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
```

```python
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]:

```python
# 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 500
000""", con)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 11500
0""", 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 (115000, 10)
```

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

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)
```

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...	2

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
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']
```

	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()
```

```
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()
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Sum
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADR VANILL WAFER
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADR VANILL WAFER
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADR VANILL WAFER
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADR VANILL WAFER
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADR VANILL WAFER

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='quicksort', na_position='last')
```

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

```
(99724, 10)
```

In [10]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

```
86.71652173913044
```

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()
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bought This for Son at College
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure co taste wi 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()
```

```
(99722, 10)


1    83711
0    16011
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

In [14]:

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

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

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

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

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its very hard to find any chicken
products made in the USA but they are out there, but this one isnt.  Its too bad too because its a good product but I wont t
ake any chances till they know what is going on with the china imports.
==================================================
IF YOU LIKE SALMON YOU WILL LOVE THESE OMAHA STEAKS SALMON VERY VERY GOOD
==================================================
OK....I thought I'd put a bit of punch to hubby's sandwich, instead of the ho-hum Best Foods Mayo---ohOoooOh--FAILURE!<br />
One bite and he said---Please! DO NOT EVER SERVE THIS TO ME AGAIN!<br />I guess it was that-bad!<br />I'll see if my neighbo
r will be able to use it w/her family.<br />If you  are a BEST FOODS lover---walk away---do NOT purchase this product!
==================================================
These people from Bavaria really know how to make this stuff. The Landjagers are super (you have to let them dry for them to
develop their full, intended flavor), and it is worth any sausage fan's time to check out their complete offering of German
style sausages and hams.  Due to the perishability of some of their products their S&H charges appear outrageous but I guess
that sending frozen or refrigerated foods costs money.  Personally, I recommend their coarse grind liverwurst but that is a
matter of personal taste.
==================================================
```

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)
```

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

In [16]:

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

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

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

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

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

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its very hard to find any chicken
```

```
products made in the USA but they are out there, but this one isnt.  Its too bad too because its a good product but I wont t
ake any chances till they know what is going on with the china imports.
==================================================
IF YOU LIKE SALMON YOU WILL LOVE THESE OMAHA STEAKS SALMON VERY VERY GOOD
==================================================
OK....I thought I'd put a bit of punch to hubby's sandwich, instead of the ho-hum Best Foods Mayo---ohOooo0h--FAILURE!One bi
te and he said---Please! DO NOT EVER SERVE THIS TO ME AGAIN!I guess it was that-bad!I'll see if my neighbor will be able to
use it w/her family.If you  are a BEST FOODS lover---walk away---do NOT purchase this product!
==================================================
These people from Bavaria really know how to make this stuff. The Landjagers are super (you have to let them dry for them to
develop their full, intended flavor), and it is worth any sausage fan's time to check out their complete offering of German
style sausages and hams.  Due to the perishability of some of their products their S&H charges appear outrageous but I guess
that sending frozen or refrigerated foods costs money.  Personally, I recommend their coarse grind liverwurst but that is a
matter of personal taste.
```

In [17]:

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

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

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

In [18]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
OK....I thought I would put a bit of punch to hubby is sandwich, instead of the ho-hum Best Foods Mayo---ohOoooOh--FAILURE!<
br />One bite and he said---Please! DO NOT EVER SERVE THIS TO ME AGAIN!<br />I guess it was that-bad!<br />I will see if my
neighbor will be able to use it w/her family.<br />If you  are a BEST FOODS lover---walk away---do NOT purchase this produc
t!
==================================================
```

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)
```

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

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

```
OK I thought I would put a bit of punch to hubby is sandwich instead of the ho hum Best Foods Mayo ohOoooOh FAILURE br One b
ite and he said Please DO NOT EVER SERVE THIS TO ME AGAIN br I guess it was that bad br I will see if my neighbor will be ab
le to use it w her family br If you are a BEST FOODS lover walk away do NOT purchase this product
```

In [21]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step
```

```
stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves'
, 'you', "you'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', 'th
ey', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "tha
t'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
d', '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', 'ove
r', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any'
, 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too',
 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
w', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'might
n', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wa
sn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

```
In [22]:
```

```python
# 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 stopword
s)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 99722/99722 [00:47<00:00, 2109.25it/s]
```

In [23]:

```python
preprocessed_reviews[1500]
```

```
'ok thought would put bit punch hubby sandwich instead ho hum best foods mayo ohoooooh failure one bite said please not ever
serve guess bad see neighbor able use w family best foods lover walk away not purchase product'
```

In [24]:

```python
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import cross_val_score
from collections import Counter
import collections
from sklearn.metrics import accuracy_score,roc_auc_score,roc_curve,confusion_matrix,auc
from sklearn import cross_validation
from scipy.sparse import csr_matrix,hstack
from sklearn.linear_model import LogisticRegression
```

```
/usr/local/lib/python3.5/dist-packages/sklearn/cross_validation.py:41: DeprecationWarning: This module was deprecated in ver
sion 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also note th
at the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [25]:
```
X_1, X_test, y_1, y_test = cross_validation.train_test_split(preprocessed_reviews,final[
'Score'], test_size=0.2, random_state=0)
X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(X_1, y_1, test_size=0.25)
print(np.asarray(X_1).shape,np.asarray(X_test).shape,np.asarray(X_tr).shape,np.asarray(X
_test).shape,np.asarray(X_cv).shape)
```

```
(79777,) (19945,) (59832,) (19945,) (19945,)
```

[3.2] Preprocessing Review Summary

In [26]:
```
## Similartly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

In [27]:
```
#BoW
count_vect = CountVectorizer() #in scikit-learn
BOW_Train = count_vect.fit_transform(X_tr)
BOW_test = count_vect.transform(X_test)
BOW_CV = count_vect.transform(X_cv)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)
```

```
print("the type of count vectorizer ",type(BOW_Train))
print("the shape of out text BOW vectorizer ",BOW_Train.get_shape())
print("the number of unique words ", BOW_Train.get_shape()[1])
```

```
some feature names  ['aa', 'aaa', 'aaaa', 'aaaaa', 'aaaaaaaaaaaa', 'aaaaaaaaaaaaaaa', 'aaaaah', 'aaaah', 'aaah', 'aaahs']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (59832, 45894)
the number of unique words  45894
```

[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/module
s/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_co
unts.get_shape()[1])
```

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

[4.3] TF-IDF

In [28]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
TFIDF_Train = tf_idf_vect.fit_transform(X_tr)
TFIDF_Test = tf_idf_vect.transform(X_test)
TFIDF_Validation = tf_idf_vect.transform(X_cv)
print("the type of count vectorizer ",type(TFIDF_Train))
print("the shape of out text TFIDF vectorizer ",TFIDF_Train.get_shape())
print("the number of unique words including both unigrams and bigrams ", TFIDF_Train.get
_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (59832, 35323)
the number of unique words including both unigrams and bigrams  35323
```

[4.4] Word2Vec

In [54]:
```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
list_of_sentance_cv=[]
list_of_sentance_test=[]
for sentance in X_tr:
    list_of_sentance.append(sentance.split())
for sentance in X_cv:
    list_of_sentance_cv.append(sentance.split())
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())
```

In [55]:
```
# Using Google News Word2Vectors
```

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


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

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

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

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin'
, binary=True)
```

```
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your own w2v ")
```

```
[('awesome', 0.8380345702171326), ('fantastic', 0.8272818326950073), ('good', 0.8190262317657471), ('excellent', 0.8023777008056641), ('perfect', 0.7637239694595337), ('terrific', 0.7582295536994934), ('wonderful', 0.7574710845947266), ('amazing', 0.709833025932312), ('nice', 0.6972397565841675), ('fabulous', 0.6679291725158691)]
==================================================
[('greatest', 0.798704981803894), ('best', 0.7466772198677063), ('tastiest', 0.7069478034973145), ('nastiest', 0.6879305839538574), ('horrible', 0.6495988368988037), ('disgusting', 0.6158303022384644), ('nicest', 0.6137348413467407), ('smoothest', 0.5876367092132568), ('superior', 0.5830625295639038), ('awful', 0.5795249938964844)]
```

In [56]:

```
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  14680
sample words  ['berry', 'dental', 'cinnamony', 'intolerable', 'vendors', 'ass', 'ins', 'sadly', 'forward', 'tropical', 'orgeat', 'necap', 'grad', 'addendum', 'national', 'seattle', 'wedding', 'studies', 'bluegrass', 'lodged', 'wetlands', 'prescription', 'seemed', 'digit', 'west', 'normals', 'affect', 'fusilli', 'chambers', 'commit', 'questioning', 'gfcf', 'al', 'pacific', 'greasier', 'nanny', 'lattes', 'bean', 'welfare', 'moderately', 'chewie', 'ganocafe', 'brits', 'shock', 'kelloggs', 'praline', 'capable', 'clarification', 'wal', 'lindt']
```

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

[4.4.1.1] Avg W2v

In [57]:

```
# 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 c
hange 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)
sent_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance_cv): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to c
hange 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_cv.append(sent_vec)
sent_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance_test): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to c
hange 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_test.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|██████████| 59832/59832 [08:32<00:00, 116.84it/s]
100%|██████████| 19945/19945 [02:13<00:00, 149.71it/s]
100%|██████████| 19945/19945 [02:09<00:00, 154.10it/s]


59832
50
```

[4.4.1.2] TFIDF weighted W2v

In [58]:

```
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X_tr)
tf_idf_matrix_cv = model.transform(X_cv)
tf_idf_matrix_test = model.transform(X_test)
# 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 [59]:

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

```
#               tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_cv.append(sent_vec)
    row += 1
tfidf_sent_vectors_test = []; # the tfidf-w2v for each sentence/review is stored in this
 list
row=0;
for sent in tqdm(list_of_sentance_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
```

```
    tfidf_sent_vectors_test.append(sent_vec)
    row += 1
```

```
100%|██████████| 59832/59832 [42:57<00:00, 23.21it/s]
100%|██████████| 19945/19945 [12:56<00:00, 25.69it/s]
100%|██████████| 19945/19945 [15:05<00:00, 22.02it/s]
```

[5] Assignment 5: Apply Logistic Regression

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

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

3. **Pertubation Test**
    - Get the weights W after fit your model with the data X i.e Train data.
    - Add a noise to the X (X' = X + e) and get the new data set X' (if X is a sparse matrix, X.data+=e)
    - Fit the model again on data X' and get the weights W'

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

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

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

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

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

7. **Representation of results**
   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.

Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

8. **Conclusion**

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

Note: Data Leakage

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

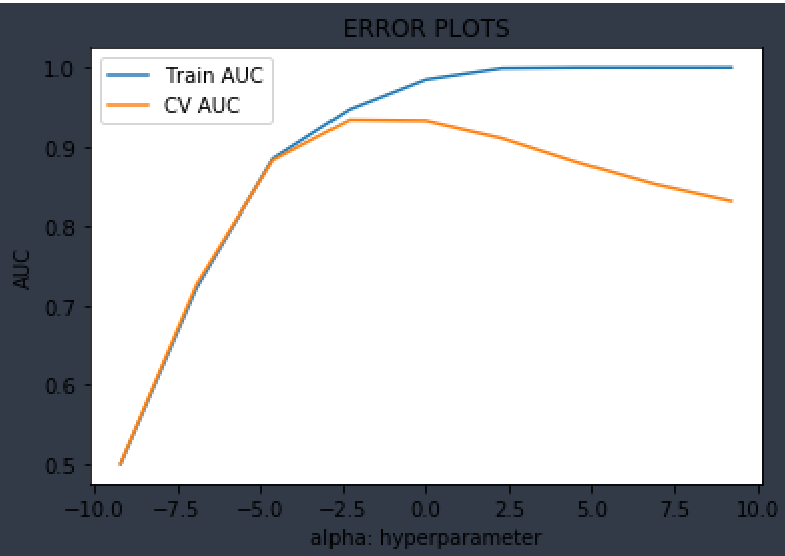
Applying Logistic Regression

[5.1] Logistic Regression on BOW, SET 1

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

In [29]:
```
# Please write all the code with proper documentation
lamda= [10**(-4),10**(-3),10**(-2),10**(-1),1,10,100,1000,10000]
```

```
BOW_val_accuracy = []
BOW_train_accuracy = []
for i in lamda:
    model = LogisticRegression(C=i,penalty='l1')
    model.fit(BOW_Train,y_tr)
    val_data  = model.predict_log_proba(BOW_CV)[:,1]
    train_data = model.predict_log_proba(BOW_Train)[:,1]
    BOW_val_accuracy.append(roc_auc_score(np.asarray(y_cv),np.asarray(val_data)))
    BOW_train_accuracy.append(roc_auc_score(np.asarray(y_tr),np.asarray(train_data)))
plt.plot(np.log(np.asarray(lamda)), BOW_train_accuracy, label='Train AUC')
plt.plot(np.log(np.asarray(lamda)), BOW_val_accuracy, label='CV AUC')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

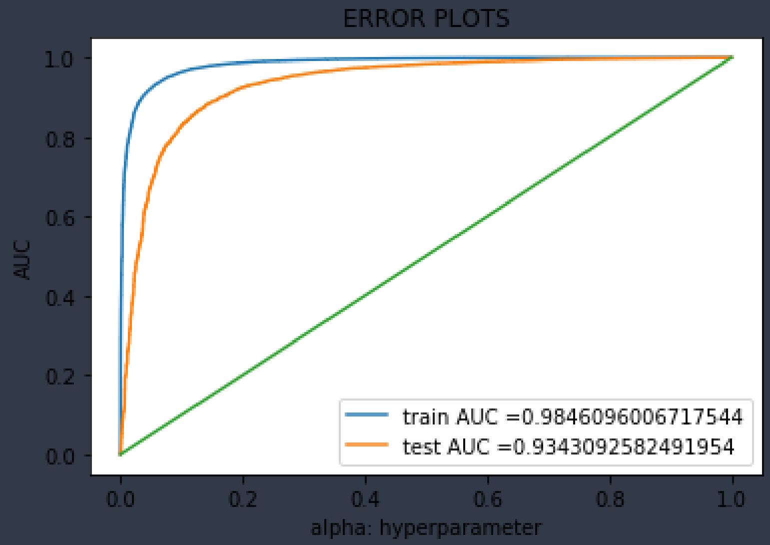


In [45]:
```
best_lambda = 1
```

Testing on our test data

In [46]:
```
model = LogisticRegression(C=best_lambda,penalty='l1')
model.fit(BOW_Train,y_tr)
test_pred  = model.predict_log_proba(BOW_test)[:,1]
train_pred = model.predict_log_proba(BOW_Train)[:,1]
train_fpr, train_tpr, thresholds = roc_curve(y_tr, train_pred)
test_fpr, test_tpr, thresholds = roc_curve(y_test, test_pred)

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.plot([0.0,1.0],[0.0,1.0])
plt.legend()
plt.xlabel("alpha: hyperparameter")
```

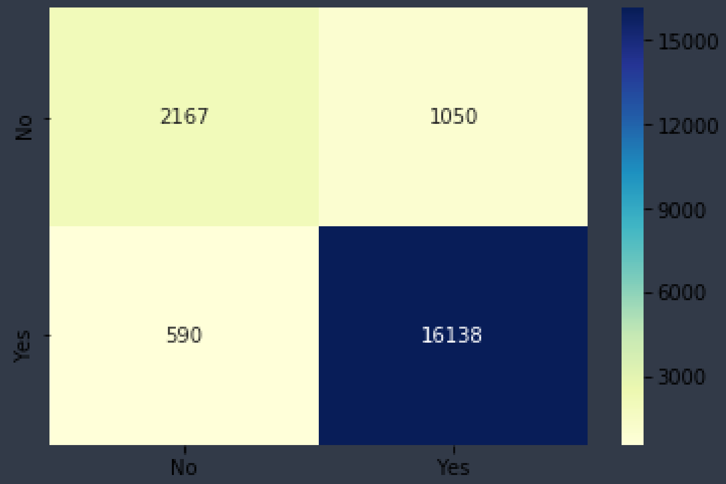
```
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



Confusion Matrix

In [47]:
```
ytrain = model.predict(BOW_Train)
ytest = model.predict(BOW_test)
ctrain = confusion_matrix(y_tr,ytrain)
ctest = confusion_matrix(y_test,ytest)
class_label=["No","Yes"]
df = pd.DataFrame(ctest, index=class_label, columns=class_label)
sns.heatmap(df, annot= True, fmt="d", cmap="YlGnBu")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0bb7df9550>
```



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

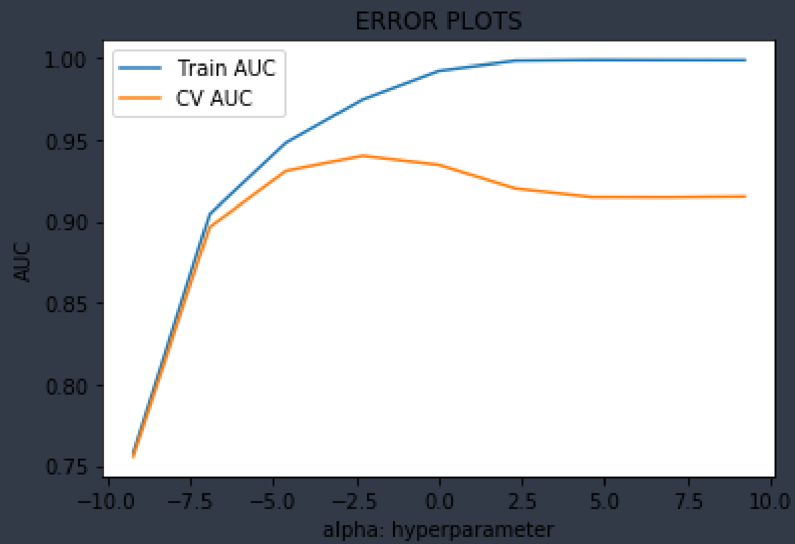
In [33]:
```
# Please write all the code with proper documentation
cf = model.coef_
zr = np.count_nonzero(cf)
sparsity = ((cf.shape[1] - zr)/(cf.shape[1]))*float(100)
print(sparsity)
# we get sparsity of 90%`
```

```
90.11635507909531
```

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

In [34]:
```
# Please write all the code with proper documentation
lamda= [10**(-4),10**(-3),10**(-2),10**(-1),1,10,100,1000,10000]
BOW_val_accuracy = []
BOW_train_accuracy = []
```

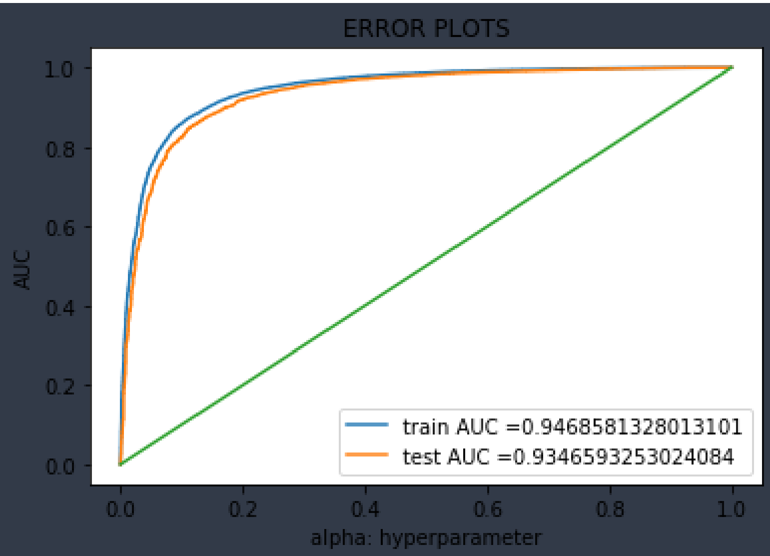
```python
for i in lamda:
    model = LogisticRegression(C=i,penalty='l2')
    model.fit(BOW_Train,y_tr)
    val_data  = model.predict_log_proba(BOW_CV)[:,1]
    train_data = model.predict_log_proba(BOW_Train)[:,1]
    BOW_val_accuracy.append(roc_auc_score(np.asarray(y_cv),np.asarray(val_data)))
    BOW_train_accuracy.append(roc_auc_score(np.asarray(y_tr),np.asarray(train_data)))
plt.plot(np.log(np.asarray(lamda)), BOW_train_accuracy, label='Train AUC')
plt.plot(np.log(np.asarray(lamda)), BOW_val_accuracy, label='CV AUC')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



```
In [48]: best_lambda= 0.1
```

Testing on test data

```
In [49]: model = LogisticRegression(C=best_lambda,penalty='l1')
model.fit(BOW_Train,y_tr)
test_pred  = model.predict_log_proba(BOW_test)[:,1]
train_pred = model.predict_log_proba(BOW_Train)[:,1]
train_fpr, train_tpr, thresholds = roc_curve(y_tr, train_pred)
test_fpr, test_tpr, thresholds = roc_curve(y_test, test_pred)

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.plot([0.0,1.0],[0.0,1.0])
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

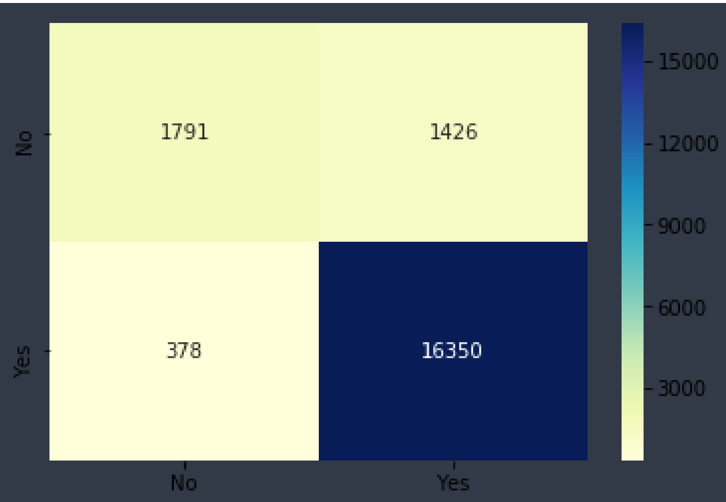


Confusion Matrix

In [50]:

```
ytrain = model.predict(BOW_Train)
ytest = model.predict(BOW_test)
ctrain = confusion_matrix(y_tr,ytrain)
ctest = confusion_matrix(y_test,ytest)
class_label=["No","Yes"]
df = pd.DataFrame(ctest, index=class_label, columns=class_label)
sns.heatmap(df, annot= True, fmt="d", cmap="YlGnBu")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0bbb113908>
```



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

In [132]:
```
# Please write all the code with proper documentation
weights1 = model.coef_
e = 0.000001
new_BOW_Train = BOW_Train
new_BOW_Train.data=new_BOW_Train.data + e
new_BOW_test = BOW_test
new_BOW_test.data=new_BOW_test.data+ e
model1 = LogisticRegression(C=best_lambda,penalty='l1')
model1.fit(new_BOW_Train,y_tr)
weights2 = model1.coef_
```

In [133]:
```
weights1 = weights1+0.000001
weigths2 = weights2+0.000001
weight_percent = ((weights1-weigths2)/weights1)*100
```

```python
for i in range(0,45713,456):
    print(weight_percent[0][i])
```

```
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
```

```
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
```

```
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
-0.0016191168141451685
0.0
0.0
0.0
0.0
0.0
0.0
0.0
-0.15781420611134978
0.0
0.0
0.0
```

In [136]:

```python
## printing all features with percentage greater than threshhold
for i in range(45713):
    if weight_percent[0][i]>2.5 or weight_percent[0][i]<-2.5:
        print(count_vect.get_feature_names()[i],weight_percent[0][i],i)
```

```
based -14.238995076376773 3193
burned -3.2633420217946094 5274
despite 5.2968565172808 10883
funny 4.1140790650513575 16224
```

```
olives -3.867465097760274 27876
packaging 2.7437721800017605 28759
pounds -10.61089232206223 30928
value -10.542017135356518 43314
wine -3.9408887834904545 44866
```

[5.1.3] Feature Importance on BOW, SET 1

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

In [77]:
```
# Please write all the code with proper documentation
a = model.coef_
b= []
for i in range(45884):
    if a[0][i]>0:
        b.append((i,a[0][i]))
b =  sorted(b,key= lambda x: x[1],reverse=True)
b = b[0:10]
print(b)
print(" So the top 10 features of positive class are--")
for i in range(0,10):
    print("feature name : %s , value : %f"%(count_vect.get_feature_names()[b[i][0]],b[i]
[1]))
```

```
[(12996, 7.158571543687503), (36034, 4.033675229657431), (44431, 3.462121134857459), (15934, 3.3211885786230115), (30396, 3.2743345963066335), (12118, 3.11413061010694), (18054, 2.978076681226532), (45460, 2.964576588615704), (12646, 2.955601654350972), (14689, 2.7590335293324326)]
 So the top 10 features of positive class are--
feature name : emeraldforest , value : 7.158572
feature name : shavings , value : 4.033675
feature name : welcome , value : 3.462121
feature name : friday , value : 3.321189
feature name : pleasantly , value : 3.274335
```

```
feature name : downside , value : 3.114131
feature name : haha , value : 2.978077
feature name : yay , value : 2.964577
feature name : eco , value : 2.955602
feature name : feridies , value : 2.759034
```

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

In [79]:

```
a = model.coef_
b= []
for i in range(45884):
    if a[0][i]<0:
        b.append((i,a[0][i]))
b =  sorted(b,key= lambda x: x[1])
b = b[0:10]
print(b)
print(" So the top 10 features of positive class are--")
for i in range(0,10):
    print("feature name : %s , value : %f"%(count_vect.get_feature_names()[b[i][0]],b[i]
[1]))
```

```
[(10921, -4.0737926595237655), (20596, -3.817204983543581), (10275, -3.587709824185055), (42664, -3.542857745757743), (5708, 
-3.3801663851103414), (44429, -3.323688218348887), (19740, -3.31508177089095), (42468, -3.2431838210135444), (40334, -3.2125
63195754616), (45180, -3.2067936376849766)]
 So the top 10 features of positive class are--
feature name : detangling , value : -4.073793
feature name : insult , value : -3.817205
feature name : deceptive , value : -3.587710
feature name : undrinkable , value : -3.542858
feature name : canceled , value : -3.380166
feature name : welch , value : -3.323688
feature name : ifyou , value : -3.315082
feature name : unappealing , value : -3.243184
```

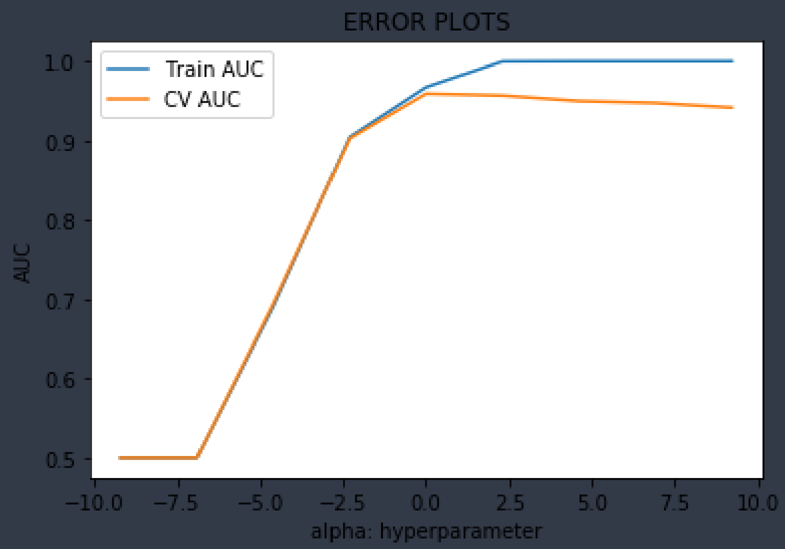
```
feature name : teeter , value : -3.212563
feature name : worst , value : -3.206794
```

In [120]:
```
# Please write all the code with proper documentation
```

[5.2] Logistic Regression on TFIDF, SET 2

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

In [31]:
```
# Please write all the code with proper documentation

lamda= [10**(-4),10**(-3),10**(-2),10**(-1),1,10,100,1000,10000]
tfidf_val_accuracy = []
tfidf_train_accuracy = []
for i in lamda:
    model = LogisticRegression(C=i,penalty='l1')
    model.fit(TFIDF_Train,y_tr)
    val_data  = model.predict_log_proba(TFIDF_Validation)[:,1]
    train_data = model.predict_log_proba(TFIDF_Train)[:,1]
    tfidf_val_accuracy.append(roc_auc_score(np.asarray(y_cv),np.asarray(val_data)))
    tfidf_train_accuracy.append(roc_auc_score(np.asarray(y_tr),np.asarray(train_data)))
plt.plot(np.log(np.asarray(lamda)), tfidf_train_accuracy, label='Train AUC')
plt.plot(np.log(np.asarray(lamda)), tfidf_val_accuracy, label='CV AUC')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

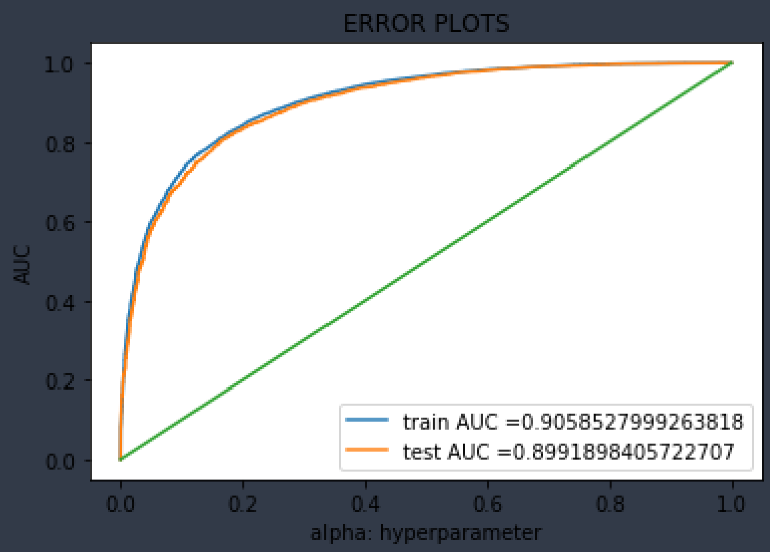


In [36]:
```
best_lambda  = 1
```

Testing on test data

In [38]:
```
model = LogisticRegression(C=best_lambda,penalty='l1')
model.fit(TFIDF_Train,y_tr)
test_pred  = model.predict_log_proba(TFIDF_Test)[:,1]
train_pred = model.predict_log_proba(TFIDF_Train)[:,1]
train_fpr, train_tpr, thresholds = roc_curve(y_tr, train_pred)
test_fpr, test_tpr, thresholds = roc_curve(y_test, test_pred)

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.plot([0.0,1.0],[0.0,1.0])
plt.legend()
```

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

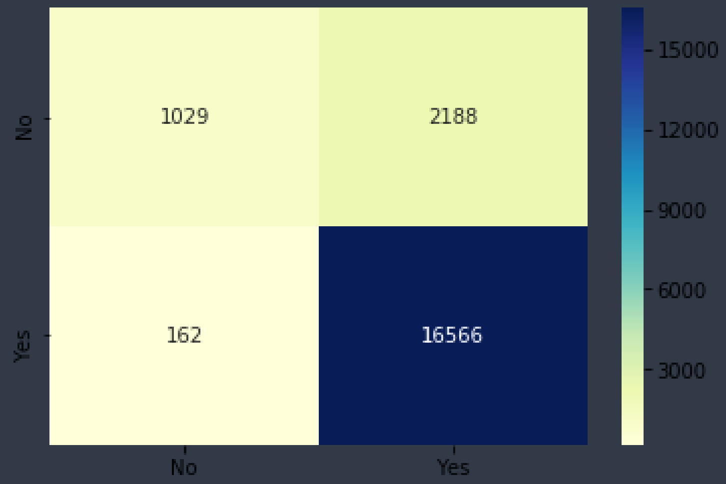


Confusion Matrix

In [40]:

```
ytrain = model.predict(TFIDF_Train)
ytest = model.predict(TFIDF_Test)
ctrain = confusion_matrix(y_tr,ytrain)
ctest = confusion_matrix(y_test,ytest)
class_label=["No","Yes"]
df = pd.DataFrame(ctest, index=class_label, columns=class_label)
sns.heatmap(df, annot= True, fmt="d", cmap="YlGnBu")
```

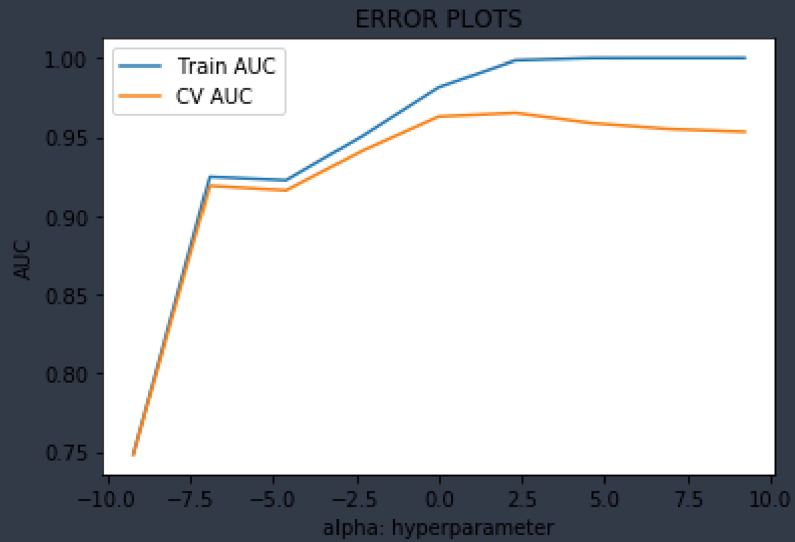
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0bc0b815c0>
```



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

In [40]:
```
# Please write all the code with proper documentation
lamda= [10**(-4),10**(-3),10**(-2),10**(-1),1,10,100,1000,10000]
tfidf_val_accuracy = []
tfidf_train_accuracy = []
for i in lamda:
    model = LogisticRegression(C=i,penalty='l2')
    model.fit(TFIDF_Train,y_tr)
    val_data  = model.predict_log_proba(TFIDF_Validation)[:,1]
    train_data = model.predict_log_proba(TFIDF_Train)[:,1]
    tfidf_val_accuracy.append(roc_auc_score(np.asarray(y_cv),np.asarray(val_data)))
    tfidf_train_accuracy.append(roc_auc_score(np.asarray(y_tr),np.asarray(train_data)))
plt.plot(np.log(np.asarray(lamda)), tfidf_train_accuracy, label='Train AUC')
plt.plot(np.log(np.asarray(lamda)), tfidf_val_accuracy, label='CV AUC')
```

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



In [51]:

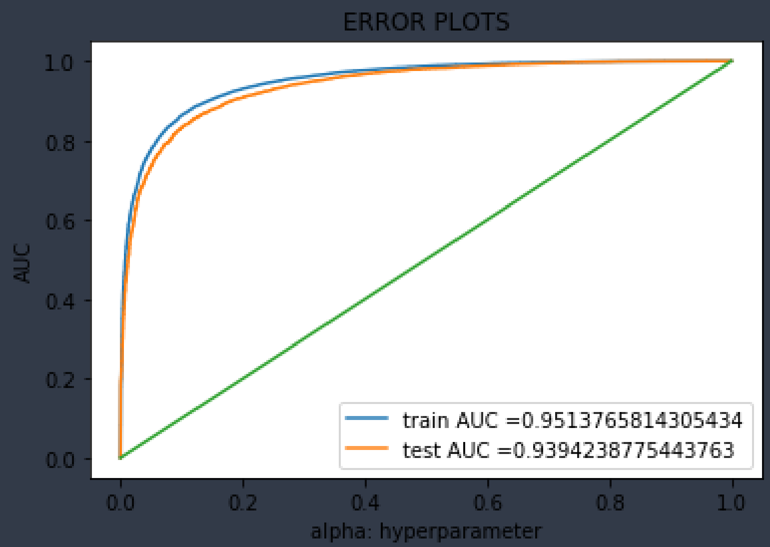
```
best_lambdat_lambda = 10
```

Testing on test data

In [52]:

```
model = LogisticRegression(C=best_lambda,penalty='l2')
model.fit(TFIDF_Train,y_tr)
test_pred  = model.predict_log_proba(TFIDF_Test)[:,1]
train_pred = model.predict_log_proba(TFIDF_Train)[:,1]
train_fpr, train_tpr, thresholds = roc_curve(y_tr, train_pred)
test_fpr, test_tpr, thresholds = roc_curve(y_test, test_pred)
```

```
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.plot([0.0,1.0],[0.0,1.0])
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



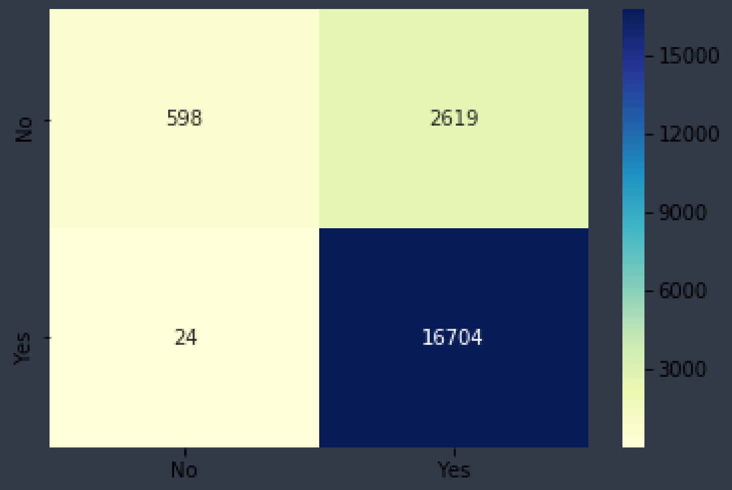
Confusion Matrix

In [53]:

```
ytrain = model.predict(TFIDF_Train)
ytest = model.predict(TFIDF_Test)
ctrain = confusion_matrix(y_tr,ytrain)
ctest = confusion_matrix(y_test,ytest)
```

```
class_label=["No","Yes"]
df = pd.DataFrame(ctest, index=class_label, columns=class_label)
sns.heatmap(df, annot= True, fmt="d", cmap="YlGnBu")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0bb8e70ac8>
```



[5.2.3] Feature Importance on TFIDF, SET 2

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

In [70]:

```
# Please write all the code with proper documentation
a = model.coef_
b= []
for i in range(35256):
    if a[0][i]>0:
        b.append((i,a[0][i]))
b =  sorted(b,key= lambda x: x[1],reverse=True)
```

```
b = b[0:10]
print(b)
print(" So the top 10 features of positive class are--")
for i in range(0,10):
    print("feature name : %s , value : %f"%(tf_idf_vect.get_feature_names()[b[i][0]],b[i
][1]))
```

```
[(13514, 11.260877431627256), (7394, 8.08283089623781), (2357, 7.98280014683183), (12946, 7.145354849178723), (23079, 6.7339
24630835463), (17672, 6.597381560703448), (17934, 6.303618044902684), (9650, 5.897557746316617), (34611, 5.495279280860837),
(20170, 5.448845291852297)]
 So the top 10 features of positive class are--
feature name : great , value : 11.260877
feature name : delicious , value : 8.082831
feature name : best , value : 7.982800
feature name : good , value : 7.145355
feature name : perfect , value : 6.733925
feature name : love , value : 6.597382
feature name : loves , value : 6.303618
feature name : excellent , value : 5.897558
feature name : wonderful , value : 5.495279
feature name : nice , value : 5.448845
```

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

In [72]:

```
# Please write all the code with proper documentation
a = model.coef_
b= []
for i in range(35256):
    if a[0][i]<0:
        b.append((i,a[0][i]))
b =  sorted(b,key= lambda x: x[1])
b = b[0:10]
print(b)
```

```
print(" So the top 10 features of positive class are--")
for i in range(0,10):
    print("feature name : %s , value : %f"%(tf_idf_vect.get_feature_names()[b[i][0]],b[i
][1]))
```

```
[(7861, -8.174959606578012), (20555, -7.406783563734185), (34763, -6.8849930993209245), (20849, -6.78703308552511), (1680, -6.213259352049335), (7879, -6.027657536276124), (31162, -6.014125142183329), (20640, -5.938605669690055), (21300, -5.693618048709711), (21090, -5.345525656201999)]
 So the top 10 features of positive class are--
feature name : disappointed , value : -8.174960
feature name : not , value : -7.406784
feature name : worst , value : -6.884993
feature name : not good , value : -6.787033
feature name : awful , value : -6.213259
feature name : disappointing , value : -6.027658
feature name : terrible , value : -6.014125
feature name : not buy , value : -5.938606
feature name : not worth , value : -5.693618
feature name : not recommend , value : -5.345526
```

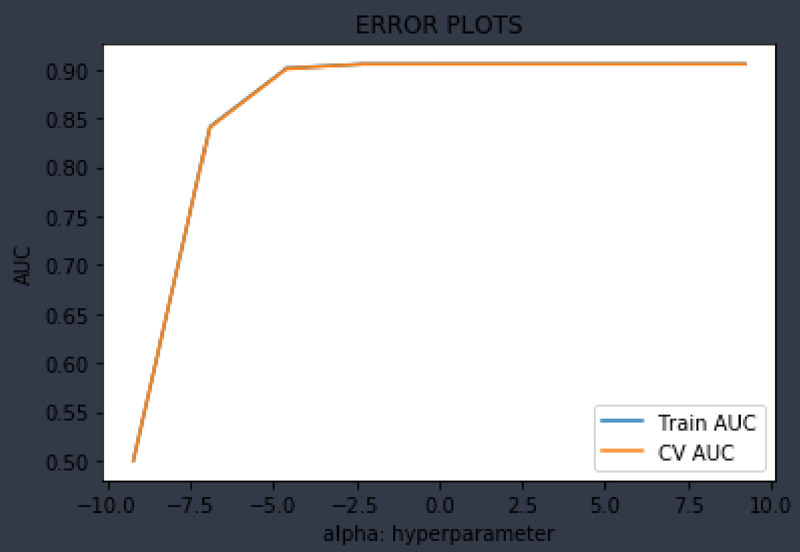
[5.3] Logistic Regression on AVG W2V, SET 3

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

In [97]:

```
# Please write all the code with proper documentation
lamda= [10**(-4),10**(-3),10**(-2),10**(-1),1,10,100,1000,10000]
aw2v_val_accuracy = []
aw2v_train_accuracy = []
for i in lamda:
    model = LogisticRegression(C=i,penalty='l1')
    model.fit(sent_vectors,y_tr)
```

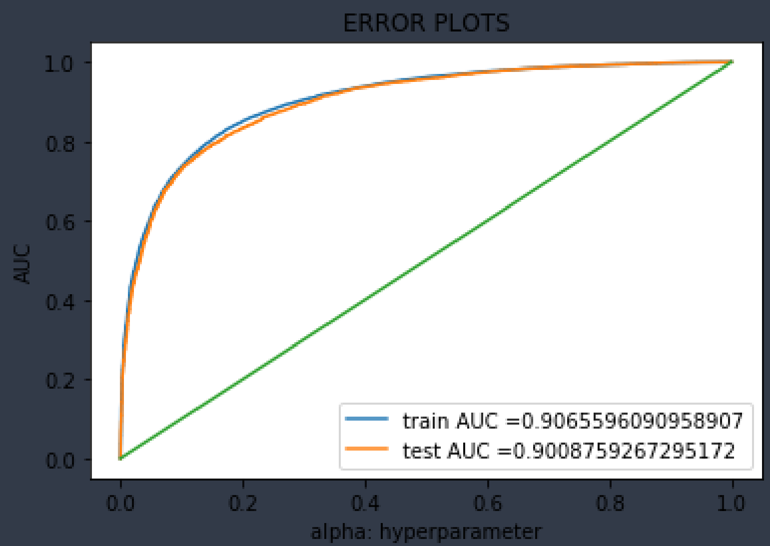
```python
    val_data  = model.predict_log_proba(sent_vectors_cv)[:,1]
    train_data = model.predict_log_proba(sent_vectors)[:,1]
    aw2v_val_accuracy.append(roc_auc_score(np.asarray(y_cv),np.asarray(val_data)))
    aw2v_train_accuracy.append(roc_auc_score(np.asarray(y_tr),np.asarray(train_data)))
plt.plot(np.log(np.asarray(lamda)), aw2v_train_accuracy, label='Train AUC')
plt.plot(np.log(np.asarray(lamda)), aw2v_val_accuracy, label='CV AUC')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [60]:
```python
best_lambda =1
```

Testing on test data

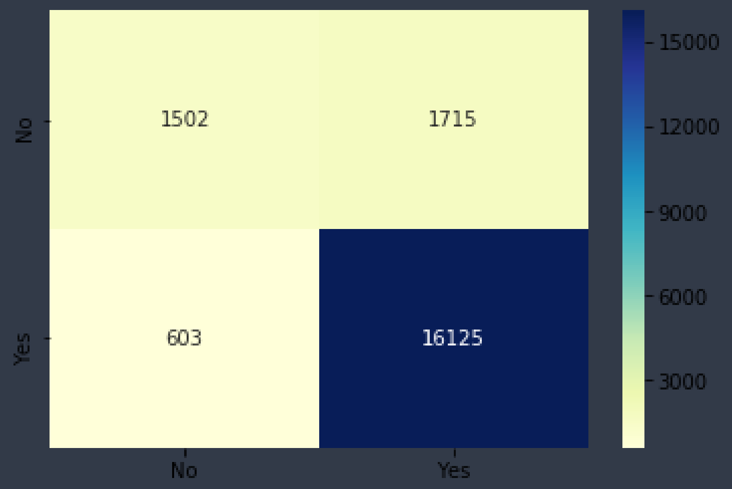
```
In [61]: model = LogisticRegression(C=best_lambda,penalty='l1')
         model.fit(sent_vectors,y_tr)
         test_pred  = model.predict_log_proba(sent_vectors_test)[:,1]
         train_pred = model.predict_log_proba(sent_vectors)[:,1]
         train_fpr, train_tpr, thresholds = roc_curve(y_tr, train_pred)
         test_fpr, test_tpr, thresholds = roc_curve(y_test, test_pred)

         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.plot([0.0,1.0],[0.0,1.0])
         plt.legend()
         plt.xlabel("alpha: hyperparameter")
         plt.ylabel("AUC")
         plt.title("ERROR PLOTS")
         plt.show()
```



Confusion Matrix

In [62]:
```
ytrain = model.predict(sent_vectors)
ytest = model.predict(sent_vectors_test)
ctrain = confusion_matrix(y_tr,ytrain)
ctest = confusion_matrix(y_test,ytest)
class_label=["No","Yes"]
df = pd.DataFrame(ctest, index=class_label, columns=class_label)
sns.heatmap(df, annot= True, fmt="d", cmap="YlGnBu")
```

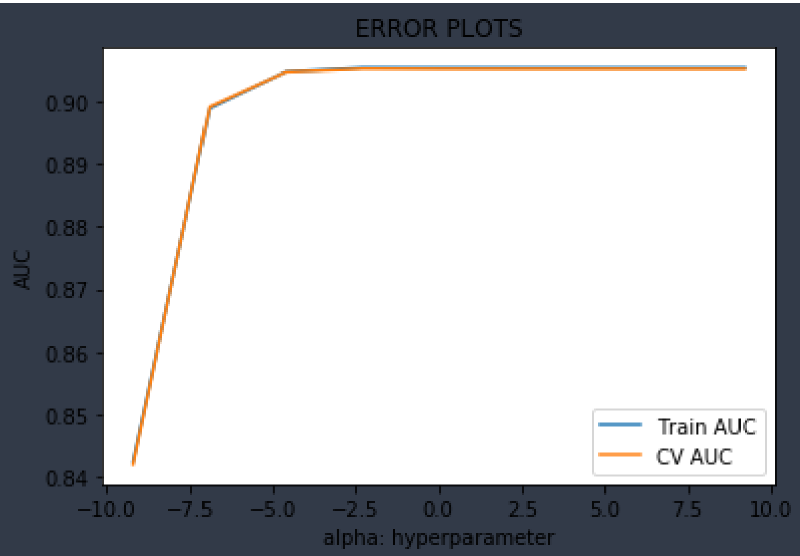
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b9dfb8978>
```



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

In [101]:
```
# Please write all the code with proper documentation
```

```
lamda= [10**(-4),10**(-3),10**(-2),10**(-1),1,10,100,1000,10000]
aw2v_val_accuracy = []
aw2v_train_accuracy = []
for i in lamda:
    model = LogisticRegression(C=i,penalty='l2')
    model.fit(sent_vectors,y_tr)
    val_data  = model.predict_log_proba(sent_vectors_cv)[:,1]
    train_data = model.predict_log_proba(sent_vectors)[:,1]
    aw2v_val_accuracy.append(roc_auc_score(np.asarray(y_cv),np.asarray(val_data)))
    aw2v_train_accuracy.append(roc_auc_score(np.asarray(y_tr),np.asarray(train_data)))
plt.plot(np.log(np.asarray(lamda)), aw2v_train_accuracy, label='Train AUC')
plt.plot(np.log(np.asarray(lamda)), aw2v_val_accuracy, label='CV AUC')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

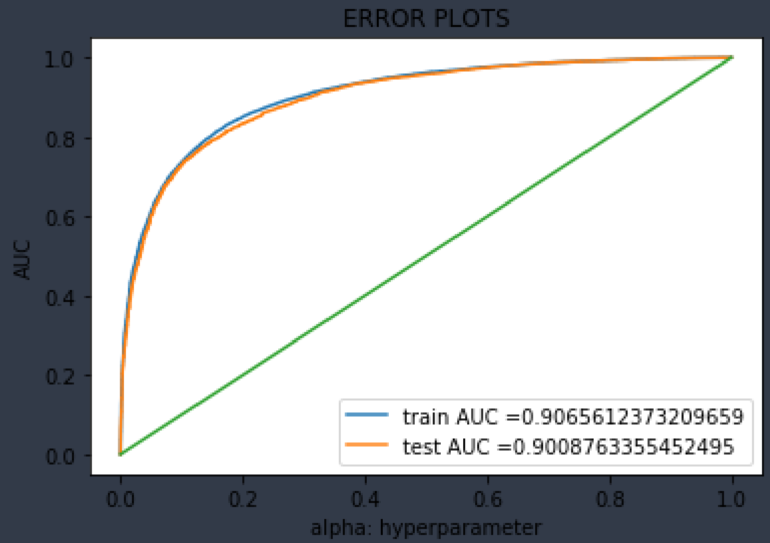


In [63]:
```
best_lambda =1
```

Tesitng on test data

In [64]:
```
model = LogisticRegression(C=best_lambda,penalty='l2')
model.fit(sent_vectors,y_tr)
test_pred  = model.predict_log_proba(sent_vectors_test)[:,1]
train_pred = model.predict_log_proba(sent_vectors)[:,1]
train_fpr, train_tpr, thresholds = roc_curve(y_tr, train_pred)
test_fpr, test_tpr, thresholds = roc_curve(y_test, test_pred)

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.plot([0.0,1.0],[0.0,1.0])
plt.legend()
plt.xlabel("alpha: hyperparameter")
```

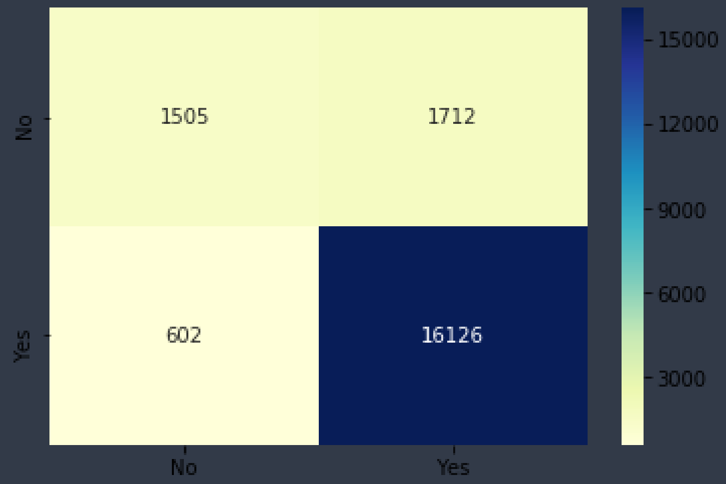
```python
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



Confusion matrix

In [65]:
```python
ytrain = model.predict(sent_vectors)
ytest = model.predict(sent_vectors_test)
ctrain = confusion_matrix(y_tr,ytrain)
ctest = confusion_matrix(y_test,ytest)
class_label=["No","Yes"]
df = pd.DataFrame(ctest, index=class_label, columns=class_label)
sns.heatmap(df, annot= True, fmt="d", cmap="YlGnBu")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b9e1dba90>
```



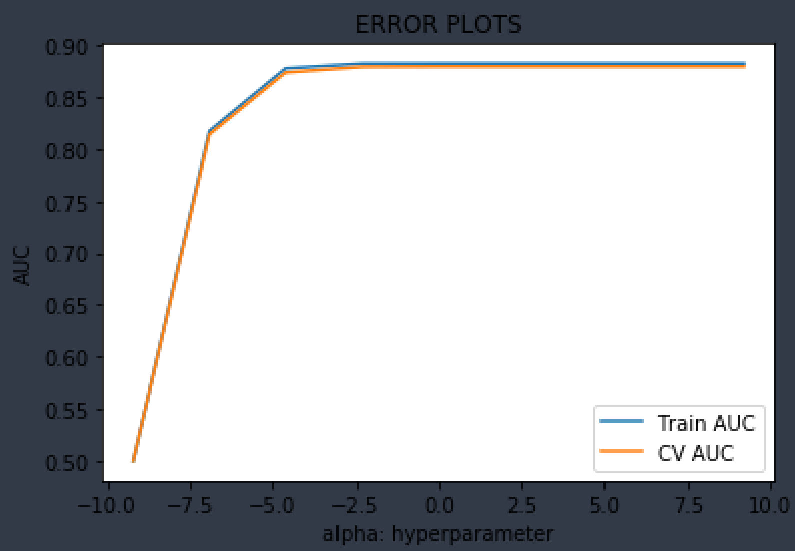
[5.4] Logistic Regression on TFIDF W2V, SET 4

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

In [105]:

```
# Please write all the code with proper documentation
lamda= [10**(-4),10**(-3),10**(-2),10**(-1),1,10,100,1000,10000]
tfidfw2v_val_accuracy = []
tfidfw2v_train_accuracy = []
for i in lamda:
    model = LogisticRegression(C=i,penalty='l1')
    model.fit(tfidf_sent_vectors,y_tr)
    val_data  = model.predict_log_proba(tfidf_sent_vectors_cv)[:,1]
    train_data = model.predict_log_proba(tfidf_sent_vectors)[:,1]
    tfidfw2v_val_accuracy.append(roc_auc_score(np.asarray(y_cv),np.asarray(val_data)))
    tfidfw2v_train_accuracy.append(roc_auc_score(np.asarray(y_tr),np.asarray(train_data
```

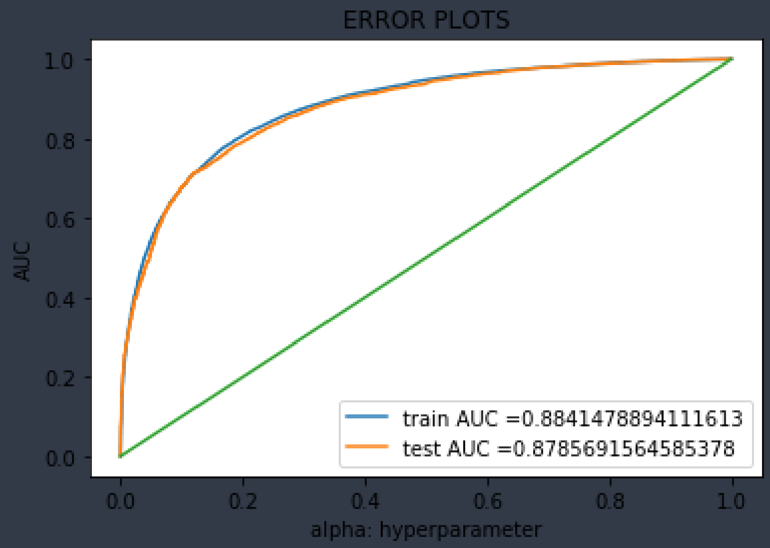
```
)))
plt.plot(np.log(np.asarray(lamda)), tfidfw2v_train_accuracy, label='Train AUC')
plt.plot(np.log(np.asarray(lamda)), tfidfw2v_val_accuracy, label='CV AUC')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



Testing on test data

In [66]:
```
model = LogisticRegression(C=best_lambda,penalty='l1')
model.fit(tfidf_sent_vectors,y_tr)
test_pred  = model.predict_log_proba(tfidf_sent_vectors_test)[:,1]
train_pred = model.predict_log_proba(tfidf_sent_vectors)[:,1]
train_fpr, train_tpr, thresholds = roc_curve(y_tr, train_pred)
```

```
test_fpr, test_tpr, thresholds = roc_curve(y_test, test_pred)

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.plot([0.0,1.0],[0.0,1.0])
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

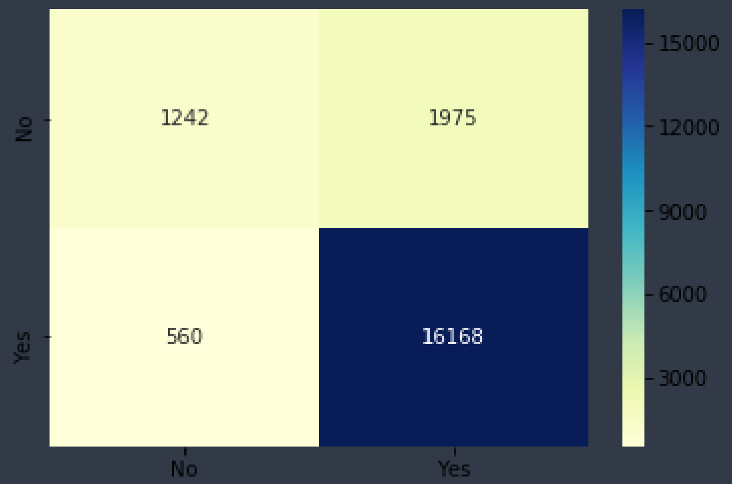


Confusion matrix

In [67]:
```
ytrain = model.predict(tfidf_sent_vectors)
ytest = model.predict(tfidf_sent_vectors_test)
ctrain = confusion_matrix(y_tr,ytrain)
```

```
ctest = confusion_matrix(y_test,ytest)
class_label=["No","Yes"]
df = pd.DataFrame(ctest, index=class_label, columns=class_label)
sns.heatmap(df, annot= True, fmt="d", cmap="YlGnBu")
```

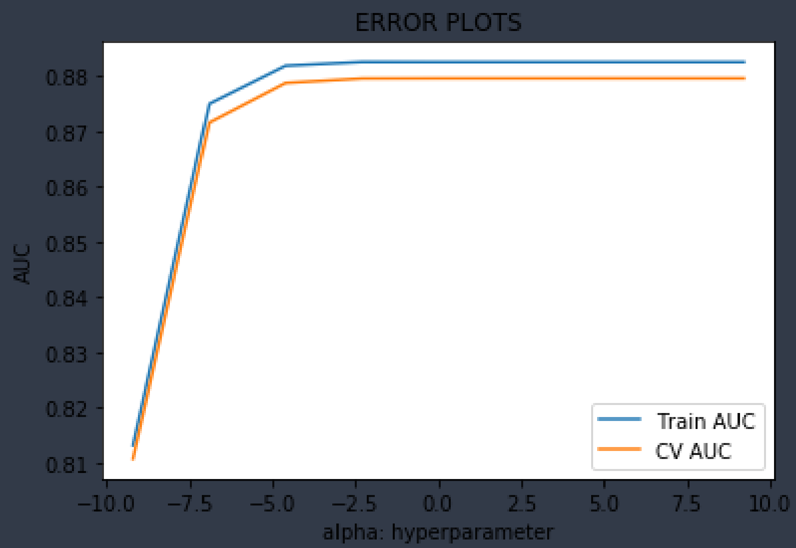
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b9e19c9b0>
```



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

In [108]:
```
# Please write all the code with proper documentation
lamda= [10**(-4),10**(-3),10**(-2),10**(-1),1,10,100,1000,10000]
tfidfw2v_val_accuracy = []
tfidfw2v_train_accuracy = []
for i in lamda:
    model = LogisticRegression(C=i,penalty='l2')
    model.fit(tfidf_sent_vectors,y_tr)
```

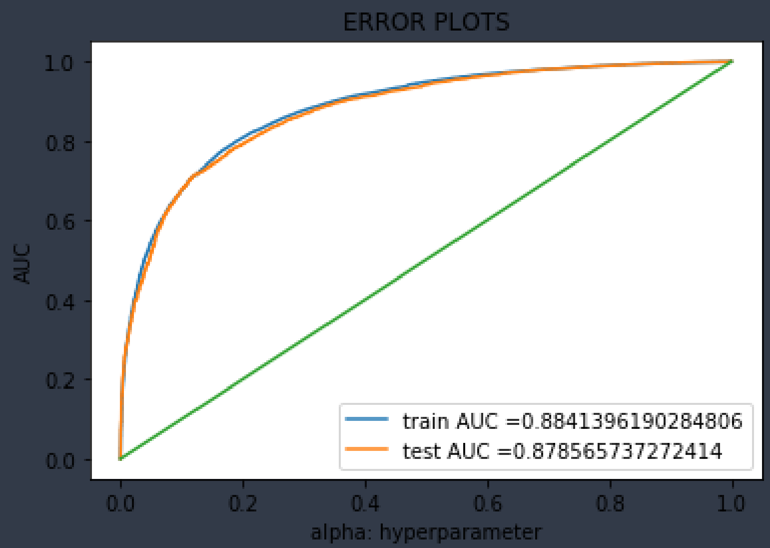
```
    val_data  = model.predict_log_proba(tfidf_sent_vectors_cv)[:,1]
    train_data = model.predict_log_proba(tfidf_sent_vectors)[:,1]
    tfidfw2v_val_accuracy.append(roc_auc_score(np.asarray(y_cv),np.asarray(val_data)))
    tfidfw2v_train_accuracy.append(roc_auc_score(np.asarray(y_tr),np.asarray(train_data
)))
plt.plot(np.log(np.asarray(lamda)), tfidfw2v_train_accuracy, label='Train AUC')
plt.plot(np.log(np.asarray(lamda)), tfidfw2v_val_accuracy, label='CV AUC')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [68]:
```
best_lambda = 1
```

Testing on test data

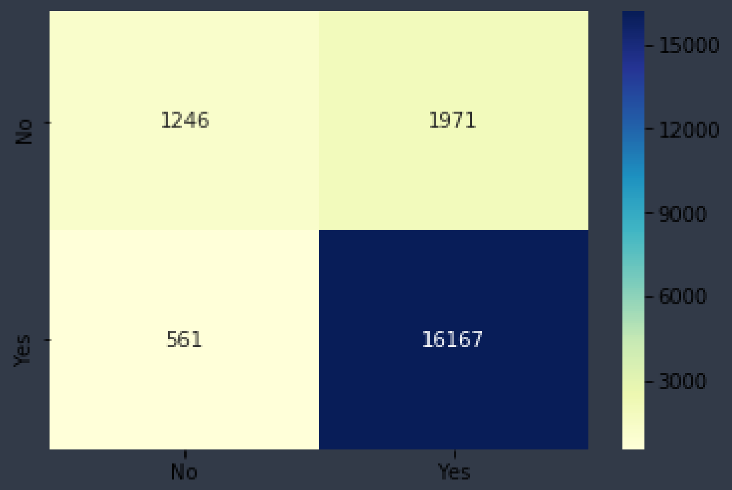
In [69]:
```
model = LogisticRegression(C=best_lambda,penalty='l2')
model.fit(tfidf_sent_vectors,y_tr)
test_pred  = model.predict_log_proba(tfidf_sent_vectors_test)[:,1]
train_pred = model.predict_log_proba(tfidf_sent_vectors)[:,1]
train_fpr, train_tpr, thresholds = roc_curve(y_tr, train_pred)
test_fpr, test_tpr, thresholds = roc_curve(y_test, test_pred)

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.plot([0.0,1.0],[0.0,1.0])
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



Confusion Matrix

In [70]:
```
ytrain = model.predict(tfidf_sent_vectors)
ytest = model.predict(tfidf_sent_vectors_test)
ctrain = confusion_matrix(y_tr,ytrain)
ctest = confusion_matrix(y_test,ytest)
class_label=["No","Yes"]
df = pd.DataFrame(ctest, index=class_label, columns=class_label)
sns.heatmap(df, annot= True, fmt="d", cmap="YlGnBu")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b9e2d6160>
```



[6] Conclusions

In [118]:
```
# Please compare all your models using Prettytable libraryfrom prettytable import Pretty
```

```
Table
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model","value of lambda","Train AUC","Test AUC"]

x.add_row(["BOW with l1 regularization",1,0.98,0.93])
x.add_row(["BOW with l2 regularization",0.1,0.94,0.93])
x.add_row(["TFIDF with l1 regularization",1,0.96,0.95])
x.add_row(["TFIDF with l2 regulariation",10,0.98,0.96])
x.add_row(["Avg_w2v with l1 regularization",1,0.90,0.90])
x.add_row(["Avg_w2v with l2 regularization",1,0.90,0.90])
x.add_row(["TFIDF_w2v with l1 regularization",1,0.88,0.87])
x.add_row(["TFIDF_w2v with l2 regularization",1,0.88,0.87])

print(x)
```

```
+----------------------------------+-----------------+-----------+----------+
|              Model               | value of lambda | Train AUC | Test AUC |
+----------------------------------+-----------------+-----------+----------+
|    BOW with l1 regularization    |        1        |    0.98   |   0.93   |
|    BOW with l2 regularization    |       0.1       |    0.94   |   0.93   |
|   TFIDF with l1 regularization   |        1        |    0.96   |   0.95   |
|   TFIDF with l2 regulariation    |        10       |    0.98   |   0.96   |
|  Avg_w2v with l1 regularization  |        1        |    0.9    |   0.9    |
|  Avg_w2v with l2 regularization  |        1        |    0.9    |   0.9    |
| TFIDF_w2v with l1 regularization |        1        |    0.88   |   0.87   |
| TFIDF_w2v with l2 regularization |        1        |    0.88   |   0.87   |
+----------------------------------+-----------------+-----------+----------+
```

In [ ]:
```
""" Here we can see that for tfidf l2 regularized model we get the heighest
    Test AUC of 0.96 with lambda = 10"""
```