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 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 """, 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 (525814, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality Dog Food
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000	Not as Advertised
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia	1	1	1	1219017600	"Delight" says it all

Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary

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
```

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

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANI WAFE

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]:

```
(364173, 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]:

```
69.25890143662969
```

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	Time	Summary
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bought This for My Son at College
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure cocoa taste with crunchy almonds inside

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()
```

```
(364171, 10)
```

Out[13]:

```
1    307061
0     57110
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)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving along a
nd he always can sing the refrain. he's learned about whales, India, drooping roses:  i love all t
he new words this book  introduces and the silliness of it all.  this is a classic book i am
willing to bet my son will STILL be able to recite from memory when he is  in college
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks is good, but I prefer
bolder taste.... imagine my surprise when I ordered 2 boxes - both were expired! One expired back
in 2005 for gosh sakes.  I admit that Amazon agreed to credit me for cost plus part of shipping, b
ut geez, 2 years expired!!!  I'm hoping to find local San Diego area shoppe that carries pods so t
hat I can try something different than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing
I do not think belongs in it is Canola oil. Canola or rapeseed is not someting a dog would ever fi
nd in nature and if it did find rapeseed in nature and eat it, it would poison them. Today's Food
industries have convinced the masses that Canola oil is a safe and even better oil than olive or v
irgin coconut, facts though say otherwise. Until the late 70's it was poisonous until they figured
out a way to fix that. I still like it but it could be better.
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the excellence of this
product.<br /><br />Thick, delicious.  Perfect.  3 ingredients: Water, Maltitol, Natural Maple
Flavor.  PERIOD.  No chemicals.  No garbage.<br /><br />Have numerous friends & family members
hooked on this stuff.  My husband & son, who do NOT like "sugar free" prefer this over major label
regular syrup.<br /><br />I use this as my SWEETENER in baking: cheesecakes, white brownies,
muffins, pumpkin pies, etc... Unbelievably delicious...<br /><br />Can you tell I like it? :)
==================================================
```

In [15]:

```
# 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)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving along a
nd he always can sing the refrain. he's learned about whales, India, drooping roses:  i love all t
he new words this book  introduces and the silliness of it all.  this is a classic book i am
willing to bet my son will STILL be able to recite from memory when he is  in college
```

In [16]:

```
# 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)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving along a
nd he always can sing the refrain. he's learned about whales, India, drooping roses:  i love all t
he new words this book  introduces and the silliness of it all.  this is a classic book i am
willing to bet my son will STILL be able to recite from memory when he is  in college
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks is good, but I prefer
bolder taste.... imagine my surprise when I ordered 2 boxes - both were expired! One expired back
in 2005 for gosh sakes.  I admit that Amazon agreed to credit me for cost plus part of shipping, b
ut geez, 2 years expired!!!  I'm hoping to find local San Diego area shoppe that carries pods so t
hat I can try something different than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing
I do not think belongs in it is Canola oil. Canola or rapeseed is not someting a dog would ever fi
nd in nature and if it did find rapeseed in nature and eat it, it would poison them. Today's Food
industries have convinced the masses that Canola oil is a safe and even better oil than olive or v
irgin coconut, facts though say otherwise. Until the late 70's it was poisonous until they figured
out a way to fix that. I still like it but it could be better.
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the excellence of this
product.Thick, delicious.  Perfect.  3 ingredients: Water, Maltitol, Natural Maple Flavor.
PERIOD.  No chemicals.  No garbage.Have numerous friends & family members hooked on this stuff.  M
y husband & son, who do NOT like "sugar free" prefer this over major label regular syrup.I use thi
s as my SWEETENER in baking: cheesecakes, white brownies, muffins, pumpkin pies, etc...
Unbelievably delicious...Can you tell I like it? :)
```

In [17]:

```
# 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 ingredients although, chicken should have been 1st rather than chicken broth, the only thing
I do not think belongs in it is Canola oil. Canola or rapeseed is not someting a dog would ever fi
```

I do not think belongs in it is Canola oil. Canola or rapeseed is not someting a dog would ever fi
nd in nature and if it did find rapeseed in nature and eat it, it would poison them. Today is Food
industries have convinced the masses that Canola oil is a safe and even better oil than olive or v
irgin coconut, facts though say otherwise. Until the late 70 is it was poisonous until they
figured out a way to fix that. I still like it but it could be better.
==

In [19]:

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

this witty little book makes my son laugh at loud. i recite it in the car as we're driving along a
nd he always can sing the refrain. he's learned about whales, India, drooping roses: i love all t
he new words this book introduces and the silliness of it all. this is a classic book i am
willing to bet my son will STILL be able to recite from memory when he is in college

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 ingredients although chicken should have been 1st rather than chicken broth the only thing I
do not think belongs in it is Canola oil Canola or rapeseed is not someting a dog would ever find
in nature and if it did find rapeseed in nature and eat it it would poison them Today is Food indu
stries have convinced the masses that Canola oil is a safe and even better oil than olive or virgi
n coconut facts though say otherwise Until the late 70 is it was poisonous until they figured out
a way to fix that I still like it but it could be better

In [21]:

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

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "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', "d
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 [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%|██████████| 364171/364171 [01:40<00:00, 3634.32it/s]
```

In [23]:

```
preprocessed_reviews[1500]
```

Out[23]:

```
'great ingredients although chicken rather chicken broth thing not think belongs canola oil canola rapeseed not someting dog would ever find nature find rapeseed nature eat would poison today food industries convinced masses canola oil safe even better oil olive virgin coconut facts though say otherwise late poisonous figured way fix still like could better'
```

[3.2] Preprocessing Review Summary

In [24]:

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

[4] Featurization

In [25]:

```
# split data points to avoid data leakage
from sklearn.model_selection import train_test_split

n_samples= 100000
x_sampled_data= preprocessed_reviews[:n_samples]
y_sampled_data= final.Score[:n_samples]

X_train, x_test, Y_train, y_test= train_test_split(x_sampled_data, y_sampled_data, test_size= .33)
```

[4.1] BAG OF WORDS

In [26]:

```
#BoW
count_vect = CountVectorizer() #in scikit-learn
X_train_bow= count_vect.fit_transform(X_train)
print("some feature names ", count_vect.get_feature_names()[:10])
print("the type of count vectorizer ",type(X_train_bow))
print("the shape of out text BOW vectorizer ",X_train_bow.get_shape())
print('='*50)

X_test_bow = count_vect.transform(x_test)
print("the type of count vectorizer ",type(X_test_bow))
print("the shape of out text BOW vectorizer ",X_test_bow.get_shape())
```

```
some feature names  ['aa', 'aaa', 'aaaaa', 'aaaaaa', 'aaaaaaaaaaaa', 'aaaaaah',
'aaaaaahhhhhyaaaaaa', 'aaaaaand', 'aaaaah', 'aaaaawsome']
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (67000, 49341)
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (33000, 49341)
```

the shape of out text BOW vectorizer (55000, 49541)

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

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

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

[4.3] TF-IDF

In [27]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
X_train_tfidf= tf_idf_vect.fit_transform(X_train)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names()[0:10])
print("the type of count vectorizer ",type(X_train_tfidf))
print("the shape of out text TFIDF vectorizer ",X_train_tfidf.get_shape())
print('='*50)

X_test_tfidf= tf_idf_vect.transform(x_test)
print("the type of count vectorizer ",type(X_test_tfidf))
print("the shape of out text TFIDF vectorizer ",X_test_tfidf.get_shape())
```

```
some sample features(unique words in the corpus) ['ab', 'abandon', 'abc', 'abdominal', 'ability',
'able', 'able break', 'able buy', 'able chew', 'able drink']
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (67000, 37915)
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (33000, 37915)
```

[4.4] Word2Vec

In [26]:

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

In [27]:

```
# 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"
```

```
# 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 ")
```

```
[('good', 0.832168698310852), ('terrific', 0.8253186941146851), ('awesome', 0.8083842992782593), (
'excellent', 0.8074345588684082), ('fantastic', 0.8049054145812988), ('wonderful',
0.7881608009338379), ('perfect', 0.7651262283325195), ('nice', 0.722446084022522), ('amazing', 0.7
004618048667908), ('fabulous', 0.6860787868499756)]
==================================================
[('greatest', 0.7601165771484375), ('disgusting', 0.7121964693069458), ('best',
0.6908397674560547), ('horrible', 0.6568304300308228), ('terrible', 0.6364208459854126), ('awful',
0.6297550201416016), ('tastiest', 0.6062493324279785), ('worse', 0.5978882908821106), ('closest',
0.5957783460617065), ('nastiest', 0.5905071496963501)]
```

In [28]:

```
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  15729
sample words  ['amazingly', 'flavorful', 'hard', 'candy', 'like', 'gummy', 'products', 'hit',
'flavor', 'real', 'fruit', 'spot', 'product', 'high', 'quality', 'flavors', 'incredibly', 'juicy',
'true', 'life', 'mainly', 'actual', 'main', 'ingredient', 'not', 'anything', 'leathers',
'closest', 'thing', 'eating', 'gets', 'terms', 'buy', 'time', 'love', 'particularly', 'lychee', 'k
iwi', 'strawberry', 'sellers', 'image', 'match', 'item', 'shipped', 'us', 'purchased', 'exact', 'c
atnip', 'local', 'pet']
```

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

[4.4.1.1] Avg W2v

In [29]:

```
# average Word2Vec
# compute average word2vec for each review.
X_tr_AvgW2V = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(X_train): # 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
    X_tr_AvgW2V.append(sent_vec)
print(len(X_tr_AvgW2V))
print(len(X_tr_AvgW2V[0]))
```

```
100%|██████████| 67000/67000 [42:13<00:00, 26.44it/s]
```

```
67000
50
```

In [30]:

```
# average Word2Vec
# compute average word2vec for each review.
X_test_AvgW2V = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(x_test): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this
to 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
    X_test_AvgW2V.append(sent_vec)
print(len(X_test_AvgW2V))
print(len(X_test_AvgW2V[0]))
```

```
100%|██████████| 33000/33000 [21:43<00:00, 25.32it/s]
```

```
33000
50
```

[4.4.1.2] TFIDF weighted W2v

In [39]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X_train)
# 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 [40]:

```
# 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

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

```
100%|██████████| 67000/67000 [4:02:16<00:00,  4.96it/s]
```

In [41]:

```
# 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

X_test_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(x_test): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.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
    X_test_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 33000/33000 [1:30:06<00:00,  4.40it/s]
```

[5] Assignment 5: Apply Logistic Regression

1. **Apply Logistic Regression on these feature sets**
    - SET 1:Review text, preprocessed one converted into vectors using (BOW)
    - SET 2:Review text, preprocessed one converted into vectors using (TFIDF)
    - SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)
    - SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)
2. **Hyper paramter tuning (find best hyper parameters corresponding the algorithm that you choose)**
    - Find the best hyper parameter which will give the maximum AUC value
    - Find the best hyper paramter using k-fold cross validation or simple cross validation data
    - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning
3. **Pertubation Test**
    - Get the weights W after fit your model with the data X i.e Train data.
    - Add a noise to the X (X' = X + e) and get the new data set X' (if X is a sparse matrix, X.data+=e)
    - Fit the model again on data X' and get the weights W'
    - Add a small eps value(to eliminate the divisible by zero error) to W and W' i.e W=W+10^-6 and W' = W'+10^-6
    - Now find the % change between W and W' (| (W-W') / (W) |)*100)
    - Calculate the 0th, 10th, 20th, 30th, ...100th percentiles, and observe any sudden rise in the values of percentage_change_vector
    - Ex: consider your 99th percentile is 1.3 and your 100th percentiles are 34.6, there is sudden rise from 1.3 to 34.6, now calculate the 99.1, 99.2, 99.3,..., 100th percentile values and get the proper value after which there is sudden rise the values, assume it is 2.5
    - Print the feature names whose % change is more than a threshold x(in our example it's 2.5)
4. **Sparsity**
    - Calculate sparsity on weight vector obtained after using L1 regularization

Calculate sparsity on weight vector obtained after using L1 regularization

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

5. **Feature importance**
   - Get top 10 important features for both positive and negative classes separately.
6. **Feature engineering**
   - To increase the performance of your model, you can also experiment with with feature engineering like :
     - Taking length of reviews as another feature.
     - Considering some features from review summary as well.
7. **Representation of results**
   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.
   - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
   - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.
8. **Conclusion**
   - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage

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

Applying Logistic Regression

In [32]:

```
# importing libraries
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score

import math
import warnings
warnings.filterwarnings("ignore")
```

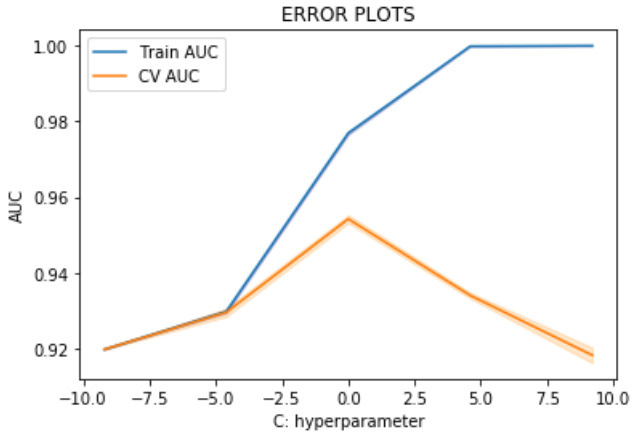
[5.1] Logistic Regression on BOW, SET 1

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

In [35]:

```
# grid search CV
ls=[10**-4, 10**-2, 10**0, 10**2, 10**4]
tuned_parameters = [{'C': ls}]

#Using GridSearchCV
model = GridSearchCV(LogisticRegression(penalty='l1'), tuned_parameters, scoring = 'f1', cv=5)
model.fit(X_train_bow, Y_train)
```

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


log_my_data = [math.log(x) for x in ls]

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

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

print('Best hyper parameter: ', model.best_params_)
print('Model Score: ', model.best_score_)
print('Model estimator: ', model.best_estimator_)
best_C= float(model.best_params_['C'])
```

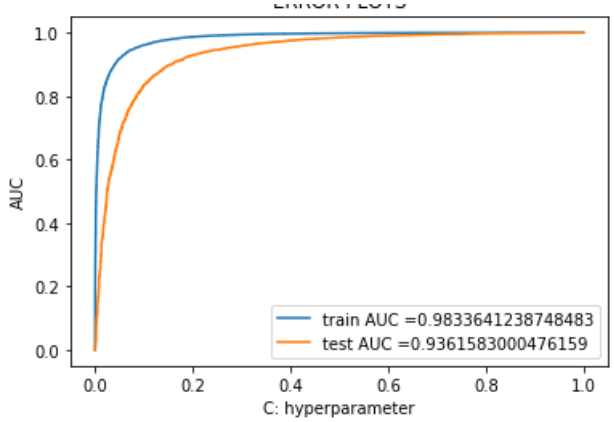



```
Best hyper parameter:  {'C': 1}
Model Score:  0.9542214184699364
Model estimator:  LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [36]:

```
lr_model = LogisticRegression(C= best_C, penalty='l1')
lr_model.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, lr_model.predict_proba(X_train_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, lr_model.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("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print('AUC: ',roc_auc_score(Y_train, lr_model.predict(X_train_bow)))
```

ERROR PLOTS




```
AUC:  0.8876055412330767
```

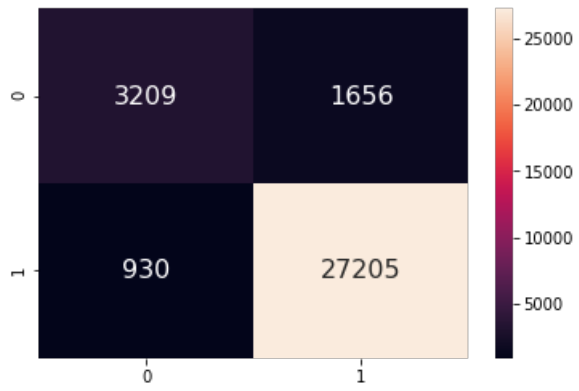
In [37]:

```
from sklearn.metrics import confusion_matrix
print("Test confusion matrix")
print(confusion_matrix(y_test, lr_model.predict(X_test_bow)))

# conprint('AUC: ',roc_auc_score(Y_train, m_nb.predict(X_tr_bow)))fusion matrix visualization usin
g seaborn heatmap
df_test= pd.DataFrame(confusion_matrix(y_test, lr_model.predict(X_test_bow)))
sns.heatmap(df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Test confusion matrix
[[ 3209  1656]
 [  930 27205]]
```

Out[37]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f39a1d486d8>
```



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

In [38]:

```
# More Sparsity (Fewer elements of W* being non-zero) by increasing Lambda (decreasing C)

import numpy as np

clf = LogisticRegression(C= best_C, penalty='l1');
clf.fit(X_train_bow, Y_train);
w = clf.coef_
print('Sparsity: ',np.count_nonzero(w))
```

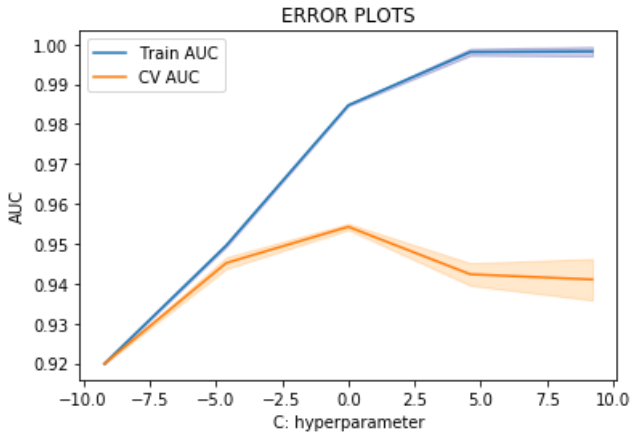
```
Sparsity:  4765
```

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

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

In [39]:

```
# grid search CV
ls=[10**-4, 10**-2, 10**0, 10**2, 10**4]
tuned_parameters = [{'C': ls}]

#Using GridSearchCV
model = GridSearchCV(LogisticRegression(), tuned_parameters, scoring = 'f1', cv=5)
model.fit(X_train_bow, Y_train)

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


log_my_data = [math.log(x) for x in ls]

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

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

print('Best hyper parameter: ', model.best_params_)
print('Model Score: ', model.best_score_)
print('Model estimator: ', model.best_estimator_)
best_C= float(model.best_params_['C'])
```

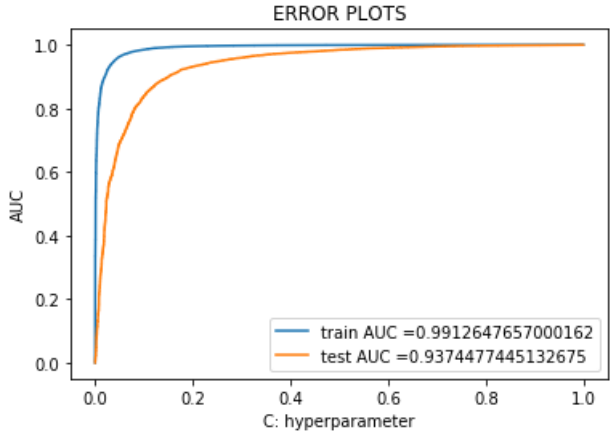



```
Best hyper parameter:  {'C': 1}
Model Score:  0.9542368769165305
Model estimator:  LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [40]:

```
lr_model = LogisticRegression(C= best_C)
lr_model.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, lr_model.predict_proba(X_train_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, lr_model.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("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print('AUC: ',roc_auc_score(Y_train, lr_model.predict(X_train_bow)))
```




```
AUC:  0.920401346573141
```

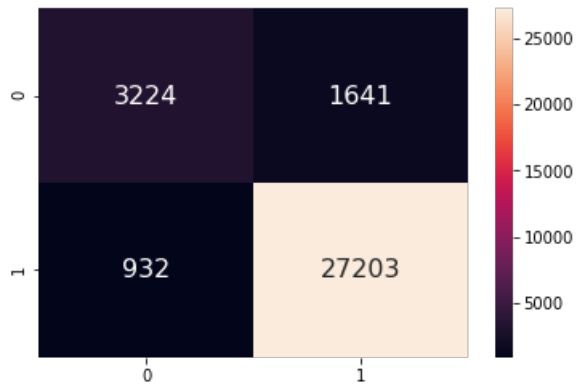
In [41]:

```
from sklearn.metrics import confusion_matrix
print("Test confusion matrix")
print(confusion_matrix(y_test, lr_model.predict(X_test_bow)))

# conprint('AUC: ',roc_auc_score(Y_train, m_nb.predict(X_tr_bow)))fusion matrix visualization usin
g seaborn heatmap
df_test= pd.DataFrame(confusion_matrix(y_test, lr_model.predict(X_test_bow)))
sns.heatmap(df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Test confusion matrix
[[ 3224  1641]
 [  932 27203]]
```

Out[41]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f39a1eb48d0>
```



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

In [58]:

```
# weights on model before adding error to train dataset
from sklearn.metrics import accuracy_score
```

```
lr_model = LogisticRegression(C= 1, penalty='l1')
lr_model.fit(X_train_bow, Y_train)
pred= lr_model.predict(X_test_bow)
print('Accuracy: ', accuracy_score(y_test, pred))

weight_1= lr_model.coef_
```

```
Accuracy:  0.9218787878787879
```

In [60]:

```
# weights on model after adding error to train dataset
from scipy.sparse import find
X_tr_epsilon= X_train_bow

#Random noise
epsilon = np.random.uniform(low=-0.0001, high=0.0001, size=(find(X_tr_epsilon)[0].size,))
#Getting the postions(row and column) and value of non-zero datapoints
a,b,c = find(X_tr_epsilon)

#Introducing random noise to non-zero datapoints
X_tr_epsilon[a,b] = epsilon + X_tr_epsilon[a,b]

lr_model = LogisticRegression(C= 1, penalty='l1')
lr_model.fit(X_tr_epsilon, Y_train)
pred= lr_model.predict(X_test_bow)
print('Accuracy: ', accuracy_score(y_test, pred))

weight_2= lr_model.coef_
```

```
Accuracy:  0.9198787878787879
```

In [68]:

```
diff= (abs(weight_1 - weight_2)/weight_1)*100
print(diff.size)
```

```
49244
```

[5.1.3] Feature Importance on BOW, SET 1

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

In [29]:

```
# Creating the model with our best alpha.
mnb_clf = LogisticRegression(C= 1, penalty='l1')
mnb_clf.fit(X_train_bow, Y_train)
```

Out[29]:

```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [36]:

```
# Ref.:https://www.kaggle.com/laowingkin/amazon-fine-food-review-sentiment-analysis
w = count_vect.get_feature_names()
coef = mnb_clf.coef_.tolist()[0]
coeff_df = pd.DataFrame({'Word' : w, 'Coefficient' : coef})
coeff_df = coeff_df.sort_values(['Coefficient', 'Word'], ascending=[0, 1])
print('')
print('-Top 20 positive-')
print(coeff_df.head(20).to_string(index=False))
print('')
print('-Top 20 negative-')
```

```
print(coeff_df.tail(20).to_string(index=False))
```

```
-Top 20 positive-
Word  Coefficient
  critical     4.969975
   similac     4.451155
  samplers     3.867715
 versatile     3.847230
    modest     3.581931
    ramune     3.505305
pleasantly     3.325100
    kisses     3.294496
 skeptical     3.196754
  security     2.977212
   patient     2.915194
  drawback     2.862605
  grateful     2.855520
     pests     2.810917
     swore     2.744712
      beat     2.701648
    delish     2.690778
   worries     2.664897
  tastiest     2.636445
complement     2.605601

-Top 20 negative-
Word  Coefficient
  unacceptable    -2.662629
       sounded    -2.669663
      avocados    -2.684959
        temper    -2.714811
          mori    -2.779016
      powergel    -2.812268
       bizarre    -2.876170
         worst    -3.210357
        markup    -3.292404
         sunny    -3.328958
        larvae    -3.364811
        archer    -3.367268
      pyrenees    -3.562408
discriminating    -3.571116
      mediocre    -3.739963
   sustainably    -3.971847
       drawing    -4.006706
         schar    -4.345440
       weakest    -4.479616
     sedentary    -9.934590
```

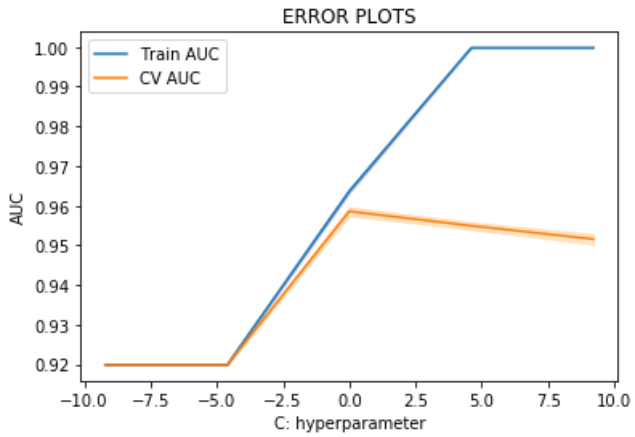
[5.2] Logistic Regression on TFIDF, SET 2

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

In [42]:

```
# grid search CV
ls=[10**-4, 10**-2, 10**0, 10**2, 10**4]
tuned_parameters = [{'C': ls}]

#Using GridSearchCV
model = GridSearchCV(LogisticRegression(penalty='l1'), tuned_parameters, scoring = 'f1', cv=5)
model.fit(X_train_tfidf, Y_train)

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


log_my_data = [math.log(x) for x in ls]

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

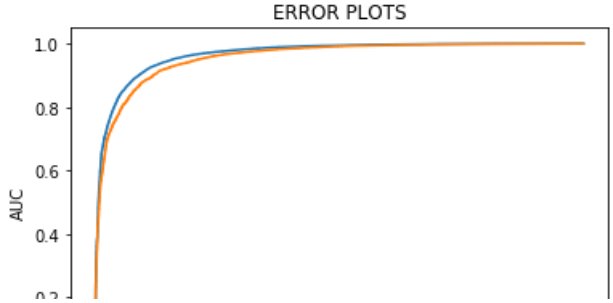
```
plt.gca().fill_between(log_my_data,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,co
lor='darkblue')

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

print('Best hyper parameter: ', model.best_params_)
print('Model Score: ', model.best_score_)
print('Model estimator: ', model.best_estimator_)
best_C= float(model.best_params_['C'])
```

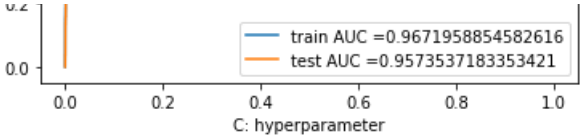



```
Best hyper parameter:  {'C': 1}
Model Score:  0.9585680517772306
Model estimator:  LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [43]:

```
lr_model = LogisticRegression(C= best_C, penalty='l1')
lr_model.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, lr_model.predict_proba(X_train_tfidf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, lr_model.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("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print('AUC: ',roc_auc_score(Y_train, lr_model.predict(X_train_tfidf)))
```






```
AUC:  0.8317892072766967
```

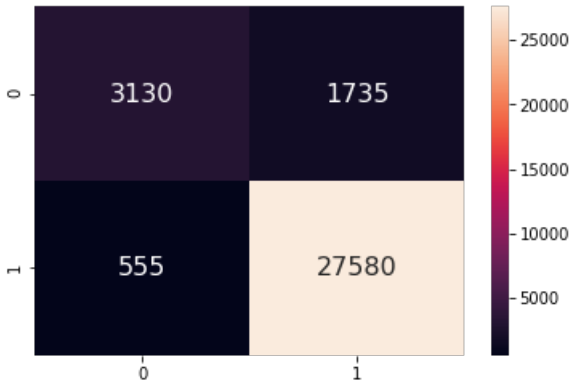
In [44]:

```
from sklearn.metrics import confusion_matrix
print("Test confusion matrix")
print(confusion_matrix(y_test, lr_model.predict(X_test_tfidf)))

# conprint('AUC: ',roc_auc_score(Y_train, m_nb.predict(X_tr_bow)))fusion matrix visualization usin
g seaborn heatmap
df_test= pd.DataFrame(confusion_matrix(y_test, lr_model.predict(X_test_tfidf)))
sns.heatmap(df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Test confusion matrix
[[ 3130  1735]
 [  555 27580]]
```

Out[44]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f39a051fe48>
```

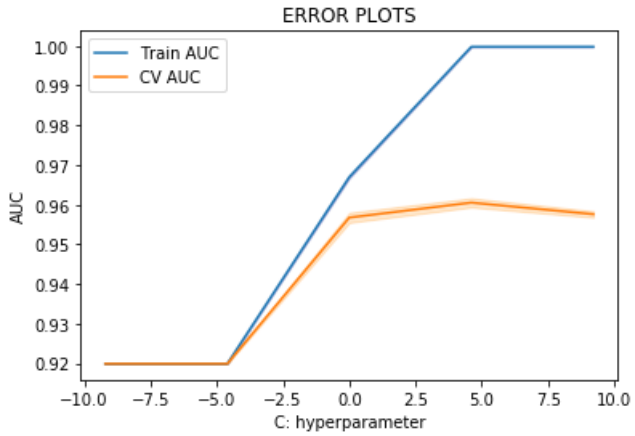


[5.2.2] Applying Logistic Regression with L2 regularization on TFIDF, SET 2

In [45]:

```
# grid search CV
ls=[10**-4, 10**-2, 10**0, 10**2, 10**4]
tuned_parameters = [{'C': ls}]

#Using GridSearchCV
model = GridSearchCV(LogisticRegression(), tuned_parameters, scoring = 'f1', cv=5)
model.fit(X_train_tfidf, Y_train)

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


log_my_data = [math.log(x) for x in ls]

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

plt.plot(log_my_data, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_my_data,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkora
nge')
```

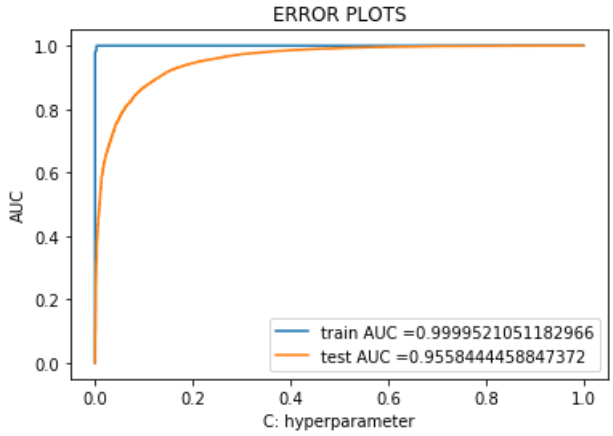
```
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print('Best hyper parameter: ', model.best_params_)
print('Model Score: ', model.best_score_)
print('Model estimator: ', model.best_estimator_)
best_C= float(model.best_params_['C'])
```




```
Best hyper parameter:  {'C': 100}
Model Score:  0.9605091482127504
Model estimator:  LogisticRegression(C=100, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [31]:

```
lr_model = LogisticRegression(C= best_C)
lr_model.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, lr_model.predict_proba(X_train_tfidf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, lr_model.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("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print('AUC: ',roc_auc_score(Y_train, lr_model.predict(X_train_tfidf)))
```



```
AUC:  0.9985218157413416
```

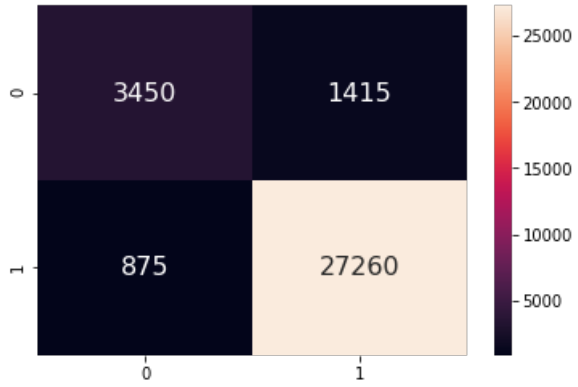
In [47]:

```
from sklearn.metrics import confusion_matrix
print("Test confusion matrix")
print(confusion_matrix(y_test, lr_model.predict(X_test_tfidf)))

# conprint('AUC: ',roc_auc_score(Y_train, m_nb.predict(X_tr_bow)))fusion matrix visualization usin
g seaborn heatmap
df_test= pd.DataFrame(confusion_matrix(y_test, lr_model.predict(X_test_tfidf)))
sns.heatmap(df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Test confusion matrix
[[ 3450  1415]
 [  875 27260]]
```

Out[47]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f39a03f80f0>
```



[5.2.3] Feature Importance on TFIDF, SET 2

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

In [37]:

```
lr_model = LogisticRegression(C= 100)
lr_model.fit(X_train_tfidf, Y_train)
```

Out[37]:

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

In [39]:

```
# Ref.:https://www.kaggle.com/laowingkin/amazon-fine-food-review-sentiment-analysis
w = tf_idf_vect.get_feature_names()
coef = lr_model.coef_.tolist()[0]
coeff_df = pd.DataFrame({'Word' : w, 'Coefficient' : coef})
coeff_df = coeff_df.sort_values(['Coefficient', 'Word'], ascending=[0, 1])
print('')
print('-Top 20 positive-')
print(coeff_df.head(20).to_string(index=False))
print('')
print('-Top 20 negative-')
print(coeff_df.tail(20).to_string(index=False))
```

```
-Top 20 positive-
Word  Coefficient
```

```
           great    27.458773
not disappointed    24.504327
       delicious    23.071549
            best    21.494229
         perfect    18.229366
            good    18.001611
            love    17.180879
       wonderful    17.086806
           loves    16.976579
       excellent    16.353848
         amazing    16.352934
         pleased    16.290647
           happy    14.641841
            glad    14.015649
         awesome    13.918403
       teas like    13.294917
          hooked    13.208958
           yummy    12.778372
       skeptical    12.732661
        favorite    12.381458

-Top 20 negative-
Word  Coefficient
      plus side   -14.107974
      not happy   -14.478391
       horrible   -14.483138
          worse   -14.599591
          bland   -15.061112
     not buying   -15.404931
      not great   -15.634905
 disappointment   -16.066087
       not good   -16.184894
       terrible   -16.872429
          threw   -17.217369
   keep looking   -17.569598
          awful   -17.628371
  disappointing   -17.819516
  not recommend   -18.857677
candy delicious   -19.581901
   disappointed   -20.538792
      two stars   -21.662343
          worst   -22.315208
      not worth   -24.874776
```

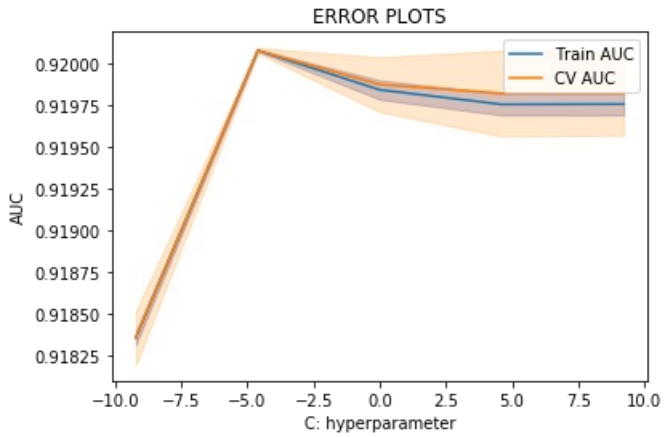
[5.3] Logistic Regression on AVG W2V, SET 3

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

In [33]:

```
X_train_AvgW2v= np.array(X_tr_AvgW2V)
X_test_AvgW2v= np.array(X_test_AvgW2V)
# grid search CV
ls=[10**-4, 10**-2, 10**0, 10**2, 10**4]
tuned_parameters = [{'C': ls}]

#Using GridSearchCV
model = GridSearchCV(LogisticRegression(penalty='l1'), tuned_parameters, scoring = 'f1', cv=5)
model.fit(X_train_AvgW2v, Y_train)

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


log_my_data = [math.log(x) for x in ls]

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

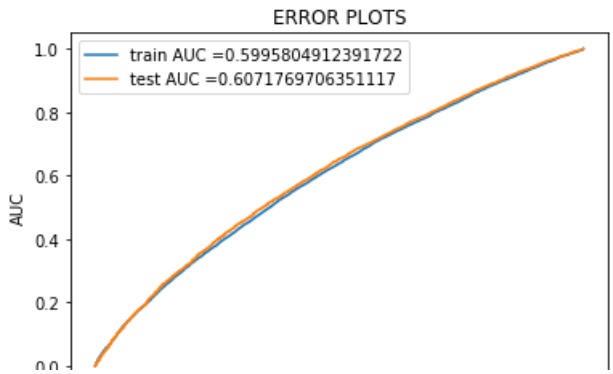
```
plt.plot(log_my_data, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_my_data,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkora
nge')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR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

print('Best hyper parameter: ', model.best_params_)
print('Model Score: ', model.best_score_)
print('Model estimator: ', model.best_estimator_)
best_C= float(model.best_params_['C'])
```

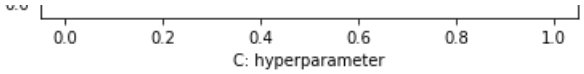



```
Best hyper parameter:  {'C': 0.01}
Model Score:  0.9200776896894183
Model estimator:  LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [34]:

```
lr_model = LogisticRegression(C= best_C, penalty='l1')
lr_model.fit(X_train_AvgW2v, 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, lr_model.predict_proba(X_train_AvgW2v)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, lr_model.predict_proba(X_test_AvgW2v)[:,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("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print('AUC: ',roc_auc_score(Y_train, lr_model.predict(X_train_AvgW2v)))
```






```
AUC:  0.5
```

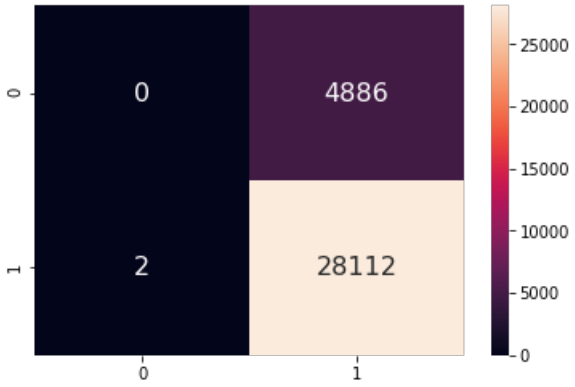
In [35]:

```
from sklearn.metrics import confusion_matrix
print("Test confusion matrix")
print(confusion_matrix(y_test, lr_model.predict(X_test_AvgW2v)))

# conprint('AUC: ',roc_auc_score(Y_train, m_nb.predict(X_tr_bow)))fusion matrix visualization usin
g seaborn heatmap
df_test= pd.DataFrame(confusion_matrix(y_test, lr_model.predict(X_test_AvgW2v)))
sns.heatmap(df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Test confusion matrix
[[    0  4886]
 [    2 28112]]
```

Out[35]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9fc14721d0>
```

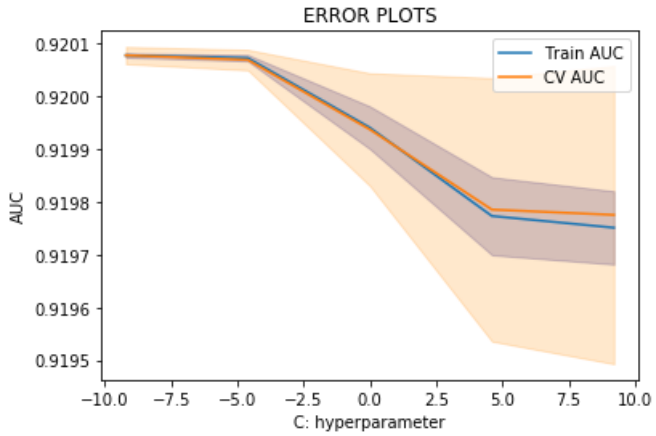


[5.3.2] Applying Logistic Regression with L2 regularization on AVG W2V, SET 3

In [36]:

```
X_train_AvgW2v= np.array(X_train_AvgW2v)
X_test_AvgW2v= np.array(X_test_AvgW2v)

# grid search CV
ls=[10**-4, 10**-2, 10**0, 10**2, 10**4]
tuned_parameters = [{'C': ls}]

#Using GridSearchCV
model = GridSearchCV(LogisticRegression(), tuned_parameters, scoring = 'f1', cv=5)
model.fit(X_train_AvgW2v, Y_train)

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


log_my_data = [math.log(x) for x in ls]

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

plt.plot(log_my_data, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_my_data,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkora
```

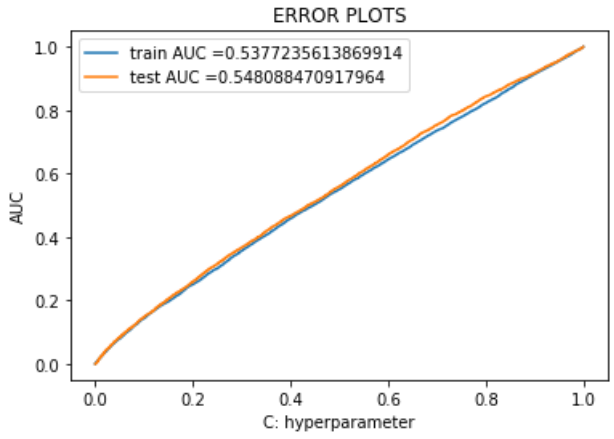
```
nge')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print('Best hyper parameter: ', model.best_params_)
print('Model Score: ', model.best_score_)
print('Model estimator: ', model.best_estimator_)
best_C= float(model.best_params_['C'])
```




```
Best hyper parameter:  {'C': 0.0001}
Model Score:  0.9200776896894183
Model estimator:  LogisticRegression(C=0.0001, class_weight=None, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
```

In [37]:

```
lr_model = LogisticRegression(C= best_C)
lr_model.fit(X_train_AvgW2v, 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, lr_model.predict_proba(X_train_AvgW2v)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, lr_model.predict_proba(X_test_AvgW2v)[:,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("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print('AUC: ',roc_auc_score(Y_train, lr_model.predict(X_train_AvgW2v)))
```



```
AUC:  0.5
```

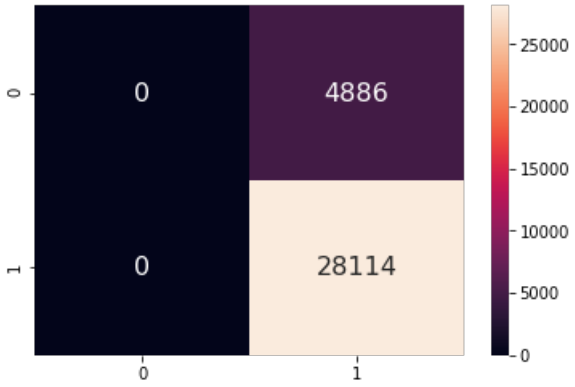
In [38]:

```
from sklearn.metrics import confusion_matrix
print("Test confusion matrix")
print(confusion_matrix(y_test, lr_model.predict(X_test_AvgW2v)))

# conprint('AUC: ',roc_auc_score(Y_train, m_nb.predict(X_tr_bow)))fusion matrix visualization usin
g seaborn heatmap
df_test= pd.DataFrame(confusion_matrix(y_test, lr_model.predict(X_test_AvgW2v)))
sns.heatmap(df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Test confusion matrix
[[    0  4886]
 [    0 28114]]
```

Out[38]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9fc1467ba8>
```



[5.4] Logistic Regression on TFIDF W2V, SET 4

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

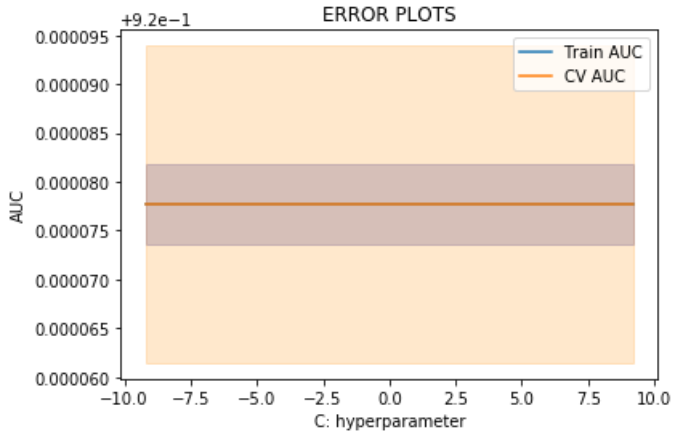
In [42]:

```
X_train_tfidfW2V= np.array(X_tr_tfidf_sent_vectors)
X_test_tfidfW2V= np.array(X_test_tfidf_sent_vectors)
```

In [43]:

```
# grid search CV
ls=[10**-4, 10**-2, 10**0, 10**2, 10**4]
tuned_parameters = [{'C': ls}]

#Using GridSearchCV
model = GridSearchCV(LogisticRegression(penalty='l1'), tuned_parameters, scoring = 'f1', cv=5)
model.fit(X_train_tfidfW2V, Y_train)

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


log_my_data = [math.log(x) for x in ls]

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

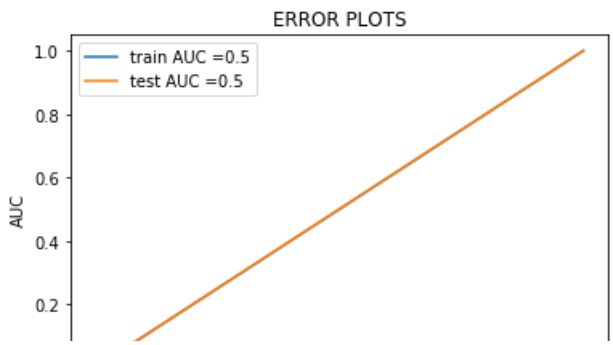
```
plt.plot(log_my_data, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_my_data,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkora
nge')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR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

print('Best hyper parameter: ', model.best_params_)
print('Model Score: ', model.best_score_)
print('Model estimator: ', model.best_estimator_)
best_C= float(model.best_params_['C'])
```

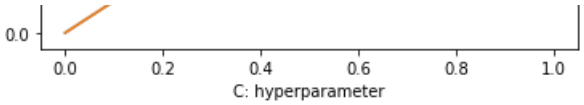



```
Best hyper parameter:  {'C': 0.0001}
Model Score:  0.9200776896894183
Model estimator:  LogisticRegression(C=0.0001, class_weight=None, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l1', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
```

In [44]:

```
lr_model = LogisticRegression(C= best_C, penalty='l1')
lr_model.fit(X_train_tfidfW2V, 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, lr_model.predict_proba(X_train_tfidfW2V)[:,1]
)
test_fpr, test_tpr, thresholds = roc_curve(y_test, lr_model.predict_proba(X_test_tfidfW2V)[:,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("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print('AUC: ',roc_auc_score(Y_train, lr_model.predict(X_train_tfidfW2V)))
```






```
AUC:  0.5
```

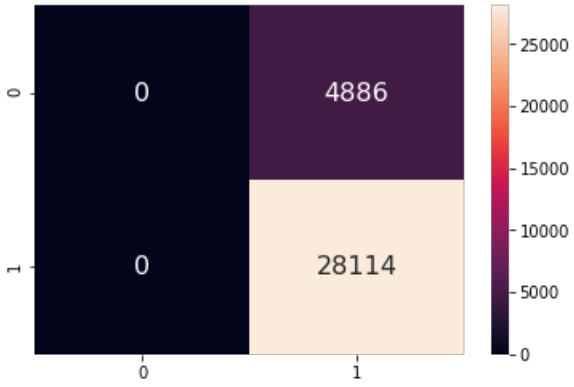
In [45]:

```
from sklearn.metrics import confusion_matrix
print("Test confusion matrix")
print(confusion_matrix(y_test, lr_model.predict(X_test_tfidfW2V)))

# conprint('AUC: ',roc_auc_score(Y_train, m_nb.predict(X_tr_bow)))fusion matrix visualization usin
g seaborn heatmap
df_test= pd.DataFrame(confusion_matrix(y_test, lr_model.predict(X_test_tfidfW2V)))
sns.heatmap(df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Test confusion matrix
[[    0  4886]
 [    0 28114]]
```

Out[45]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9fc006e080>
```

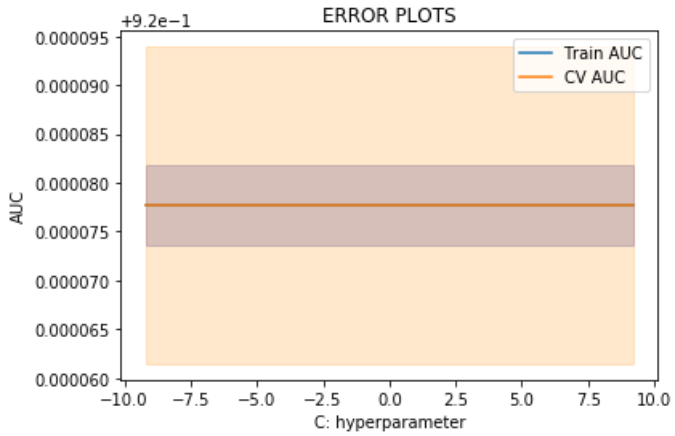


[5.4.2] Applying Logistic Regression with L2 regularization on TFIDF W2V, SET 4

In [46]:

```
# grid search CV
ls=[10**-4, 10**-2, 10**0, 10**2, 10**4]
tuned_parameters = [{'C': ls}]

#Using GridSearchCV
model = GridSearchCV(LogisticRegression(), tuned_parameters, scoring = 'f1', cv=5)
model.fit(X_train_tfidfW2V, Y_train)

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


log_my_data = [math.log(x) for x in ls]

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

plt.plot(log_my_data, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_my_data,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkora
nge')
plt.legend()
```

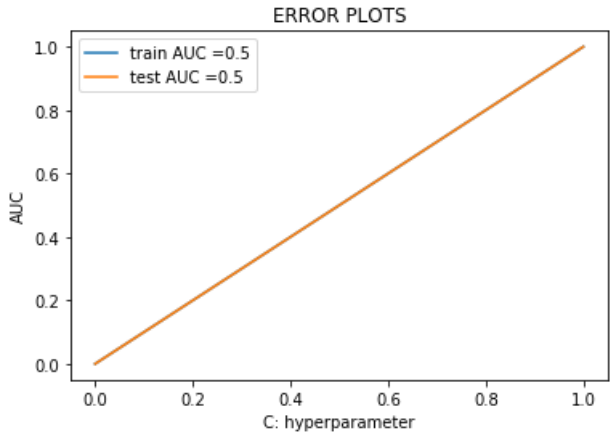
```
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print('Best hyper parameter: ', model.best_params_)
print('Model Score: ', model.best_score_)
print('Model estimator: ', model.best_estimator_)
best_C= float(model.best_params_['C'])
```




```
Best hyper parameter:  {'C': 0.0001}
Model Score:  0.9200776896894183
Model estimator:  LogisticRegression(C=0.0001, class_weight=None, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
```

In [47]:

```
lr_model = LogisticRegression(C= best_C)
lr_model.fit(X_train_tfidfW2V, 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, lr_model.predict_proba(X_train_tfidfW2V)[:,1]
)
test_fpr, test_tpr, thresholds = roc_curve(y_test, lr_model.predict_proba(X_test_tfidfW2V)[:,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("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print('AUC: ',roc_auc_score(Y_train, lr_model.predict(X_train_tfidfW2V)))
```



```
AUC:  0.5
```

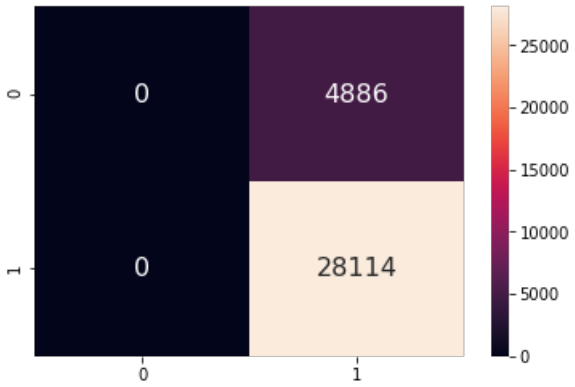
In [48]:

```
from sklearn.metrics import confusion_matrix
print("Test confusion matrix")
print(confusion_matrix(y_test, lr_model.predict(X_test_tfidfW2V)))

# conprint('AUC: ',roc_auc_score(Y_train, m_nb.predict(X_tr_bow)))fusion matrix visualization usin
g seaborn heatmap
df_test= pd.DataFrame(confusion_matrix(y_test, lr_model.predict(X_test_tfidfW2V)))
sns.heatmap(df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Test confusion matrix
[[    0  4886]
 [    0 28114]]
```

Out[48]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9fb24f2da0>
```



[6] Conclusions

In [45]:

```
# http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()
y = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Hyper parameter", "AUC", "Train(%)", "Test(%)"]
y.field_names = ["Vectorizer", "Model", "Hyper parameter", "AUC", "Train(%)", "Test(%)"]

x.add_row(["BOW", 'Logistic Regression',1, .88, 98, 93])
y.add_row(["BOW", 'Logistic Regression',  1, .92, 99, 93])

x.add_row(["BOW", 'Logistic Regression',  1, .81, 96, 95])
y.add_row(["BOW", 'Logistic Regression',  100, .99, 99, 95])

x.add_row(["Avg-W2Vec", 'Logistic Regression', .01, .5, 60, 60])
y.add_row(["Avg-W2Vec", 'Logistic Regression', .0001, .5, 54, 55])

x.add_row(["TFIDF-W2Vec", 'Logistic Regression', .0001, .5, .5, .5])
y.add_row(["TFIDF-W2Vec", 'Logistic Regression', .0001, .5, .5, .5])

print('\t\t\t\t- L1 -')
print(x)
print('\t\t\t\t- L2 -')
print(y)
```

```
    - L1 -
+-------------+---------------------+-----------------+------+----------+---------+
|  Vectorizer |        Model        | Hyper parameter | AUC  | Train(%) | Test(%) |
+-------------+---------------------+-----------------+------+----------+---------+
```

```
|     BOW     | Logistic Regression |        1        | 0.88 |    98    |    93    |
|     BOW     | Logistic Regression |        1        | 0.81 |    96    |    95    |
|  Avg-W2Vec  | Logistic Regression |        1        | 0.71 |    90    |    48    |
| TFIDF-W2Vec | Logistic Regression |      0.0001     | 0.5  |   0.5    |   0.5    |
+-------------+---------------------+-----------------+------+----------+----------+
```

- L2 -

```
+-------------+---------------------+-----------------+------+----------+----------+
|  Vectorizer |        Model        | Hyper parameter | AUC  | Train(%) | Test(%)  |
+-------------+---------------------+-----------------+------+----------+----------+
|     BOW     | Logistic Regression |        1        | 0.92 |    99    |    93    |
|     BOW     | Logistic Regression |       100       | 0.99 |    99    |    95    |
|  Avg-W2Vec  | Logistic Regression |       100       | 0.72 |    90    |    48    |
| TFIDF-W2Vec | Logistic Regression |      0.0001     | 0.5  |   0.5    |   0.5    |
+-------------+---------------------+-----------------+------+----------+----------+
```

In [ ]: