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
import warnings
warnings.filterwarnings("ignore")
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 500000 data points
# you can change the number to any other number based on your computing power

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

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 100000""", con)

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

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

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

Out[2]:

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

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

In [3]:

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

In [4]:

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

```
(80668, 7)
```

Out[4]:

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

In [5]:

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

Out[5]:

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

In [6]:

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

Out[6]:

```
393063
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

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

In [7]:

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

Out[7]:

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

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

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

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

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

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

In [8]:

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

In [9]:

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

Out[9]:

```
(87775, 10)
```

In [10]:

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

Out[10]:

```
87.775
```

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

In [11]:

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

display.head()
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	12248928
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	12128832

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()
```

```
(87773, 10)
```

Out[13]:

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

In [14]:

```
final.sort_values("Time",inplace=True)
```

In [15]:

```
final
```

Out[15]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomi
70688	76882	B00002N8SM	A32DW342WBJ6BX	Buttersugar	0	0
1146	1245	B00002Z754	A29Z5PI9BW2PU3	Robbie	7	7
1145	1244	B00002Z754	A3B8RCEI0FXFI6	B G Chase	10	10
28086	30629	B00008RCMI	A19E94CF5O1LY7	Andrew Arnold	0	0
28087	30630	B00008RCMI	A284C7M23F0APC	A. Mendoza	0	0
61299	66610	B0000SY9U4	A3EEDHNHI4WNSH	Joanna J. Young	23	23
38740	42069	B0000EIEQU	A1YMJX4YWCE6P4	Jim Carson "http://www.jimcarson.com"	12	12
38889	42227	B0000A0BS8	A1IU7S4HCK1XK0	Joanna Daneman	5	5
38888	42226	B0000A0BS8	A23GFTVIETX7DS	Debbie Lee Wesselmann	5	5
10992	11991	B0000T15M8	A2928LJN5IISB4	chatchi	5	5
28085	30628	B00008RCMI	A3AKWA5CWSKOOH	Ilaxi S. Patel "Editor, kidsfreesouls.com & A...	0	0
97546	105988	B0000DG4EJ	AVCJ3K0HFRRUM	H. Johnson	0	0
96196	104537	B0000DG5B6	A1S3DOTCYJPE4O	hervin02 "hervin02"	0	0
62127	67497	B0000D9N7U	AOEIH82DRRMW	Patrick O'Brien	26	26

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomi
87386	95119	B0000DIYIJ	A3S4XR84R8S0TV	Brook Lindquist	0	1
39671	43130	B0000W2SZS	A2BETN6Y2DEFZ1	Catnip	11	11
48952	53177	B002UUJ590	A2IF5C0I5BH11F	Kala	17	18
24061	26313	B000121BY6	A281NPSIMI1C2R	Rebecca of Amazon "The Rebecca Review"	9	9
86598	94281	B0000CNU2Q	A1NOWEOLKMRRXM	T. Reinhardt "olivia lee"	27	27
86599	94282	B0000CNU2Q	A1IU7S4HCK1XK0	Joanna Daneman	14	14
81698	88850	B00015UELO	A1ZF35RV6WGYFG	Gloriya O. Grinsteiner	4	6
94002	102194	B0000UD67Y	A18O1KPT80HUDQ	K. Moore "collegian"	0	0
94024	102216	B0000GH6UG	A1J2NULS2YDNAQ	Matt Cromwell	8	12
94001	102193	B0000UD67Y	A2QG8VTCMUQDO2	A. J. Lamb	0	0
24220	26484	B0000TLEEW	A3M174IC0VXOS2	Gail Cooke	5	6
94494	102712	B0000D9N63	A2P8AVWJO0CVGL	Dipper Lips "DIP"	3	9
7427	8111	B0000EIE2Z	A3M174IC0VXOS2	Gail Cooke	3	3
25005	27304	B000J36EQC	A28SJYEFR84MU1	L Flores	0	0
97771	106224	B0000DJT3C	A1ETIK7N9ZWZY9	Call Me Jonah	5	7
94382	102594	B0000D9N6V	A28ECE800BV42W	"bungfritz"	5	5
...	...	...	...	...	...	...

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomi
42269	45991	B007VQQT1K	A34P4V70RNC2YV	S. Guss	0	0
76059	82772	B0049K99RW	A1Y73Y4VX3AJMZ	Rispir Chrone	0	0
25112	27424	B003WEFSAI	A37O0JPLJ8BOXP	Texaschick59	0	0
29158	31794	B0049D7HRS	A3LR9HCV3D96I3	Gypsy Healer	0	0
87843	95629	B000LKXDXU	A2J3PR6J36UTVH	Joyce	0	0
97624	106071	B007JTKEQK	A1DOMJI7GXGPNY	Jyouk	0	0
14526	15842	B007TJGZ5E	A3UOYYQS5Z47MS	David A. Levin "DaveL"	0	0
14300	15605	B000255OIG	AUINI96NMGXUI	Kkrys23	0	0
14299	15604	B000255OIG	A3SSEJ8IEM4YGW	Seagaul	0	0
82884	90215	B00866AM2G	ADTOX2JFWWA0B	Arnos Vale	0	0
82885	90216	B00866AM2G	AY839W9JQDZM2	Daniella	0	0
15069	16426	B007TJGZ54	A29BJSTYH9W3JI	Harry	0	0
43703	47562	B004M0Y8T8	A2QJS6MHTIFSRI	Georgie	0	0
13539	14784	B000S859NC	A2H7STZ2URUCOE	Christopher Whedon "the odd bead"	0	0
52220	56723	B0012XBD7I	A32NC2UF34RJQY	D. Pagliassotti	0	0
55100	59787	B002K9BG16	A30A7W9CZ77GFY	Cecelia Thomas "Lady Kinrowan"	0	0
89213	97089	B004O8KBK8	A1JPKFGGF128X1	MTNick	0	0
6548	7178	B004OQLIHK	AKHOMSUQRSA91	Pen Name	0	0

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomi
60967	66252	B007OSBGOK	A10QOESY9VJ9K	Gina	0	0
43268	47077	B001C4PKIK	A3IMXYITIO8WHN	Thomas R. Jackson	0	0
16026	17512	B0045Z6K50	A3HM6TNYB7FNDL	C. Furman	0	0
90340	98294	B0002LY6W0	A1BX08Y0GIT5RU	L. Nguyen "Always on the lookout for a good d...	0	0
76594	83330	B005ZBZLT4	AAMUNRK134Y5P	Tony Schy	0	0
78715	85601	B003ZURM80	A1O6MADFNBRX7H	Denise Lake	0	0
50708	55049	B000IHJEDE	A2DFSA2JXQKVY3	C-Rush	0	0
76593	83329	B005ZBZLT4	A308RR8J9NJOOZ	Josh	0	0
22401	24518	B0016JJEFG	AO9WE22147CRH	Arvind Rajan	0	0
56673	61474	B005YVU4A6	A2LU545SISQOJ8	Kelly	0	0
37074	40274	B005VOOT52	A2FKFQQPU498JT	cc	0	0
5259	5703	B009WSNWC4	AMP7K1O84DH1T	ESTY	0	0

87773 rows × 10 columns

[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)

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 [16]:

```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)

sent_1000 = final['Text'].values[1000]
print(sent_1000)
print("="*50)

sent_1500 = final['Text'].values[1500]
print(sent_1500)
print("="*50)

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

```
I bought a few of these after my apartment was infested with fruit flies. After only a few hours,
the trap had &quot;attracted&quot; many flies and within a few days they were practically gone. Th
is may not be a long term  solution, but if flies are driving you crazy, consider buying this. One
caution- the surface is very sticky, so try to avoid touching it.
==================================================
I have made these brownies for family and for a den of cub scouts and no one would have known they
were gluten free and everyone asked for seconds!  These brownies have a fudgy texture and have bit
s of chocolate chips in them which are delicious.  I would say the mix is very thick and a little
difficult to work with.  The cooked brownies are slightly difficult to cut into very neat edges as
the edges tend to crumble a little and I would also say that they make a slightly thinner layer of
brownies than most of the store brand gluten containing but they taste just as good, if not
better.  Highly recommended!<br /><br />(For those wondering, this mix requires 2 eggs OR 4 egg wh
ites and 7 tbs melted butter to prepare.  They do have suggestions for lactose free and low fat pr
eparations)
==================================================
This gum is my absolute favorite. By purchasing on amazon I can get the savings of large quanities
at a very good price. I highly recommend to all gum chewers. Plus as you enjoy the peppermint flav
or and freshing of breath you are whitening your teeth all at the same time.
==================================================
This is an excellent product, both tastey and priced right. It's difficult to find this product in
regular local grocery stores, so I was thrilled to find it.
==================================================
```

In [17]:

```
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

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

In [18]:

```
# 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()
```

```python
print(text)
print("="*50)

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

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

```
I bought a few of these after my apartment was infested with fruit flies. After only a few hours,
the trap had "attracted" many flies and within a few days they were practically gone. This may not
be a long term  solution, but if flies are driving you crazy, consider buying this. One  caution-
the surface is very sticky, so try to avoid touching it.
==================================================
I have made these brownies for family and for a den of cub scouts and no one would have known they
were gluten free and everyone asked for seconds!  These brownies have a fudgy texture and have bit
s of chocolate chips in them which are delicious.  I would say the mix is very thick and a little
difficult to work with.  The cooked brownies are slightly difficult to cut into very neat edges as
the edges tend to crumble a little and I would also say that they make a slightly thinner layer of
brownies than most of the store brand gluten containing but they taste just as good, if not
better.  Highly recommended!(For those wondering, this mix requires 2 eggs OR 4 egg whites and 7 t
bs melted butter to prepare.  They do have suggestions for lactose free and low fat preparations)
==================================================
This gum is my absolute favorite. By purchasing on amazon I can get the savings of large quanities
at a very good price. I highly recommend to all gum chewers. Plus as you enjoy the peppermint flav
or and freshing of breath you are whitening your teeth all at the same time.
==================================================
This is an excellent product, both tastey and priced right. It's difficult to find this product in
regular local grocery stores, so I was thrilled to find it.
```

In [19]:

```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 [20]:

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

```
This gum is my absolute favorite. By purchasing on amazon I can get the savings of large quanities
at a very good price. I highly recommend to all gum chewers. Plus as you enjoy the peppermint flav
or and freshing of breath you are whitening your teeth all at the same time.
==================================================
```

In [21]:

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

```
I bought a few of these after my apartment was infested with fruit flies. After only a few hours,
the trap had &quot;attracted&quot; many flies and within a few days they were practically gone. Th
```

the trap had "attracted" many flies and within a few days they were practically gone. It is may not be a long term solution, but if flies are driving you crazy, consider buying this. One caution- the surface is very sticky, so try to avoid touching it.

In [22]:

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

This gum is my absolute favorite By purchasing on amazon I can get the savings of large quanities at a very good price I highly recommend to all gum chewers Plus as you enjoy the peppermint flavor and freshing of breath you are whitening your teeth all at the same time

In [23]:

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

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "y
ou're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [24]:

```
# 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%|██████████| 87773/87773 [00:54<00:00, 1604.60it/s]
```

In [25]:

```
preprocessed_reviews[1500]
```

Out[25]:

```
'gum absolute favorite purchasing amazon get savings large quanities good price highly recommend g
um chewers plus enjoy peppermint flavor freshing breath whitening teeth time'
```

[3.2] Preprocessing Review Summary

In [26]:

```
## Similartly you can do preprocessing for review summary also.
from tqdm import tqdm
preprocessed_summary = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Summary'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwords)
    preprocessed_summary.append(sentance.strip())
```

```
100%|██████████| 87773/87773 [00:28<00:00, 3049.04it/s]
```

In [27]:

```
len(preprocessed_summary)
```

Out[27]:

```
87773
```

[4] Featurization

[4.1] BAG OF WORDS

In [26]:

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

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

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

[4.2] Bi-Grams and n-Grams.

In [27]:

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

#removing stop words like "not" should be avoided before building n-grams
# count_vect = CountVectorizer(ngram_range=(1,2))
```

```
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

# you can choose these numebrs min_df=10, max_features=5000, of your choice
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_bigram_counts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (87773, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[4.3] TF-IDF

In [28]:

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

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

```
some sample features(unique words in the corpus) ['aa', 'aafco', 'aback', 'abandon', 'abandoned', 'abdominal', 'ability', 'able', 'able add', 'able brew']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (87773, 51709)
the number of unique words including both unigrams and bigrams  51709
```

[4.4] Word2Vec

In [28]:

```
# 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 [29]:

```
# Using Google News Word2Vectors

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


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

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True
```

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

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=Tr
ue)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your
own w2v ")
```

```
[('fantastic', 0.8465518355369568), ('terrific', 0.8245105147361755), ('awesome',
0.8190785646438599), ('good', 0.8167353868484497), ('excellent', 0.8091518878936768), ('perfect',
0.7542491555213928), ('wonderful', 0.7512638568878174), ('amazing', 0.7020801901817322), ('nice',
0.6902601718902588), ('fabulous', 0.6900163888931274)]
==================================================
[('greatest', 0.7815344333648682), ('coolest', 0.7344396710395813), ('best', 0.7160952091217041),
('nastiest', 0.6755936741828918), ('tastiest', 0.6603464484214783), ('disgusting',
0.6454557776451111), ('terrible', 0.636536180973053), ('horrible', 0.6360681056976318), ('awful',
0.6344166398048401), ('nicest', 0.621245265007019)]
```

In [30]:

```
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  ['bought', 'apartment', 'infested', 'fruit', 'flies', 'hours', 'trap', 'attracted',
'many', 'within', 'days', 'practically', 'gone', 'may', 'not', 'long', 'term', 'solution',
'driving', 'crazy', 'consider', 'buying', 'one', 'caution', 'surface', 'sticky', 'try', 'avoid', '
touching', 'really', 'good', 'idea', 'final', 'product', 'outstanding', 'use', 'car', 'window', 'e
verybody', 'asks', 'made', 'two', 'thumbs', 'received', 'shipment', 'could', 'hardly', 'wait', 'lo
ve', 'call']
```

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

[4.4.1.1] Avg W2v

In [31]:

```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this
to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|██████████| 87773/87773 [03:53<00:00, 376.51it/s]
```

```
87773
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 value
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 cell_val = tfidf

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

```
100%|████████████████████████████████████████████████████████████| 4986/4986
[00:20<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 [28]:

```
# Please write all the code with proper documentation
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score


X_train, X_test, y_train, y_test = train_test_split(preprocessed_reviews,final['Score'], test_size=
0.33,shuffle=False) # this is random splitting
```

In [29]:

```
final.shape
```

Out[29]:

```
(87773, 10)
```

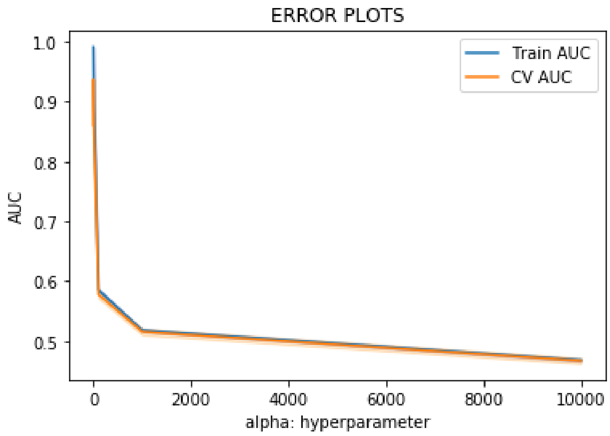
In [30]:

```
alp=[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000,10000]
```

In [88]:

```
vectorizer = CountVectorizer(ngram_range=(1,2),min_df=10)
vectorizer.fit(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_test_bow = vectorizer.transform(X_test)
MNB = MultinomialNB()
parameters = {'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000,10000]}
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(alp, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(alp,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='dar
kblue')

plt.plot(alp, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(alp,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [59]:

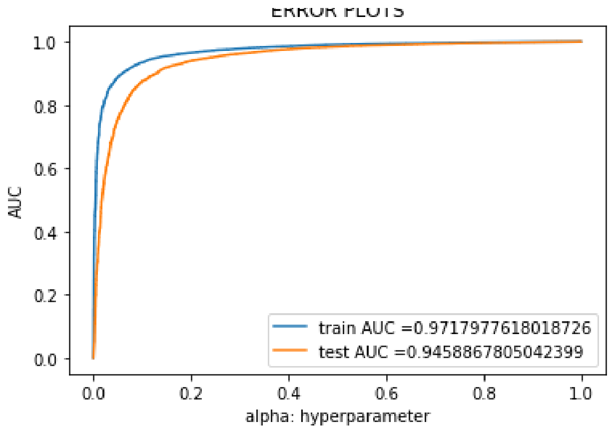
```
clf.best_params_
```

Out[59]:

```
{'alpha': 1}
```

In [89]:

```
# 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=1)
MNB.fit(X_train_bow, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability 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, train_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, clf.predict(X_train_bow)))
print("Test confusion matrix")
print(confusion_matrix(y_test, clf.predict(X_test_bow)))
```




```
==================================================================================================

Train confusion matrix
[[ 5611   768]
 [ 1793 31228]]
Test confusion matrix
[[ 3775   943]
 [ 1472 22776]]
```

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

In [90]:

```
# (below code is taken from taken from given website)
https://stackoverflow.com/questions/29867367/sklearn-multinomial-nb-most-informative-features

feature_names = vectorizer.get_feature_names()
coefs_with_fns = sorted(zip(MNB.coef_[0], feature_names))
top = coefs_with_fns[:-(10 + 1):-1]
for (coef_1, fn_1) in top:
    print("\t%.4f\t%-15s" % (coef_1, fn_1))
```

```
 -3.9576 not
 -4.7584 like
 -4.9046 good
 -4.9807 great
 -5.1239 one
 -5.2118 taste
 -5.2538 coffee
 -5.2861 flavor
 -5.2959 love
 -5.3031 would
```

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

In [91]:

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

# (below code is taken from taken from given website)
https://stackoverflow.com/questions/29867367/sklearn-multinomial-nb-most-informative-features
feature_names = vectorizer.get_feature_names()
coefs_with_fns = sorted(zip(MNB.coef_[0], feature_names))
top = coefs_with_fns[:10]
for (coef_1, fn_1) in top:
    print("\t%.4f\t%-15s" % (coef_1, fn_1))
```

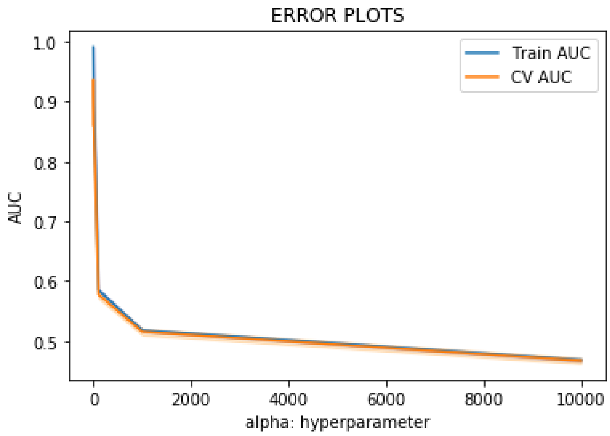
```
 -14.2927 absolutely horrible
 -14.2927 complete waste
 -14.2927 highly disappointed
 -14.2927 horrible not
 -14.2927 money go
 -14.2927 product unless
 -14.2927 recommend unless
```

```
-14.2927 threw rest
-14.2927 wanted love
-14.2927 worst ever
```

[5.2] Applying Naive Bayes on TFIDF, SET 2

In [92]:

```
# Please write all the code with proper documentation
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(X_train) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_tfidf = tf_idf_vect.transform(X_train)
X_test_tfidf = tf_idf_vect.transform(X_test)
MNB = MultinomialNB()
parameters = {'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000,10000]}
clf = GridSearchCV(MNB, parameters, cv=3, scoring='roc_auc')
clf.fit(X_train_bow, y_train)##################################################wrong X_bow to X_tfidf convert it

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(alp, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(alp,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

plt.plot(alp, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(alp,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [93]:

```
clf.best_params_
```

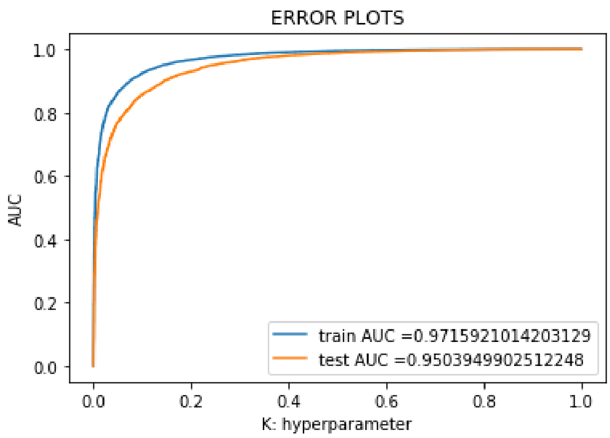
Out[93]:

```
{'alpha': 1}
```

In [94]:

```
# 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=1)
```

```
MNB = MultinomialNB(alpha=1)
MNB.fit(X_train_tfidf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, MNB.predict_proba(X_train_tfidf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, MNB.predict_proba(X_test_tfidf)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: 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_tfidf)))
print("Test confusion matrix")
print(confusion_matrix(y_test, MNB.predict(X_test_tfidf)))
```




```
====================================================================================================

Train confusion matrix
[[ 2196  4183]
 [   66 32955]]
Test confusion matrix
[[ 1143  3575]
 [   55 24193]]
```

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

In [95]:

```
# Please write all the code with proper documentation
feature_names = tf_idf_vect .get_feature_names()
coefs_with_fns = sorted(zip(MNB.coef_[0], feature_names))
top = coefs_with_fns[:-(10 + 1):-1]
for (coef_1, fn_1) in top:
    print("\t%.4f\t%-15s" % (coef_1, fn_1))
```

```
 -5.3292 not
 -5.6725 great
 -5.7348 good
 -5.7823 like
 -5.8374 coffee
 -5.8990 love
 -5.9152 tea
 -6.0199 one
 -6.0301 product
 -6.0344 taste
```

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

In [96]:

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

# (below code is taken from taken from given website)
https://stackoverflow.com/questions/29867367/sklearn-multinomial-nb-most-informative-features
feature_names = vectorizer.get_feature_names()
coefs_with_fns = sorted(zip(MNB.coef_[0], feature_names))
top = coefs_with_fns[:10]
for (coef_1, fn_1) in top:
    print("\t%.4f\t%-15s" % (coef_1, fn_1))
```

```
-12.2468 absolutely horrible
-12.2468 complete waste
-12.2468 highly disappointed
-12.2468 horrible not
-12.2468 money go
-12.2468 product unless
-12.2468 recommend unless
-12.2468 threw rest
-12.2468 wanted love
-12.2468 worst ever
```

In [33]:

```
len_of_review=[]
for i in preprocessed_reviews:
    len_of_review.append(len(i))
```

In [34]:

```
# Please write all the code with proper documentation
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from scipy.sparse import coo_matrix, hstack
 # this is random splitting
```

In [35]:

```
df=pd.DataFrame(data=list(zip(preprocessed_reviews,preprocessed_summary,len_of_review)),columns=["
preprocessed_reviews","preprocessed_summary","len_of_review"])
```

In [36]:

```
df
```

Out[36]:

	preprocessed_reviews	preprocessed_summary	len_of_review
0	bought apartment infested fruit flies hours tr...	sure death flies	207
1	really good idea final product outstanding use...	great product	109
2	received shipment could hardly wait try produc...	wow make islickers	277
3	nothing product bother link top page buy used ...	chewed gum many times used	99
4	love stuff sugar free not rot gums tastes good...	best sugarless gum ever	64
5	never tried kona coffee aloha island definitel...	yummy	582
6	fresh limes underappreciated joy kitchen squir...	limes rule	596
7	grocery store kinds gourmet coffees laid one t...	gave coffees	249

	preprocessed_reviews	preprocessed_summary	len_of_review
8	blend one starbucks gentler blends like taste ...	five stars like starbucks	416
9	chatchi favorite afternoon treat became myster...	fruits labor	500
10	tennis player hubby mine got pack rack opel co...	refreshing mouth freshner	311
11	forget store bought jerky make premium jerky a...	yummy spicy	150
12	sauce excellent indeed spicy brand also makes ...	excellent sauce	172
13	never real swiss fondue really missing somethi...	try	767
14	definitely cute product order get nice amount ...	cute candy	140
15	discovered oils years ago bought one flavor th...	imparts wonderful light flavor dishes	748
16	huge fan jelly belly jelly beans really enjoye...	jelly belly overload	344
17	yes juice apparently not person loves drink le...	fell love paris	495
18	panko bread crumbs awesome used breading make ...	best bread crumbs never go back	711
19	japanese version breadcrumb pan bread portugue...	essential tonkatsu etc	278
20	highly recommend business company like job nic...	everything excellent	82
21	tried many packaged chai products liquid dry s...	simply chai	249
22	first turned onto chocolate visited private me...	chocolate like never	671
23	searched great decaf chai finally found excell...	excellent chai	190
24	enjoy rich spread toast crumpets english muffi...	like grandmother used	211
25	good get half pound cotswold english pub chees...	yes eat one sitting sipping ha ha beware	536
26	winter fresh blueberries exceed food budget dr...	best blueberries	193
27	love sleepytime tea drinking years soothing be...	favorite tea	59
28	shaped like legos taste like sweettarts need said	enjoy playing food	49
29	no exagerration roaring blue wonderful mixture...	best example blue cheese ever tried	304
...	...	...	...
87743	recently returned wonderful three week excursi...	great irish tea	195
87744	got hooked chai decided give coffee still litt...	nice traditional chai	496
87745	love drink mix taste delicious impossible yo f...	drink mix	104
87746	individually packaged assorted rice crackers r...	delicious	188
87747	best chips ever unsalted low sodium diet criti...	michael seasons unsalted potato chips	283
87748	like young coconut tastes like young coconut f...	ai not no coconut water better sun tropics coc...	116
87749	great coffee good price subscription buyer buy...	morning coffee	62
87750	love faucet husband installed one old house cu...	love faucet	182
87751	gone treat dinner treat dogs work run chance l...	dogs love	100
87752	love think little expensive everyday use refre...	love	74
87753	opinion best coconut water especially since no...	love	80
87754	great coffee easy brew coffee great aroma good...	super coffee	94
87755	hubby drinking oz night months found falling d...	sleeping lot better	132
87756	rooibos natural red tea something personal tas...	great tea	246
87757	great hs lunch kid enjoy snack also buy salted...	great hs lunch	63
87758	nifty hot chocolate discs added warm milk milk...	butler chocolate real deal direct ireland	465
87759	loved cranberry like flavor slightly crunchy t...	yummy healthy	108
87760	ordered raisins multiple times always great ar...	delicious	121
87761	love assortment different countries origins fu...	many varieties	88
87762	wanted food dog skin problems skin greatly imp...	great food	299
87763	everyday coffee choice good around crowd pleas...	full bodied without bitter taste	118
87764	go spread serve cheese platter goes well soft ...	best fig spread market	184

	preprocessed_reviews	preprocessed_summary	len_of_review
87765	like product price point flavor strong overpow...	product good	83
87766	tea good perfect price not go wrong product mu...	star tea super price	62
87767	small salty taste good strong good thing packa...	not bad	221
87768	recently new keurig world tried handful flavor...	one best	171
87769	drink lot tea world far worst tasting tea purc...	bad tasting tea	263
87770	tried orange iced coffee morning really liked ...	chike	69
87771	excellent smooth taste one loves chocolate enj...	chocolate heaven	114
87772	purchased product local store ny kids love qui...	delicious	114

87773 rows × 3 columns

In [37]:

```
X_train_r, X_test_r, y_train, y_test = train_test_split(df,final['Score'], test_size=0.33,shuffle=F
alse)
```

In [138]:

```
X_test_r[]
```

Out[138]:

	preprocessed_reviews	preprocessed_summary	len_of_review
58807	crisps awesome give english crisps american ch...	excellent	119
58808	got bag oinkies pig skin sweet potato middle d...	pug loves got different hartz treat	230
58809	individual packaging brand name decent price s...	salted cashews planters not love	162
58810	buy type gummi bears haribo brand awesome qual...	gummi bears ever eat	186
58811	quite pleased e flavor product good strong yet...	yummy hot chocolate	125
58812	serenity house teas arrived today could not ar...	absolutlely best chocolate tea ever tasted	1111
58813	not buy product chicken jerky treat product ca...	made china fda says contaminated	513
58814	purchased licorice hopes like kind purchased d...	tasty	132
58815	ordered chips due readers reviews good additio...	delicious even dont enjoy healthy food	263
58816	oh good first cup medium setting sweet second ...	yummy	123
58817	absolute best hot cocoa keurig brewers hot coc...	favorite hot cocoa	207
58818	great deal try buy store amazon bought came pl...	great deal	109
58819	far best tasting tea around amazon service par...	twinings lady grey decaf	51
58820	hot cocoa flavors tried best far richest flavo...	good flavor wont break bank	246
58821	good quality product two small pooches loved p...	pooches love	143
58822	month old son sees brightly colored pouches si...	great baby food	491
58823	disposable cups great simple clean method port...	disposable k cups keurig brewers	201
58824	love coffee love kind kona coffee decaf regula...	great coffee	81
58825	year old scottish terrier food allergies no wh...	healthy treat dogs allergies	129
58826	try consume organic products led tea drink tea...	one favs	189
58827	colleague making maple brown sugar oatmeal wor...	tasty filling low sugar	223
58828	product arrived promptly packaged efficiently ...	pleased quality	430
58829	love product recommended co worker gout proble...	cherry remedy	163
58830	purchased golden malted natural pancake waffle...	stale	209
58831	mother father recipient wonderful pieces choco...	chocolate heaven	109
58832	nestle mountain blend best flavor coffees inst...	best coffee world	81

	preprocessed_reviews	preprocessed_summary	len_of_review
58833	green tea really nice lite flavor aroma not ov...	great flavor	63
58834	followed microwave directions precisely precis...	warning little food	589
58835	wabash valley purple popcorn really good lot f...	yummy expensive	221
58836	feel love drink spent month trinidad years ago...	ordering	88
...	...	...	...
87743	recently returned wonderful three week excursi...	great irish tea	195
87744	got hooked chai decided give coffee still litt...	nice traditional chai	496
87745	love drink mix taste delicious impossible yo f...	drink mix	104
87746	individually packaged assorted rice crackers r...	delicious	188
87747	best chips ever unsalted low sodium diet criti...	michael seasons unsalted potato chips	283
87748	like young coconut tastes like young coconut f...	ai not no coconut water better sun tropics coc...	116
87749	great coffee good price subscription buyer buy...	morning coffee	62
87750	love faucet husband installed one old house cu...	love faucet	182
87751	gone treat dinner treat dogs work run chance l...	dogs love	100
87752	love think little expensive everyday use refre...	love	74
87753	opinion best coconut water especially since no...	love	80
87754	great coffee easy brew coffee great aroma good...	super coffee	94
87755	hubby drinking oz night months found falling d...	sleeping lot better	132
87756	rooibos natural red tea something personal tas...	great tea	246
87757	great hs lunch kid enjoy snack also buy salted...	great hs lunch	63
87758	nifty hot chocolate discs added warm milk milk...	butler chocolate real deal direct ireland	465
87759	loved cranberry like flavor slightly crunchy t...	yummy healthy	108
87760	ordered raisins multiple times always great ar...	delicious	121
87761	love assortment different countries origins fu...	many varieties	88
87762	wanted food dog skin problems skin greatly imp...	great food	299
87763	everyday coffee choice good around crowd pleas...	full bodied without bitter taste	118
87764	go spread serve cheese platter goes well soft ...	best fig spread market	184
87765	like product price point flavor strong overpow...	product good	83
87766	tea good perfect price not go wrong product mu...	star tea super price	62
87767	small salty taste good strong good thing packa...	not bad	221
87768	recently new keurig world tried handful flavor...	one best	171
87769	drink lot tea world far worst tasting tea purc...	bad tasting tea	263
87770	tried orange iced coffee morning really liked ...	chike	69
87771	excellent smooth taste one loves chocolate enj...	chocolate heaven	114
87772	purchased product local store ny kids love qui...	delicious	114

28966 rows × 3 columns

In [51]:

```
vectorizer = CountVectorizer(ngram_range=(1,2),min_df=10)
```

In [52]:

```
vectorizer.fit(X_train_r["preprocessed_reviews"])
review_train=vectorizer.transform(X_train_r["preprocessed_reviews"])
review_test=vectorizer.transform(X_test_r["preprocessed_reviews"])
```

In [53]:

```
vectorizer.fit(X_train_r["preprocessed_summary"])
summary_train=vectorizer.transform(X_train_r["preprocessed_summary"])
summary_test=vectorizer.transform(X_test_r["preprocessed_summary"])
```

In [54]:

```
print(review_train.shape,summary_train.shape)
```

```
(58807, 34516) (58807, 2973)
```

In [55]:

```
X_train_b=hstack((review_train,summary_train))
X_test_b=hstack((review_test,summary_test))
```

In [63]:

```
X_train_b.shape
```

Out[63]:

```
(58807, 37489)
```

In [65]:

```
X_train_r["len_of_review"].shape
```

Out[65]:

```
(58807,)
```

In [66]:

```
X_train_bow=hstack((X_train_b,X_train_r["len_of_review"].values.reshape(-1,1)))
X_test_bow=hstack((X_test_b,X_test_r["len_of_review"].values.reshape(-1,1)))
```

In [67]:

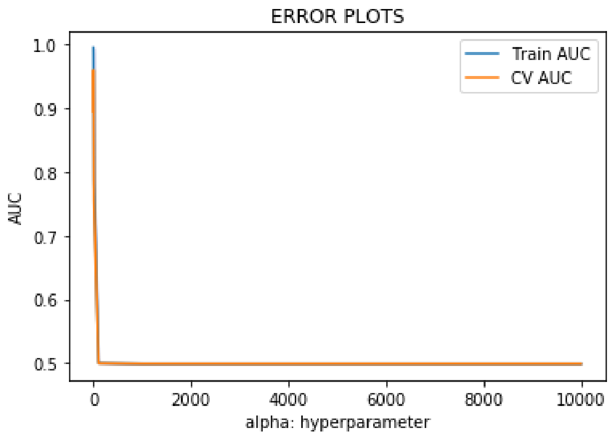
```
X_train_bow.shape
```

Out[67]:

```
(58807, 37490)
```

In [68]:

```
MNB = MultinomialNB()
parameters = {'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000,10000]}
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(alp, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(alp,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='dar
kblue')

plt.plot(alp, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(alp,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [69]:

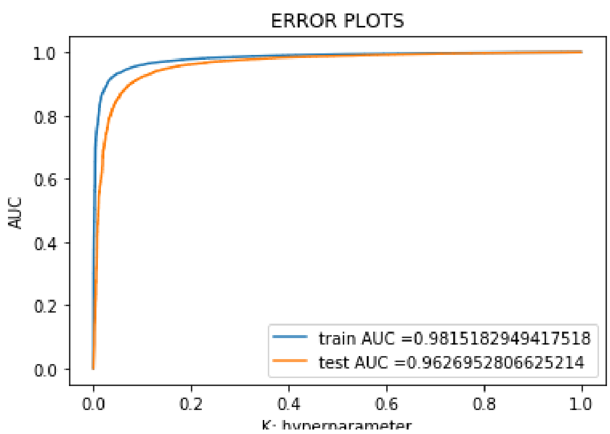
```
clf.best_params_
```

Out[69]:

```
{'alpha': 1}
```

In [70]:

```
# 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=1)
MNB.fit(X_train_bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability 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, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: 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)))
```



K: hyperparameter

```
==========================================================================================

Train confusion matrix
[[ 8426   723]
 [ 2408 47250]]
Test confusion matrix
[[ 4357   675]
 [ 1397 22537]]
```

[6] Conclusions

In [72]:

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

In [73]:

```
pt=PrettyTable()

pt.field_names=["vectorizer","Model","Hyper parameter","AUC"]
pt.add_row(["BOW","multinomialNB(MNB)","1","0.945"])
pt.add_row(["tf-idf","multinomialNB(MNB)","1","0.950"])
pt.add_row(["BOW","MNB with lengthofreview,summmary as features","1","0.962"])
```

In [74]:

```
print(pt)
```

```
+------------+----------------------------------------------+-----------------+-------+
| vectorizer |                    Model                     | Hyper parameter |  AUC  |
+------------+----------------------------------------------+-----------------+-------+
|    BOW     |              multinomialNB(MNB)              |        1        | 0.945 |
|   tf-idf   |              multinomialNB(MNB)              |        1        | 0.950 |
|    BOW     | MNB with lengthofreview,summmary as features |        1        | 0.962 |
+------------+----------------------------------------------+-----------------+-------+
```