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

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

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

from tqdm import tqdm
import os
```

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 50
0000 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 Sco
re != 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 50000""", con)

# Give reviews with Score>3 a positive rating(1), and reviews with a sc
ore<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 (50000, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomin
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	

In [3]:
```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [4]:
```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[4]:

	UserId	ProductId	ProfileName	Time	Score	Text	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	HelpfulnessDenon

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenom
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	
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 [8]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=Tr
ue, inplace=False, kind='quicksort', na_position='last')
```

In [9]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time"
,"Text"}, keep='first', inplace=False)
final.shape
```

Out[9]:
```
(46072, 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]:
```
92.144
```

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	HelpfulnessDenom
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	

In [12]:

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

In [13]:

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

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

```
(46071, 10)
```

Out[13]:

```
1    38479
0     7592
Name: Score, dtype: int64
```

[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)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
```

ause its a good product but I wont take any chances till they know what
is going on with the china imports.
==========
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp
or cobbler. home made is better, but a heck of a lot more work. this is
great to have on hand for last minute dessert needs where you really wa
nt to impress wih your creativity in cooking! recommended.
==========
Great flavor, low in calories, high in nutrients, high in protein! Usua
lly protein powders are high priced and high in calories, this one is a
great bargain and tastes great, I highly recommend for the lady gym rat
s, probably not "macho" enough for guys since it is soy based...
==========
For those of you wanting a high-quality, yet affordable green tea, you
should definitely give this one a try. Let me first start by saying tha
t everyone is looking for something different for their ideal tea, and
I will attempt to briefly highlight what makes this tea attractive to a
wide range of tea drinkers (whether you are a beginner or long-time tea
enthusiast). I have gone through over 12 boxes of this tea myself, and
highly recommend it for the following reasons:

-Quality: Fi
rst, this tea offers a smooth quality without any harsh or bitter after
tones, which often turns people off from many green teas. I've found m
y ideal brewing time to be between 3-5 minutes, giving you a light but
flavorful cup of tea. However, if you get distracted or forget about y
our tea and leave it brewing for 20+ minutes like I sometimes do, the q
uality of this tea is such that you still get a smooth but deeper flavo
r without the bad after taste. The leaves themselves are whole leaves
(not powdered stems, branches, etc commonly found in other brands), and
the high-quality nylon bags also include chunks of tropical fruit and o
ther discernible ingredients. This isn't your standard cheap paper bag
with a mix of unknown ingredients that have been ground down to a fine
powder, leaving you to wonder what it is you are actually drinking.

-Taste: This tea offers notes of real pineapple and other hint
s of tropical fruits, yet isn't sweet or artificially flavored. You ha
ve the foundation of a high-quality young hyson green tea for those tru
e "tea flavor" lovers, yet the subtle hints of fruit make this a truly
unique tea that I believe most will enjoy. If you want it sweet, you c

an add sugar, splenda, etc but this really is not necessary as this tea
offers an inherent warmth of flavor through it's ingredients.
<br

```
/>-Price:  This tea offers an excellent product at an exceptional price
(especially when purchased at the prices Amazon offers).  Compared to o
ther brands which I believe to be of similar quality (Mighty Leaf, Rish
i, Two Leaves, etc.), Revolution offers a superior product at an outsta
nding price.  I have been purchasing this through Amazon for less per b
ox than I would be paying at my local grocery store for Lipton, etc.<br
/><br />Overall, this is a wonderful tea that is comparable, and even b
etter than, other teas that are priced much higher.  It offers a well-b
alanced cup of green tea that I believe many will enjoy.  In terms of t
aste, quality, and price, I would argue you won't find a better combina
tion that that offered by Revolution's Tropical Green Tea.
==================================================
```

In [15]:

```python
# remove urls from text python: https://stackoverflow.com/a/40823105/40
84039
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 from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
```

In [16]:

```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 from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp
or cobbler. home made is better, but a heck of a lot more work. this is
great to have on hand for last minute dessert needs where you really wa
nt to impress wih your creativity in cooking! recommended.
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usua
lly protein powders are high priced and high in calories, this one is a
great bargain and tastes great, I highly recommend for the lady gym rat
s, probably not "macho" enough for guys since it is soy based...
==================================================
For those of you wanting a high-quality, yet affordable green tea, you
should definitely give this one a try. Let me first start by saying tha
t everyone is looking for something different for their ideal tea, and
I will attempt to briefly highlight what makes this tea attractive to a
wide range of tea drinkers (whether you are a beginner or long-time tea
enthusiast).  I have gone through over 12 boxes of this tea myself, and
highly recommend it for the following reasons:-Quality:  First, this te
a offers a smooth quality without any harsh or bitter after tones, whic
h often turns people off from many green teas.  I've found my ideal bre
wing time to be between 3-5 minutes, giving you a light but flavorful c
up of tea.  However, if you get distracted or forget about your tea and
leave it brewing for 20+ minutes like I sometimes do, the quality of th
```

```
is tea is such that you still get a smooth but deeper flavor without th
e bad after taste.  The leaves themselves are whole leaves (not powdere
d stems, branches, etc commonly found in other brands), and the high-qu
ality nylon bags also include chunks of tropical fruit and other discer
nible ingredients.  This isn't your standard cheap paper bag with a mix
of unknown ingredients that have been ground down to a fine powder, lea
ving you to wonder what it is you are actually drinking.-Taste:  This t
ea offers notes of real pineapple and other hints of tropical fruits, y
et isn't sweet or artificially flavored.  You have the foundation of a
high-quality young hyson green tea for those true "tea flavor" lovers,
yet the subtle hints of fruit make this a truly unique tea that I belie
ve most will enjoy.  If you want it sweet, you can add sugar, splenda,
etc but this really is not necessary as this tea offers an inherent war
mth of flavor through it's ingredients.-Price:  This tea offers an exce
llent product at an exceptional price (especially when purchased at the
prices Amazon offers).  Compared to other brands which I believe to be
of similar quality (Mighty Leaf, Rishi, Two Leaves, etc.), Revolution o
ffers a superior product at an outstanding price.  I have been purchasi
ng this through Amazon for less per box than I would be paying at my lo
cal grocery store for Lipton, etc.Overall, this is a wonderful tea that
is comparable, and even better than, other teas that are priced much hi
gher.  It offers a well-balanced cup of green tea that I believe many w
ill enjoy.  In terms of taste, quality, and price, I would argue you wo
n't find a better combination that that offered by Revolution's Tropica
l Green Tea.
```

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)
```

```
Great flavor, low in calories, high in nutrients, high in protein! Usua
lly protein powders are high priced and high in calories, this one is a
great bargain and tastes great, I highly recommend for the lady gym rat
s, probably not "macho" enough for guys since it is soy based...
==================================================
```

In [19]:
```
#remove words with numbers python: https://stackoverflow.com/a/1808237
0/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
```

In [20]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
Great flavor low in calories high in nutrients high in protein Usually
protein powders are high priced and high in calories this one is a grea
t bargain and tastes great I highly recommend for the lady gym rats pro
bably not macho enough for guys since it is soy based
```

In [21]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'no
```

```
t'
# <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', 'o
urs', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselve
s', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'it
s', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'th
is', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'h
ave', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between',
'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out',
'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'h
ow', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 's
o', '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', "is
n'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())
```

```
100%|██████████| 46071/46071 [00:30<00:00, 1496.89it/s]
```

In [23]:
```
preprocessed_reviews[1500]
```

Out[23]:
```
'great flavor low calories high nutrients high protein usually protein
powders high priced high calories one great bargain tastes great highly
recommend lady gym rats probably not macho enough guys since soy based'
```

[3.2] Preprocessing Review Summary

In [24]:
```
## Similartly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

In [0]:
```
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(preprocessed_reviews)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)
```

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

```
some feature names  ['aa', 'aahhhs', 'aback', 'abandon', 'abates', 'abb
ott', 'abby', 'abdominal', 'abiding', 'ability']
====================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 12997)
the number of unique words  12997
```

[4.2] Bi-Grams and n-Grams.

In [0]:

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

#removing stop words like "not" should be avoided before building n-gra
ms
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation http://scikit-learn.
org/stable/modules/generated/sklearn.feature_extraction.text.CountVecto
rizer.html

# you can choose these numebrs min_df=10, max_features=5000, of your ch
oice
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_s
hape())
print("the number of unique words including both unigrams and bigrams "
, final_bigram_counts.get_shape()[1])
```

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

[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.ge
t_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', 'a
ble find', 'able get', 'absolute', 'absolutely', 'absolutely deliciou
s', '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 [0]:
```
# 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 [0]:
```
# 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 val
ues
# To use this code-snippet, download "GoogleNews-vectors-negative300.bi
n"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edi
t
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17
SRFAzZPY
# 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_trai
n_w2v = True, to train your own w2v ")
```

```
[('snack', 0.9951335191726685), ('calorie', 0.9946465492248535), ('wond
erful', 0.9946032166481018), ('excellent', 0.9944332838058472), ('espec
```

```
ially', 0.9941144585609436), ('baked', 0.9940600395202637), ('salted',
0.994047224521637), ('alternative', 0.9937226176261902), ('tasty', 0.99
36816692352295), ('healthy', 0.9936649799346924)]
==================================================
[('varieties', 0.9994194507598877), ('become', 0.9992934465408325), ('p
opcorn', 0.9992750883102417), ('de', 0.9992610216140747), ('miss', 0.99
92451071739197), ('melitta', 0.999218761920929), ('choice', 0.999210238
4567261), ('american', 0.9991837739944458), ('beef', 0.999178051948547
4), ('finish', 0.9991567134857178)]
```

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', 'st
inky', 'right', 'nearby', 'used', 'ca', 'not', 'beat', 'great', 'receiv
ed', 'shipment', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'ins
tead', 'removed', 'easily', 'daughter', 'designed', 'printed', 'use',
'car', 'windows', 'beautifully', 'shop', 'program', 'going', 'lot', 'fu
n', 'everywhere', 'like', 'tv', 'computer', 'really', 'good', 'idea',
'final', 'outstanding', 'window', 'everybody', 'asks', 'bought', 'mad
e']
```

[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, yo
u might need to change this to 300 if you use google's w2v
```

```
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    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<00:00, 1330.47it/s]
```

```
4986
50
```

[4.4.1.2] TFIDF weighted W2v

In [0]:
```
# 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 v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [0]:
```
# 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 ce
ll_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is st
ored in this list
row=0;
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
```

```
    weight_sum =0; # num of words with a valid vector in the sentence/r
eview
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|[redacted]| 4986/4986 [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 [0]:
```
# Please write all the code with proper documentation
```

In [25]:
```
df = pd.DataFrame({'text':preprocessed_reviews,'class':final['Score']})
```

In [27]:
```
df.head(2)
```

Out[27]:

	text	class
22620	dogs loves chicken product china wont buying a...	0
22621	dogs love saw pet store tag attached regarding...	1

In [28]:
```
df.shape
```

Out[28]:
```
(46071, 2)
```

In [29]:
```
Y = df['class'].values
X = df['text'].values
```

In [ ]:

In [30]:

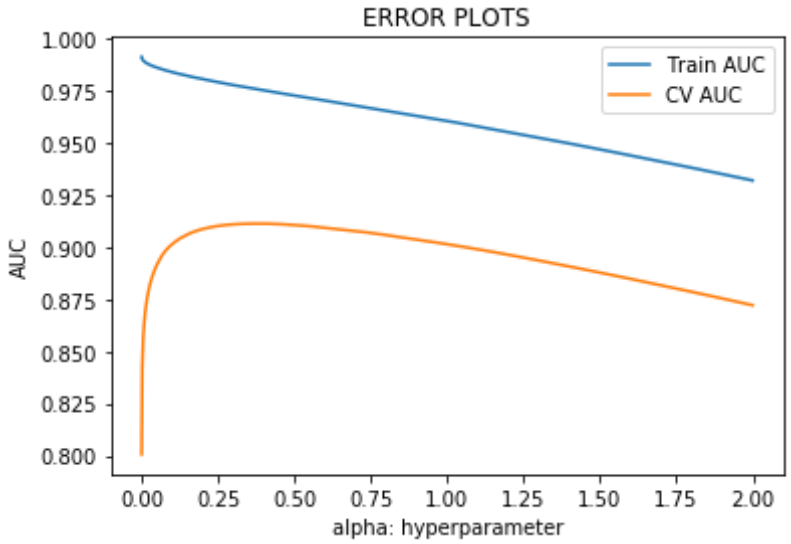
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selec
tion.train_test_split.html
from sklearn.model_selection import train_test_split

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=
0.33, shuffle=Flase): this is for time series split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3
3) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_
size=0.33) # this is random splitting


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

print("="*100)

from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
X_train_bow= vectorizer.fit_transform(X_train) # fit has to happen only
 on train data

# we use the fitted CountVectorizer to convert the text to vector
#X_train_bow = vectorizer.transform(X_train)
X_cv_bow = vectorizer.transform(X_cv)
X_test_bow = vectorizer.transform(X_test)

print("After vectorizations")
print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)
print("="*100)
```

```
(20680,) (20680,)
(10187,) (10187,)
(15204,) (15204,)
====================================================================
```

```
==============================
After vectorizations
(20680, 26954) (20680,)
(10187, 26954) (10187,)
(15204, 26954) (15204,)
=====================================================================
==============================
```

In [38]:
```
alpha_lst = np.arange(0.0001,2,0.001)
```

In [39]:
```
 from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

train_auc = []
cv_auc = []
a = np.arange(0.0001,2,0.001)
for i in a:
    mnb = MultinomialNB(alpha=i)
    mnb.fit(X_train_bow, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probab
ility estimates of the positive class
    # not the predicted outputs
    y_train_pred =  mnb.predict_proba(X_train_bow)[:,1]
    y_cv_pred =  mnb.predict_proba(X_cv_bow)[:,1]
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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



In [27]:

```
```

In [28]:

```
best_alpha = 0.35
```

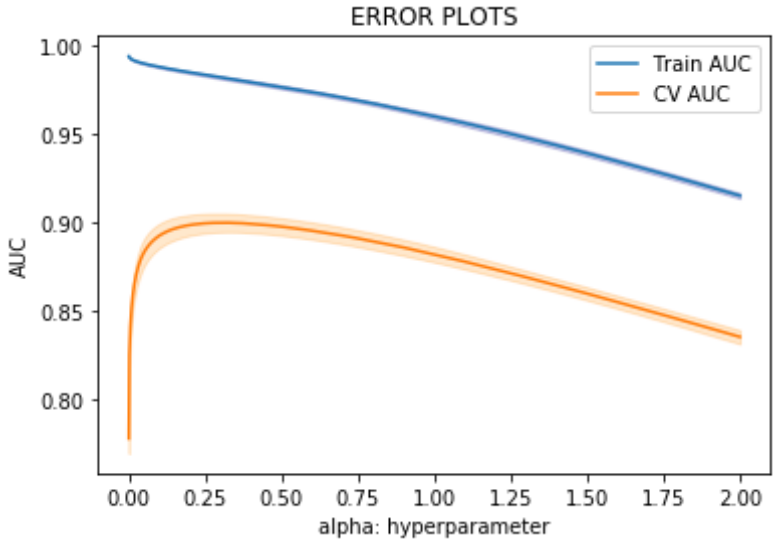
In [40]:

```
parameters = {'alpha':  np.arange(0.0001,2,0.001)}
```

In [41]:

```
#BOW
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selec
tion.GridSearchCV.html
from sklearn.model_selection import GridSearchCV

mnb = MultinomialNB(alpha=parameters)
clf = GridSearchCV(mnb, parameters, cv=3, scoring='roc_auc')
clf.fit(X_train_bow, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

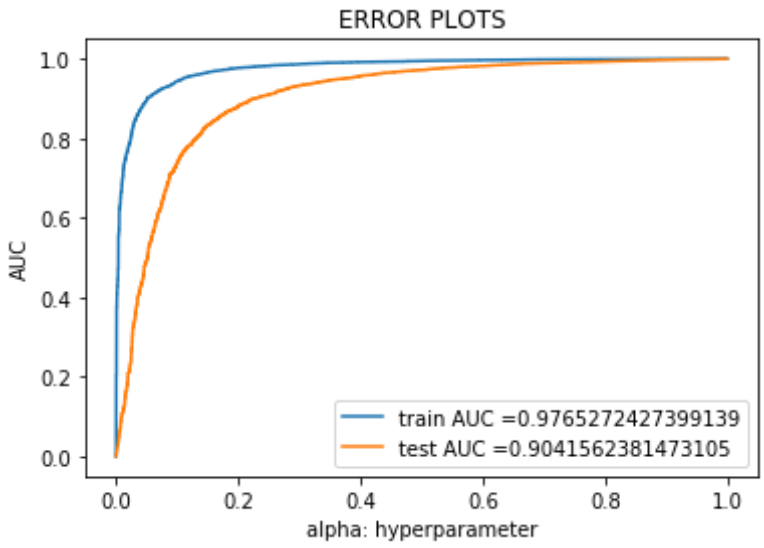
```
plt.plot(alpha_lst, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4
084039
plt.gca().fill_between(alpha_lst,train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.2,color='darkblue')

plt.plot(alpha_lst, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4
084039
plt.gca().fill_between(alpha_lst,cv_auc - cv_auc_std,cv_auc + cv_auc_st
d,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [42]:
```
best_alpha=0.35
```

In [44]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc
_curve.html#sklearn.metrics.roc_curve
```

```
from sklearn.metrics import roc_curve, auc


mnb = MultinomialNB(alpha=best_alpha)
mnb.fit(X_train_bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, mnb.predict_proba
(X_train_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, mnb.predict_proba(X_
test_bow)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_
tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, mnb.predict(X_train_bow)))
print("Test confusion matrix")
print(confusion_matrix(y_test, mnb.predict(X_test_bow)))
```




```
=====================================================================
==============================
Train confusion matrix
[[ 2742   557]
 [  510 16871]]
Test confusion matrix
[[ 1636   935]
 [  644 11989]]
```

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

In [50]:
```
NB_optimal = MultinomialNB(alpha=best_alpha)

# fitting the model
NB_optimal.fit(X_train, y_train)
```

```
# predict the response
pred = NB_optimal.predict(X_test)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the NB classifier for k = %d is %f%%' % (optim
al_aplha, acc))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call l
ast)
<ipython-input-50-57ec08f98280> in <module>
      2
      3 # fitting the model
----> 4 NB_optimal.fit(X_train, y_train)
      5
      6 # predict the response

~\Anaconda3\lib\site-packages\sklearn\naive_bayes.py in fit(self, X, y,
 sample_weight)
    583         self : object
    584         """
--> 585         X, y = check_X_y(X, y, 'csr')
    586         _, n_features = X.shape
    587

~\Anaconda3\lib\site-packages\sklearn\utils\validation.py in check_X_y
(X, y, accept_sparse, accept_large_sparse, dtype, order, copy, force_al
l_finite, ensure_2d, allow_nd, multi_output, ensure_min_samples, ensure
_min_features, y_numeric, warn_on_dtype, estimator)
    754                     ensure_min_features=ensure_min_features,
    755                     warn_on_dtype=warn_on_dtype,
--> 756                     estimator=estimator)
    757     if multi_output:
    758         y = check_array(y, 'csr', force_all_finite=True, ensure
_2d=False,

~\Anaconda3\lib\site-packages\sklearn\utils\validation.py in check_arra
y(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_
```

```
all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_feature
s, warn_on_dtype, estimator)
    525             try:
    526                 warnings.simplefilter('error', ComplexWarning)
--> 527                 array = np.asarray(array, dtype=dtype, order=or
der)
    528             except ComplexWarning:
    529                 raise ValueError("Complex data not supported\n"

~\Anaconda3\lib\site-packages\numpy\core\numeric.py in asarray(a, dtyp
e, order)
    499
    500     """
--> 501     return array(a, dtype, copy=False, order=order)
    502
    503

ValueError: could not convert string to float: 'admit first taste not i
mpressed nuts pretty sure get taste lick almond dip chocolate coco mix
 plus not big fan dark chocolate not help could not stop finally wife t
ake away insisting would using future culinary masterpiece conclusion a
lmonds good think even washed chocolate things would addictive probably
 not bad thing since far know nuts good ya'
```

In [ ]:

In [ ]:

In [ ]:

In [46]:

```
neg_class_prob_sorted = mnb.coef_[0, :].argsort()
pos_class_prob_sorted = mnb.coef_[0, :].argsort()[::-1]

print('Important Features')
print(np.take(vectorizer.get_feature_names(), neg_class_prob_sorted[:20
]))
print("separator")
```

```
print(np.take(vectorizer.get_feature_names(), pos_class_prob_sorted[::2
0]))
```

```
Important Features
['fullers' 'leeks' 'legion' 'legitimate' 'legitimately' 'bushy' 'shit'
 'shirts' 'shirt' 'lending' 'dukans' 'leomonade' 'lectures' 'buts'
 'lethal' 'shihtzu' 'lethargy' 'butterorganic' 'ducale' 'lettieri']
separator
['not' 'time' 'bag' ... 'leblanc' 'bunches' 'buts']
```

In [47]:
```
bowData = pd.DataFrame({"features":vectorizer.get_feature_names()})
```

In [48]:
```
bowData.head(10)
```

Out[48]:

	features
0	aa
1	aaa
2	aaaa
3	aaaaaaaaaaaa
4	aaaaaaaaaaaaaaa
5	aaaaaaahhhhhh
6	aaaaaawwwwwwwwww
7	aaah
8	aadp
9	aafco

In [ ]:
```

```

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

```

```

In [122]:
```
# Please write all the code with proper documentation

if 1 in final['Score']:
    pos_class_prob_sorted = abs(mnb.coef_[0, :].argsort())
    print(np.take(vectorizer.get_feature_names(), pos_class_prob_sorted
[:10]))
```

```
['leach' 'burrows' 'burroughs' 'burrough' 'simpleness' 'livein' 'burpe
d'
 'simultaneous' 'sinewy' 'sinker']
```

In [96]:
```
df_score = pd.DataFrame({'scores':final['Score']})
```

In [132]:
```
if 0 in final['Score']:
    pos_class_prob_sorted = abs(mnb.coef_[0:].argsort())
    print(np.take(vectorizer.get_feature_names(), pos_class_prob_sorted
[:-10]))
```

```
[]
```

In [113]:
```
pos_class_prob_sorted = abs(mnb.coef_[0, :].argsort())
print(np.take(vectorizer.get_feature_names(), pos_class_prob_sorted[:10
]))
```

```
['leach' 'burrows' 'burroughs' 'burrough' 'simpleness' 'livein' 'burpe
d'
 'simultaneous' 'sinewy' 'sinker']
```

[5.2] Applying Naive Bayes on TFIDF, SET 2

In [206]:
```
# Please write all the code with proper documentation

# https://scikit-learn.org/stable/modules/generated/sklearn.model_selec
tion.train_test_split.html
from sklearn.model_selection import train_test_split
```

```
# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=
0.33, shuffle=Flase): this is for time series split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3
3) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_
size=0.33) # this is random splitting


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

print("="*100)


tf_idf_vect = TfidfVectorizer()
tf_idf_vect.fit(X_train)
print("some sample features(unique words in the corpus)",tf_idf_vect.ge
t_feature_names()[0:10])
print('='*50)

final_tf_idf = tf_idf_vect.transform(X_train)
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])


# we use the fitted CountVectorizer to convert the text to vector

X_cv_tfidf = tf_idf_vect.transform(X_cv)
X_test_tfidf = tf_idf_vect.transform(X_test)
```

```
(39400,) (39400,)
(19407,) (19407,)
(28966,) (28966,)
====================================================================
==============================
some sample features(unique words in the corpus) ['aa', 'aaaa', 'aaaaaa
```

```
aaaaaa', 'aaaaaaahhhhhh', 'aaaaaawwwwwwwwww', 'aaaand', 'aaah', 'aaah
s', 'aafco', 'aah']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (39400, 37539)
the number of unique words including both unigrams and bigrams  37539
```

In [208]:

```
#BOW
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selec
tion.GridSearchCV.html
from sklearn.model_selection import GridSearchCV

mnb = MultinomialNB(alpha=parameters)
clf = GridSearchCV(mnb, parameters, cv=3, scoring='roc_auc')
clf.fit(final_tf_idf, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(parameters, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4
084039
plt.gca().fill_between(parameters,train_auc - train_auc_std,train_auc +
 train_auc_std,alpha=0.2,color='darkblue')

plt.plot(alpha_lst, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4
084039
plt.gca().fill_between(parameters,cv_auc - cv_auc_std,cv_auc + cv_auc_s
td,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

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

```
ValueError                                Traceback (most recent call l
ast)
<ipython-input-208-c45191b8bd74> in <module>
      5 mnb = MultinomialNB(alpha=parameters)
      6 clf = GridSearchCV(mnb, parameters, cv=3, scoring='roc_auc')
----> 7 clf.fit(final_tf_idf, y_train)
      8
      9 train_auc= clf.cv_results_['mean_train_score']

~\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py in fit
(self, X, y, groups, **fit_params)
    720                 return results_container[0]
    721
--> 722             self._run_search(evaluate_candidates)
    723
    724         results = results_container[0]

~\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py in _ru
n_search(self, evaluate_candidates)
   1189     def _run_search(self, evaluate_candidates):
   1190         """Search all candidates in param_grid"""
-> 1191         evaluate_candidates(ParameterGrid(self.param_grid))
   1192
   1193


~\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py in eva
luate_candidates(candidate_params)
    709                                 for parameters, (train, test)
    710                                 in product(candidate_params,
--> 711                                            cv.split(X, y, group
s)))
    712
    713                 all_candidate_params.extend(candidate_params)

~\Anaconda3\lib\site-packages\sklearn\externals\joblib\parallel.py in _
__call__(self, iterable)
    915             # remaining jobs.
    916             self._iterating = False
--> 917             if self.dispatch_one_batch(iterator):
    918                 self._iterating = self._original_iterator is no
```

```
    918                 self._iterating = self._original_iterator is no
t None
    919

~\Anaconda3\lib\site-packages\sklearn\externals\joblib\parallel.py in d
ispatch_one_batch(self, iterator)
    757                 return False
    758             else:
--> 759                 self._dispatch(tasks)
    760                 return True
    761

~\Anaconda3\lib\site-packages\sklearn\externals\joblib\parallel.py in _
dispatch(self, batch)
    714         with self._lock:
    715             job_idx = len(self._jobs)
--> 716             job = self._backend.apply_async(batch, callback=cb)
    717             # A job can complete so quickly than its callback i
s
    718             # called before we get here, causing self._jobs to

~\Anaconda3\lib\site-packages\sklearn\externals\joblib\_parallel_backen
ds.py in apply_async(self, func, callback)
    180     def apply_async(self, func, callback=None):
    181         """Schedule a func to be run"""
--> 182         result = ImmediateResult(func)
    183         if callback:
    184             callback(result)

~\Anaconda3\lib\site-packages\sklearn\externals\joblib\_parallel_backen
ds.py in __init__(self, batch)
    547         # Don't delay the application, to avoid keeping the inp
ut
    548         # arguments in memory
--> 549         self.results = batch()
    550
    551     def get(self):

~\Anaconda3\lib\site-packages\sklearn\externals\joblib\parallel.py in _
__call__(self)
```

```
__call__(self)

    223         with parallel_backend(self._backend, n_jobs=self._n_job
s):
    224             return [func(*args, **kwargs)
--> 225                     for func, args, kwargs in self.items]
    226
    227     def __len__(self):

~\Anaconda3\lib\site-packages\sklearn\externals\joblib\parallel.py in <
listcomp>(.0)
    223         with parallel_backend(self._backend, n_jobs=self._n_job
s):
    224             return [func(*args, **kwargs)
--> 225                     for func, args, kwargs in self.items]
    226
    227     def __len__(self):

~\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py in
 _fit_and_score(estimator, X, y, scorer, train, test, verbose, paramete
rs, fit_params, return_train_score, return_parameters, return_n_test_sa
mples, return_times, return_estimator, error_score)
    512     train_scores = {}
    513     if parameters is not None:
--> 514         estimator.set_params(**parameters)
    515
    516     start_time = time.time()

~\Anaconda3\lib\site-packages\sklearn\base.py in set_params(self, **par
ams)
    211                                 'Check the list of available p
arameters '
    212                                 'with `estimator.get_params().
keys()`.' %
--> 213                                 (key, self))
    214
    215             if delim:

ValueError: Invalid parameter alpha_lst for estimator MultinomialNB(alp
ha={'alpha_lst': [0.0001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 1
000]}
```

```
0000]},

       class_prior=None, fit_prior=True). Check the list of available p
arameters with `estimator.get_params().keys()`.
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

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

In [0]:

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

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

In [0]:

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

[6] Conclusions

In [0]:

```
# Please compare all your models using Prettytable library
```