaseEstimator, TransformerMixin):
    from tqdm import tqdm
    def fit(self, x, y=None):
        return self

    def transform(self, data):
        features = np.recarray(shape=(len(data),),
                               dtype=[('Summary1', object), (
'Text1', object)])
        i=0
        for sentance1 in tqdm(data['Text'].values):
            sentance1 = re.sub(r"http\S+", "", sentance1)
            sentance1 = BeautifulSoup(sentance1, 'lxml').get_
text()
            sentance1 = decontracted(sentance1)
            sentance1 = re.sub("\S*\d\S*", "", sentance1).str
ip()
            sentance1 = re.sub('[^A-Za-z]+', ' ', sentance1)
    # https://gist.github.com/sebleier/554280
            sentance1 = ' '.join(e.lower() for e in sentance1
.split() if e.lower() not in stopwords)
            features['Text1'][i] = sentance1.strip()
            i=i+1
        i=0
        for sentance11 in tqdm(data['Summary'].values):
            sentance11 = re.sub(r"http\S+", "", sentance11)
            sentance11 = BeautifulSoup(sentance11, 'lxml').ge
t_text()
            sentance11 = decontracted(sentance11)
            sentance11 = re.sub("\S*\d\S*", "", sentance11).s
trip()
            sentance11 = re.sub('[^A-Za-z]+', ' ', sentance11
)
    # https://gist.github.com/sebleier/554280
            sentance11 = ' '.join(e.lower() for e in sentance
11.split() if e.lower() not in stopwords)
            features['Summary1'][i] = sentance11.strip()
            i=i+1

        return features
```

In [102]:

```
#print(X_train1.shape)
X_train1 = timesorted_data[:28965]
y_train1 = X_train1['Score'].values
X_cv1 = timesorted_data[28965:57930]
X_test1 = timesorted_data[57930:]
y_test1 = X_test1['Score'].values
print(y_train1.shape)
```

```
#print(y_train1)
```

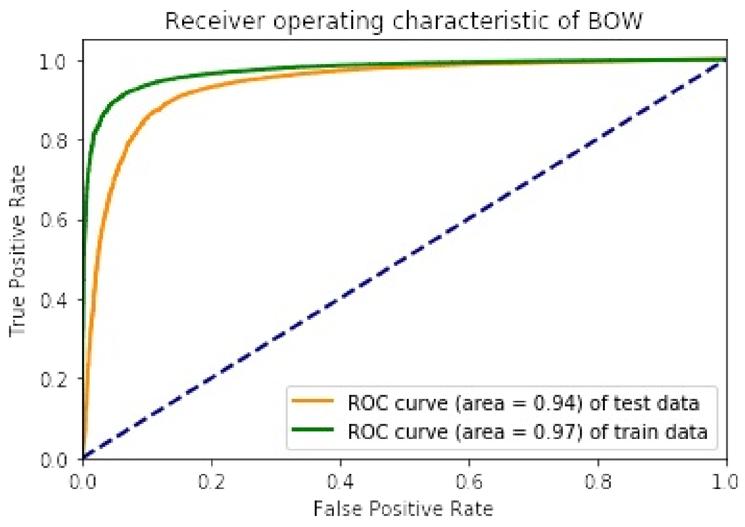
```
(28965,)
```

In [103]:

```
from sklearn.metrics import roc_curve, auc
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.preprocessing import FunctionTransformer
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

#adding a function to find the length of the review text
def get_text_length(x):
    return np.array([len(t) for t in x]).reshape(-1, 1)

# Featureunion adding the length of the text to the BOW vecto
rs. Hyperparameter considered alpha=0.1
union = Pipeline([
    ('subjectbody', SubjectBodyExtractor()),
    ('features', FeatureUnion([
        ('summarytext', Pipeline([
            ('selector', ItemSelector(key='Summary1')),
            ('vectorizer', CountVectorizer(min_df=10)),
        ])),
        ('text', Pipeline([
            ('selector', ItemSelector(key='Text1')),
            ('vectorizer', CountVectorizer(min_df=10)),
        ])),
        ('length', Pipeline([
            ('selector',ItemSelector(key='Text1')),
            ('count', FunctionTransformer(get_text_length, va
lidate=False)),
        ]))
    ])),
    ('clf', (MultinomialNB(alpha=0.1)))])
```

```
#fitting only to train data to avoid data leakage
features_train = union.fit(X_train1,y_train1)
y_train_pred =  union.predict_proba(X_train1)[:,1]
y_test_pred =  union.predict_proba(X_test1)[:,1]

fpr_train, tpr_train, _ = roc_curve(y_train1, y_train_pred)
roc_auc_train = auc(fpr_train, tpr_train)
fpr_test, tpr_test, _ = roc_curve(y_test1, y_test_pred)
roc_auc_test = auc(fpr_test, tpr_test)
plt.figure()
lw = 2
plt.plot(fpr_test, tpr_test, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f) of test data'
 % roc_auc_test)
plt.plot(fpr_train, tpr_train, color='green',
         lw=lw, label='ROC curve (area = %0.2f) of train data
' % roc_auc_train)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic of BOW')
plt.legend(loc="lower right")
plt.show()

y_test_prediction =  union.predict(X_test1)
class_names = 0,1

# Plot non-normalized confusion matrix
plot_confusion_matrix(y_test1, y_test_prediction, classes=cla
ss_names, normalize=False,
                      title='Confusion matrix of BOW')
plot_confusion_matrix(y_test1, y_test_prediction, classes=cla
ss_names, normalize=True,
                      title='Normalized Confusion matrix of B
OW')
```

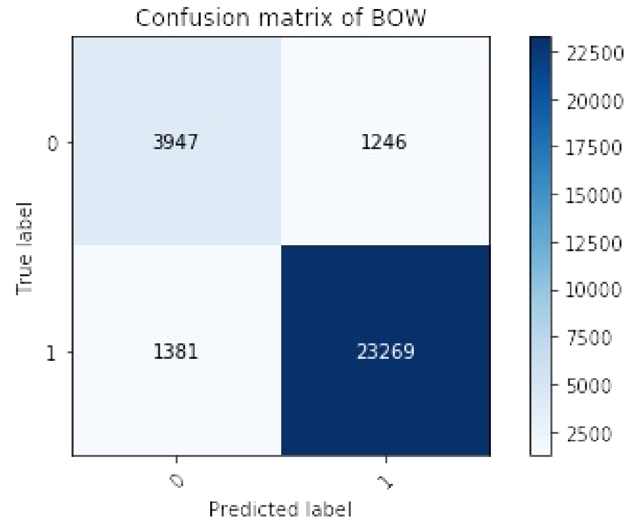
```
100%|██████████| 28965/28965 [00:21<00:00, 133
5.04it/s]
  6%|▌         | 1749/28965 [00:00<00:13, 2077
.36it/s]C:\Users\nairh\Anaconda3\lib\site-pack
ages\bs4\__init__.py:272: UserWarning: "b'...'
" looks like a filename, not markup. You shoul
d probably open this file and pass the filehan
dle into Beautiful Soup.
  ' Beautiful Soup.' % markup)
100%|██████████| 28965/28965 [00:14<00:00, 198
7.13it/s]
100%|██████████| 28965/28965 [00:20<00:00, 142
4.43it/s]
  6%|▌         | 1759/28965 [00:00<00:13, 2044
.63it/s]C:\Users\nairh\Anaconda3\lib\site-pack
ages\bs4\__init__.py:272: UserWarning: "b'...'
" looks like a filename, not markup. You shoul
d probably open this file and pass the filehan
dle into Beautiful Soup.
  ' Beautiful Soup.' % markup)
100%|██████████| 28965/28965 [00:13<00:00, 214
7.37it/s]
100%|██████████| 29843/29843 [00:21<00:00, 135
8.01it/s]
 49%|████▉     | 14612/29843 [00:07<00:06, 218
4.60it/s]C:\Users\nairh\Anaconda3\lib\site-pac
kages\bs4\__init__.py:272: UserWarning: "b'...
'" looks like a filename, not markup. You shou
ld probably open this file and pass the fileha
ndle into Beautiful Soup.
  ' Beautiful Soup.' % markup)
100%|██████████| 29843/29843 [00:15<00:00, 194
8.33it/s]
```

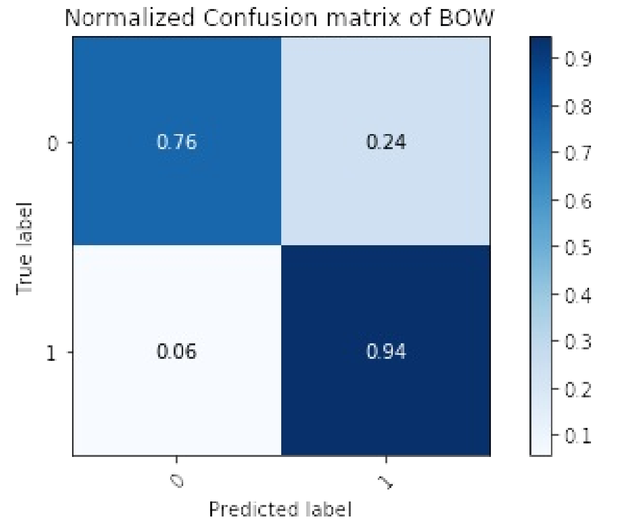


```
100%|██████████| 29843/29843 [00:22<00:00, 134
3.41it/s]
 49%|████▉     | 14670/29843 [00:07<00:08, 168
7.04it/s]C:\Users\nairh\Anaconda3\lib\site-pac
kages\bs4\__init__.py:272: UserWarning: "b'...
'" looks like a filename, not markup. You shou
ld probably open this file and pass the fileha
ndle into Beautiful Soup.
  ' Beautiful Soup.' % markup)
100%|██████████| 29843/29843 [00:16<00:00, 176
7.93it/s]
```

Out[103]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x20
602c18748>
```



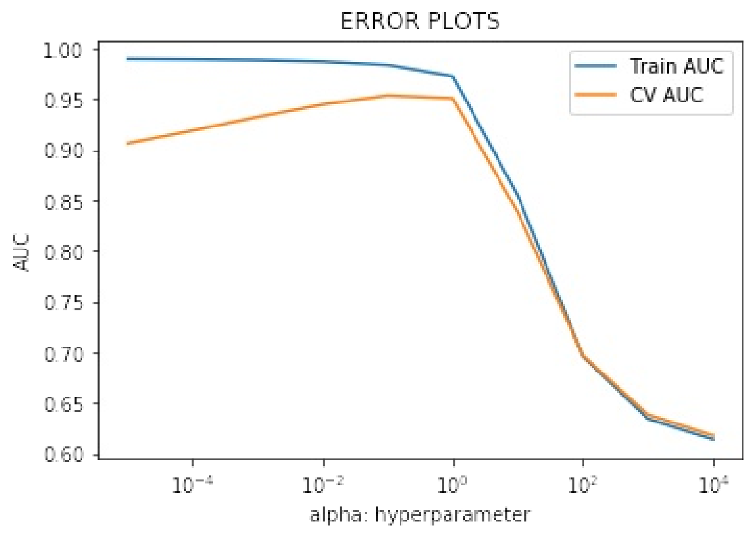


Feature engineering has been performed by incorporating the length of the review text and the review summary in addition to the usual review texts. Here, data has been split after sorting the data based on time. Inclusion of these features has improved the AUC from 0.92 to 0.94. This improvement in result can also be observed in the increase of the diagonal terms in confusion matrix as the TN increases from 0.71 to 0.76.

[5.2] Applying Naive Bayes on TFIDF, SET 2

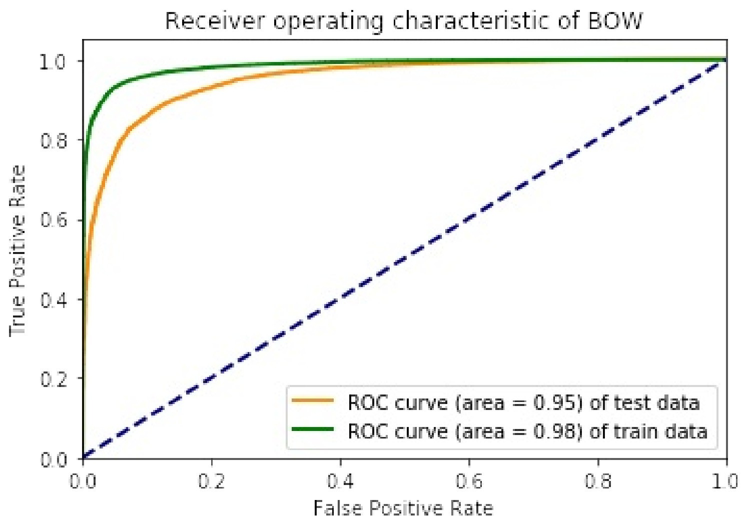
In [38]:

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

train_auc = []
cv_auc = []
alpha_p = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.
0, 1000.0, 10000.0]
for i in alpha_p:
    clf = MultinomialNB(alpha=i)
    clf.fit(final_tf_idf_train, y_train)
    y_train_pred =  clf.predict_proba(final_tf_idf_train)[:,1
]
    y_cv_pred =  clf.predict_proba(final_tf_idf_cv)[:,1]
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.semilogx(alpha_p, train_auc, label='Train AUC')
plt.semilogx(alpha_p, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [39]:

```
from sklearn.metrics import roc_curve, auc
clf = MultinomialNB(alpha=0.1)
clf.fit(final_tf_idf_train, y_train)

y_train_pred =  clf.predict_proba(final_tf_idf_train)[:,1]
y_test_pred =  clf.predict_proba(final_tf_idf_test)[:,1]
fpr_train, tpr_train, _ = roc_curve(y_train, y_train_pred)
roc_auc_train = auc(fpr_train, tpr_train)
fpr_test, tpr_test, _ = roc_curve(y_test, y_test_pred)
roc_auc_test = auc(fpr_test, tpr_test)
plt.figure()
lw = 2
plt.plot(fpr_test, tpr_test, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f) of test data'
 % roc_auc_test)
plt.plot(fpr_train, tpr_train, color='green',
         lw=lw, label='ROC curve (area = %0.2f) of train data
' % roc_auc_train)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic of BOW')
plt.legend(loc="lower right")
plt.show()
```



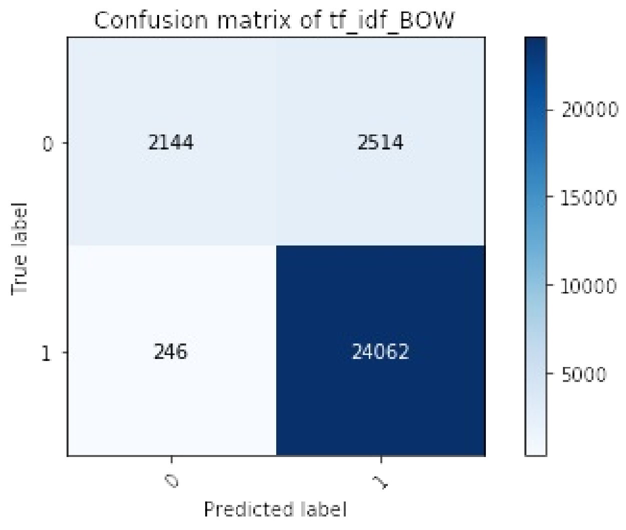
In [40]:

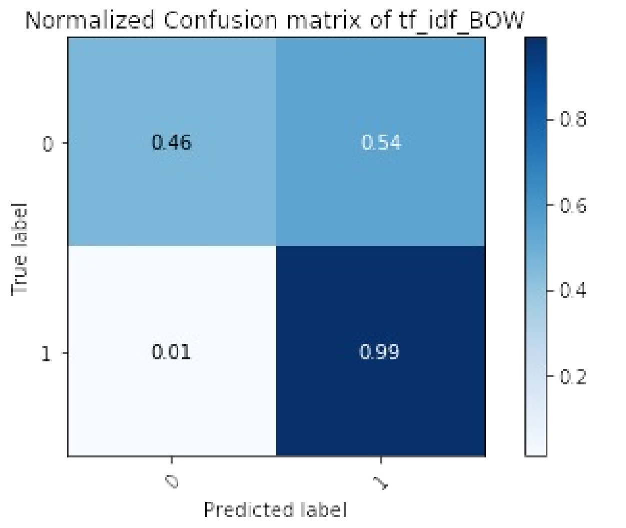
```
clf = MultinomialNB(alpha=0.1)
clf.fit(final_tf_idf_train, y_train)



y_test_pred =  clf.predict(final_tf_idf_test)
class_names = 0,1

# Plot non-normalized confusion matrix
plot_confusion_matrix(y_test, y_test_pred, classes=class_name
s, normalize=False,
                      title='Confusion matrix of tf_idf_BOW')
plot_confusion_matrix(y_test, y_test_pred, classes=class_name
s, normalize=True,
                      title='Normalized Confusion matrix of t
f_idf_BOW')
```

Out[40]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x20
```

```
6005735f8>
```





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

In [41]:

```python
# Please write all the code with proper documentation
import numpy as np
clf = MultinomialNB(alpha=0.1)
clf.fit(final_tf_idf_train, y_train)
```

```
positive_class_prob_sorted = sorted(clf.feature_log_prob_[1,
:], reverse=True)

print("The log probabilities of top 10 features of positive c
lass",positive_class_prob_sorted[0:10])
```

```
The log probabilities of top 10 features of po
sitive class [-5.216932987431211, -5.558904724
233968, -5.635538469939925, -5.685468623843244
, -5.742868367815882, -5.786356067421885, -5.8
17105572467525, -5.91497373459664, -5.92830528
42277, -5.930149922811605]
```

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

In [42]:

```
# Please write all the code with proper documentation
import numpy as np
clf = MultinomialNB(alpha=0.1)
clf.fit(final_tf_idf_train, y_train)

negative_class_prob_sorted = sorted(clf.feature_log_prob_[0,
:], reverse=True)

print( "The log probabilities of top 10 features of negative
class",negative_class_prob_sorted[:10])
```

```
The log probabilities of top 10 features of ne
gative class [-4.70841414194601, -5.4894960930
3242, -5.609663505693307, -5.625675153849732,
-5.654216648135872, -5.79220658441217, -5.9753
67081149988, -6.048529356022203, -6.0886905628
8089, -6.170275329601554]
```

[6] Conclusions

In [47]:

```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "Model" ,"Hyperparameter", "AU
C"]

x.add_row(["BOW", "MultinomialNB", 0.1, 0.92])
x.add_row(["tfidf BOW", "MultinomialNB", 0.1, 0.95])



print(x)
```

```
+------------+---------------+----------------
+------+
| Vectorizer |     Model     | Hyperparameter
| AUC  |
+------------+---------------+----------------
+------+
|    BOW     | MultinomialNB |      0.1
| 0.92 |
| tfidf BOW  | MultinomialNB |      0.1
| 0.95 |
+------------+---------------+----------------
+------+
```