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 [ ]:
```
from collections import defaultdict
final1 = defaultdict(list)

f=open('1.txt')
for line in f:
    final1['label'].append(int(line[-2]))
    final1['Text'].append(line[:-3])
```

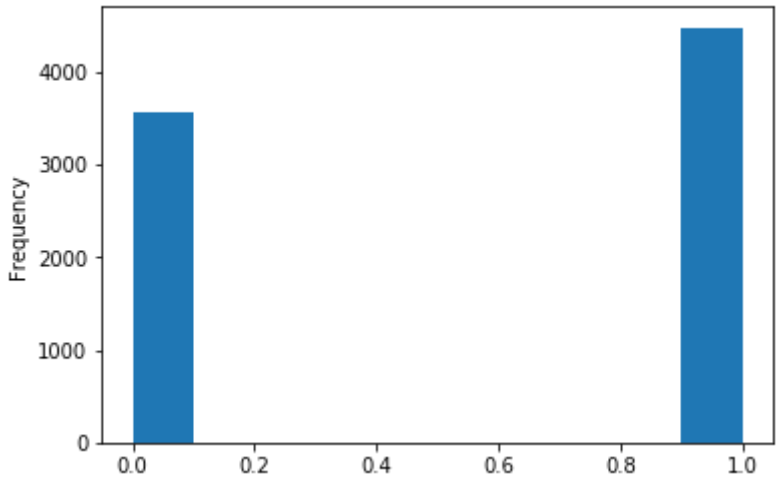
In [3]:
```
label2 =[]
text2=[]
f=open('2.txt')
for line in f:
    final1['label'].append(int(line[0]))
    final1['Text'].append(line[2:])

final =  pd.DataFrame(final1)
```

In [4]:
```
final.head()
```

Out[4]:

	Text	label
0	A very, very, very slow-moving, aimless movie ...	0
1	Not sure who was more lost - the flat characte...	0
2	Attempting artiness with black & white and cle...	0
3	Very little music or anything to speak of.	0
4	The best scene in the movie was when Gerardo i...	1

In [5]:
```
#data set is balance or not
final['label'].astype(int).plot.hist();
```



In [6]:
```
# 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)
```

```
A very, very, very slow-moving, aimless movie about a distressed, drifting young man.
==================================================
the da vinci code is awesome!

==================================================
Love luv lubb the Da Vinci Code!

==================================================
Brokeback Mountain was so awesome.

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

In [7]:
```
# 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)
```

```
A very, very, very slow-moving, aimless movie about a distressed, drifting young man.
```

In [8]:
```
# 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)
```

```
A very, very, very slow-moving, aimless movie about a distressed, drifting young man.
==================================================
the da vinci code is awesome!

==================================================
Love luv lubb the Da Vinci Code!

==================================================
Brokeback Mountain was so awesome.
```

In [9]:
```
# 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 [10]:
```
sent_4900 = decontracted(sent_4900)
print(sent_4900)
print("="*50)
```

```
Brokeback Mountain was so awesome.

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

In [11]:
```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
A very, very, very slow-moving, aimless movie about a distressed, drifting young man.
```

In [12]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_0 = re.sub('[^A-Za-z0-9]+', ' ', sent_0)
print(sent_0)
```

```
A very very very slow moving aimless movie about a distressed drifting young man
```

In [13]:
```
# 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', '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', 'furthe
r',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'mor
e',\
            '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', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "were
n't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [14]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed = []
# 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.append(sentance.strip())
```

```
100%|██████████| 8038/8038 [00:02<00:00, 3858.57it/s]
```

In [15]:
```
preprocessed[1500]
```

Out[15]:
```
'love luv lubb da vinci code'
```

Featurization

In [16]:
```
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(preprocessed)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

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

```
some feature names  ['aailiyah', 'aaron', 'abandoned', 'ability', 'able', 'abortion', 'abrams', 'abroad', 'absolute',
'absolutely']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (8038, 3975)
the number of unique words  3975
```

In [17]:
```
#tfidf
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names()[0:10])
print('='*50)

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

```
some sample features(unique words in the corpus) ['absolutely', 'absolutely awesome', 'absolutely love', 'acceptabl
e', 'aching', 'aching cock', 'acne', 'acne love', 'acting', 'action']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (8038, 655)
the number of unique words including both unigrams and bigrams  655
```

Word2Vec

In [18]:
```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
for sentance in preprocessed:
    list_of_sentance.append(sentance.split())
```

In [19]:
```
# Using Google News Word2Vectors

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


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

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

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

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

```
[('homosexuality', 0.9947419166564941), ('either', 0.9937832951545715), ('becoming', 0.9877654314041138), ('acceptabl
e', 0.9784659743309021), ('think', 0.9190821051597595), ('brokeback', 0.8539977669715881), ('mountain', 0.84261256456
37512), ('beautiful', 0.8298031091690063), ('horrible', 0.8289213180541992), ('anyway', 0.8279935121536255)]
==================================================
[('every', 0.9977586269378662), ('performance', 0.9976140260696411), ('little', 0.9973092675209045), ('john', 0.99726
4564037323), ('half', 0.9972462058067322), ('kids', 0.9971147775650024), ('playing', 0.9970337748527527), ('end', 0.9
968582987785339), ('family', 0.9968501925468445), ('come', 0.9968061447143555)]
```

In [20]:
```
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  600
sample words  ['real', 'girl', 'suspense', 'wotshisface', 'begin', 'impossible', 'racism', 'make', 'stand', 'week',
'solid', 'nothing', 'kids', 'full', 'talking', 'lacks', 'iii', 'whimpering', 'felt', 'likes', 'hell', 'differently',
'together', 'friday', 'mission', 'keys', 'demons', 'mtv', 'guess', 'blonds', 'interesting', 'half', 'awards', 'consid
er', 'mean', 'art', 'john', 'place', 'character', 'came', 'piece', 'no', 'looking', 'mention', 'fan', 'case', 'kind
a', 'turned', 'day', 'hoot']
```

In [21]:
```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300 if you use g
oogle's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|██████████| 8038/8038 [00:00<00:00, 13296.18it/s]

8038
50
```

In [22]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(preprocessed)
# 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_)))

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

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

```
100%|██████████| 8038/8038 [00:02<00:00, 2965.50it/s]
```

split data into train and test for BoW

In [23]:
```
# Please write all the code with proper documentation
# Please write all the code with proper documentation
# Please write all the code with proper documentation
X = preprocessed
Y = final['label']
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
from sklearn.model_selection import train_test_split

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=Flase)# this is for time series sp
lit
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2) # this is random splitting
#X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33) # this is random splitting

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

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

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

NaiveBayes

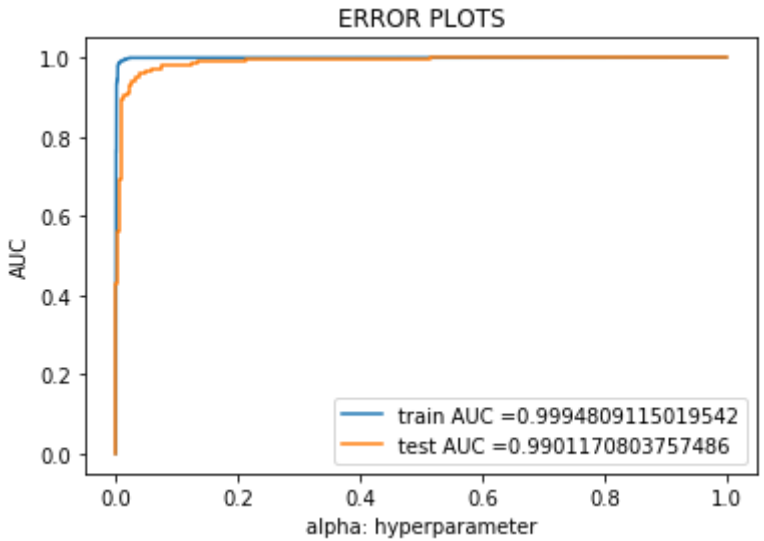
In [25]:
```
import seaborn as sns
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
#import GridSearchCV
from sklearn.model_selection import GridSearchCV
param_grid ={"alpha" : [ 10**-4, 10**-3, 10**-2, 10**-1, 10**-0, 10**1, 10**2, 10**3, 10**4]}
clf = MultinomialNB(class_prior=None)
#clf = RandomForestClassifier(random_state=0)
clf_cv_NB_BoW= GridSearchCV(clf,param_grid,cv=5)
clf_cv_NB_BoW.fit(X_train_bow,y_train)
```

Out[25]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=MultinomialNB(alpha=1.0, class_prior=None,
                                     fit_prior=True),
             iid='warn', n_jobs=None,
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000,
                                   10000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [26]:
```
best_alpha = clf_cv_NB_BoW.best_params_
best_alpha
```

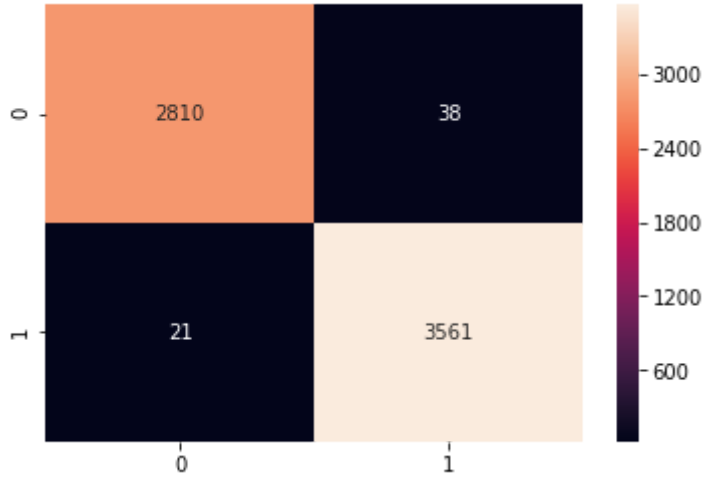
Out[26]:
```
{'alpha': 0.1}
```

In [27]:
```
#Testing with Test data
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

best_alpha = 0.1
clf_NB_BoW = MultinomialNB(alpha=best_alpha, class_prior=None)
clf_NB_BoW.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, clf_NB_BoW.predict_proba(X_train_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, clf_NB_BoW.predict_proba(X_test_bow)[:,1])

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

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
uniform_data = confusion_matrix(y_train, clf_NB_BoW.predict(X_train_bow))
ax = sns.heatmap(uniform_data,annot= True, fmt= "d")
```



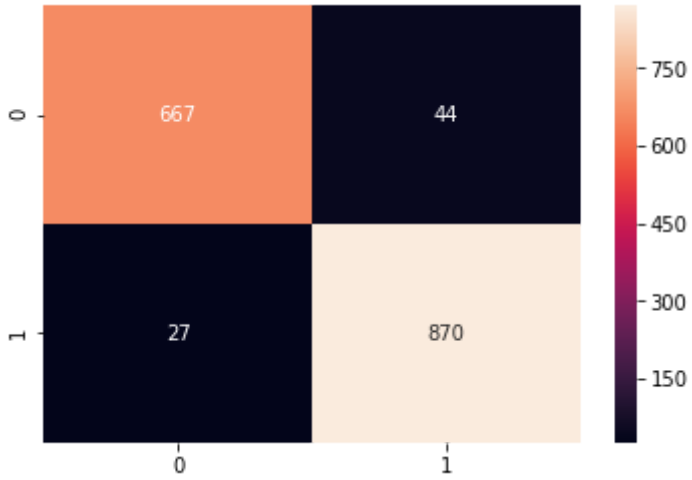
==========

Train confusion matrix



In [28]:
```
print("Test confusion matrix")
uniform_data = confusion_matrix(y_test, clf_NB_BoW.predict(X_test_bow))
ax = sns.heatmap(uniform_data,annot= True, fmt= "d")
```

```
Test confusion matrix
```



Logistic Regression

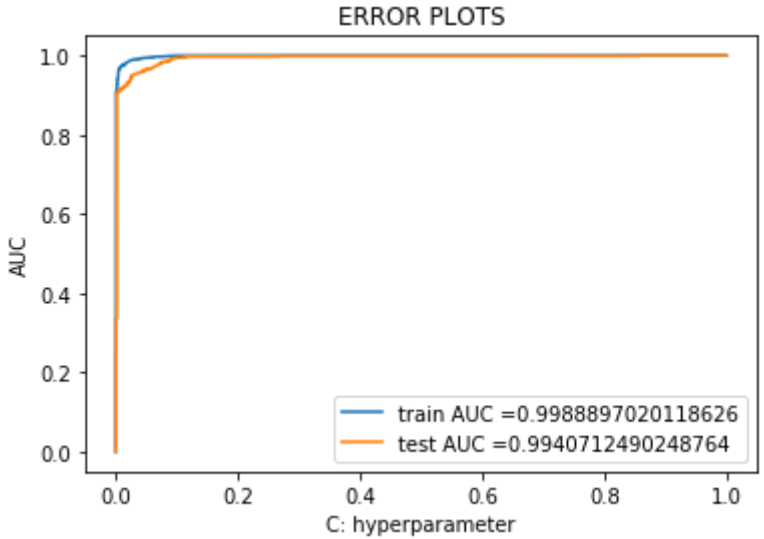
In [28]:
```
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
#import GridSearchCV
from sklearn.model_selection import GridSearchCV
param_grid ={"C" : [1, 0.1, 0.01, 0.001, 0.0001]}
clf = LogisticRegression(penalty='l1', random_state=0, solver='saga',multi_class='multinomial')
#clf = RandomForestClassifier(random_state=0)
clf_cv_LR_BoW= GridSearchCV(clf,param_grid,cv=5)
clf_cv_LR_BoW.fit(X_train_bow,y_train)
```

Out[28]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100,
                                          multi_class='multinomial',
                                          n_jobs=None, penalty='l1',
                                          random_state=0, solver='saga',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'C': [1, 0.1, 0.01, 0.001, 0.0001]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [30]:
```
clf_cv_LR_BoW.best_params_
```

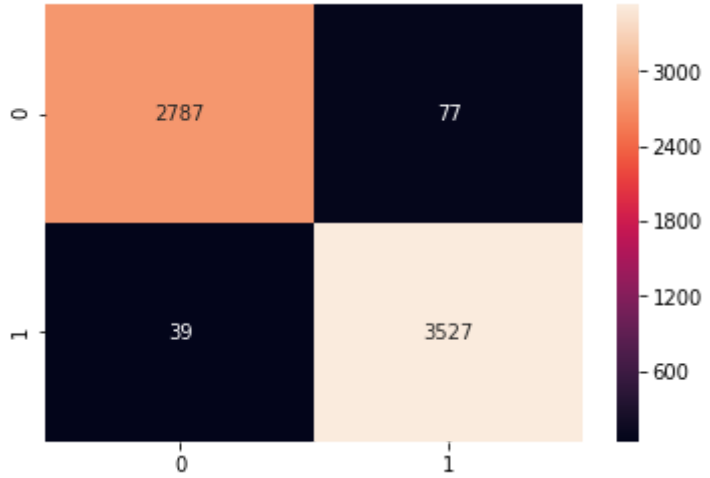
Out[30]:
```
{'C': 1}
```

In [31]:
```
#Testing with Test data
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc

best_C = 1
clf_LR_BoW = LogisticRegression(penalty='l1', C=best_C, random_state=0, solver='saga',multi_class='multinomial')
clf_LR_BoW.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, clf_LR_BoW.predict_proba(X_train_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, clf_LR_BoW.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("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
uniform_data = confusion_matrix(y_train, clf_LR_BoW.predict(X_train_bow))
ax = sns.heatmap(uniform_data,annot= True, fmt= "d")
```



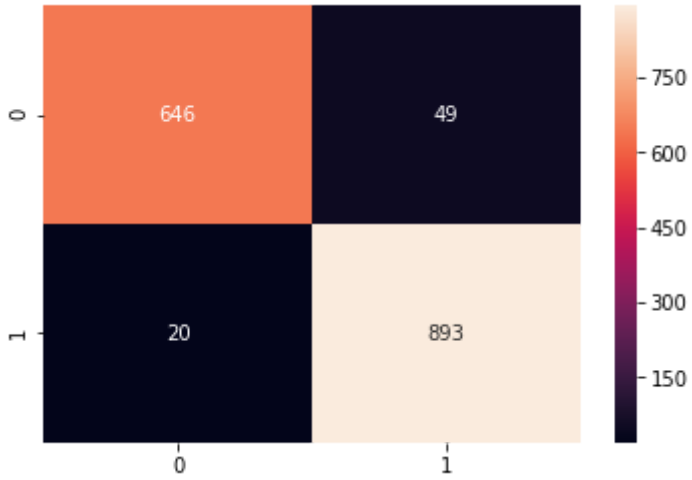
==========

Train confusion matrix



In [32]:
```
print("Test confusion matrix")
uniform_data = confusion_matrix(y_test, clf_LR_BoW.predict(X_test_bow))
ax = sns.heatmap(uniform_data,annot= True, fmt= "d")
```

```
Test confusion matrix
```

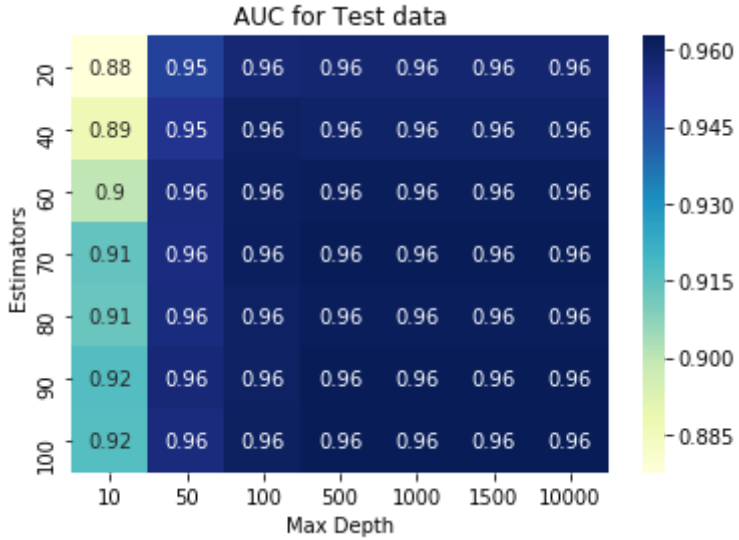


RandomForestClassifier

In [33]:
```
from sklearn.ensemble import RandomForestClassifier
#import GridSearchCV
from sklearn.model_selection import GridSearchCV
param_grid = {'max_depth': [10, 50, 100, 500, 1000, 1500, 10000], 'n_estimators': [20, 40, 60, 70, 80, 90, 100]}
clf = RandomForestClassifier(random_state=0)
clf_cv_RF_BoW= GridSearchCV(clf,param_grid,cv=5)
clf_cv_RF_BoW.fit(X_train_bow,y_train)
```

Out[33]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators='warn', n_jobs=None,
                                              oob_score=False, random_state=0,
                                              verbose=0, warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'max_depth': [10, 50, 100, 500, 1000, 1500, 10000],
                         'n_estimators': [20, 40, 60, 70, 80, 90, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

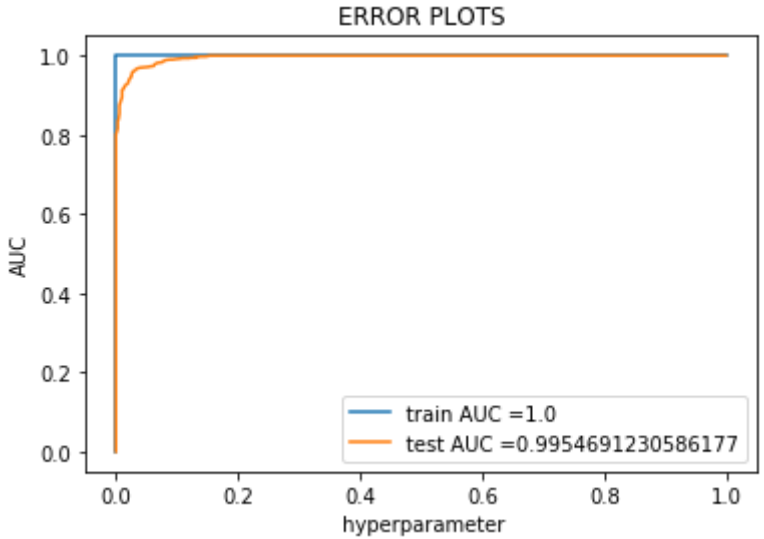
In [34]:
```
max_depth_list = list(clf_cv_RF_BoW.cv_results_['param_max_depth'].data)
estimators_list = list(clf_cv_RF_BoW.cv_results_['param_n_estimators'].data)
data = pd.DataFrame(data={'Estimators':estimators_list, 'Max Depth':max_depth_list, 'AUC':clf_cv_RF_BoW.cv_results_['mean_test_score']})
data = data.pivot(index='Estimators', columns='Max Depth', values='AUC')
sns.heatmap(data, annot=True, cmap="YlGnBu").set_title('AUC for Test data')
plt.show()
```



In [35]:
```
clf_cv_RF_BoW.best_params_
```

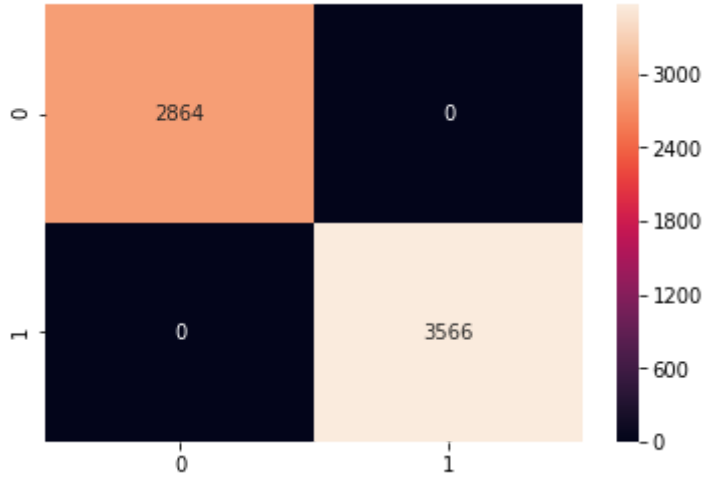
Out[35]:
```
{'max_depth': 500, 'n_estimators': 70}
```

In [36]:
```
#Testing with Test data
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, auc
clf_RF_BoW = RandomForestClassifier(random_state=0,max_depth=500, n_estimators=70)
clf_RF_BoW.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, clf_RF_BoW.predict_proba(X_train_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, clf_RF_BoW.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(" hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
uniform_data = confusion_matrix(y_train, clf_RF_BoW.predict(X_train_bow))
ax = sns.heatmap(uniform_data,annot= True, fmt= "d")
```



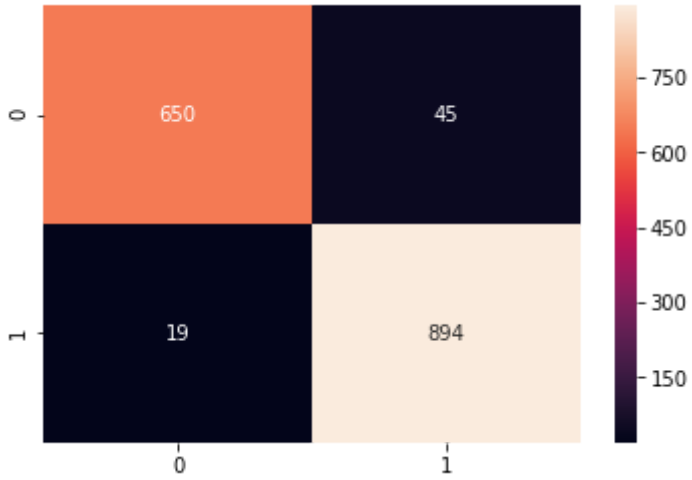
==========

Train confusion matrix



In [37]:
```
print("Test confusion matrix")
uniform_data = confusion_matrix(y_test, clf_RF_BoW.predict(X_test_bow))
ax = sns.heatmap(uniform_data,annot= True, fmt= "d")
```

```
Test confusion matrix
```



split data into train and test for tfidf

In [38]:
```
# Please write all the code with proper documentation
X = preprocessed
Y = final['label']
# Please write all the code with proper documentation
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
from sklearn.model_selection import train_test_split

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=Flase)# this is for time series sp
lit
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2) # this is random splitting
#X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33) # this is random splitting

from sklearn.feature_extraction.text import CountVectorizer
vectorizer = TfidfVectorizer(ngram_range=(1,2), min_df=10)#in scikit-learn
vectorizer.fit(X_train) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_TFIDF = vectorizer.transform(X_train)
#X_cv_TFIDF = vectorizer.transform(X_cv)
X_test_TFIDF = vectorizer.transform(X_test)
```

NaiveBayes

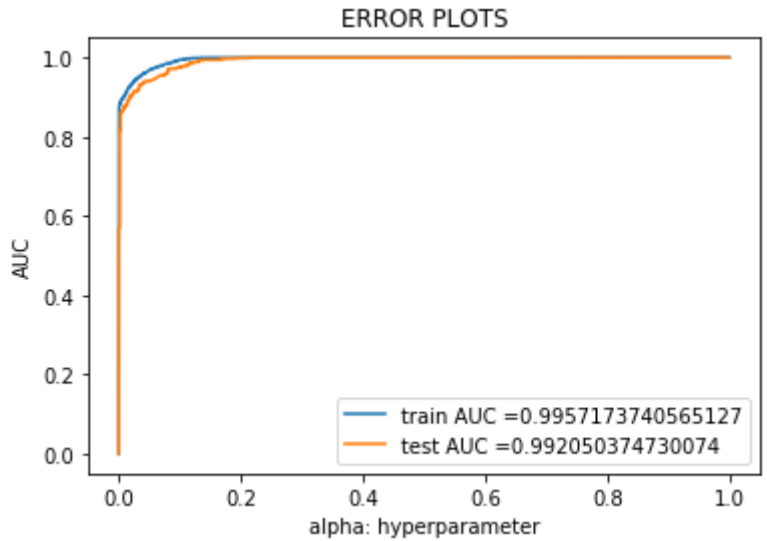
In [39]:
```
import seaborn as sns
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
#import GridSearchCV
from sklearn.model_selection import GridSearchCV
param_grid ={"alpha" : [ 10**-4, 10**-3, 10**-2, 10**-1, 10**-0, 10**1, 10**2, 10**3, 10**4]}
clf = MultinomialNB(class_prior=None)
#clf = RandomForestClassifier(random_state=0)
clf_cv_NB_tfidf= GridSearchCV(clf,param_grid,cv=5)
clf_cv_NB_tfidf.fit(X_train_TFIDF,y_train)
```

Out[39]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=MultinomialNB(alpha=1.0, class_prior=None,
                                     fit_prior=True),
             iid='warn', n_jobs=None,
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000,
                                   10000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [40]:
```
clf_cv_NB_tfidf.best_params_
```

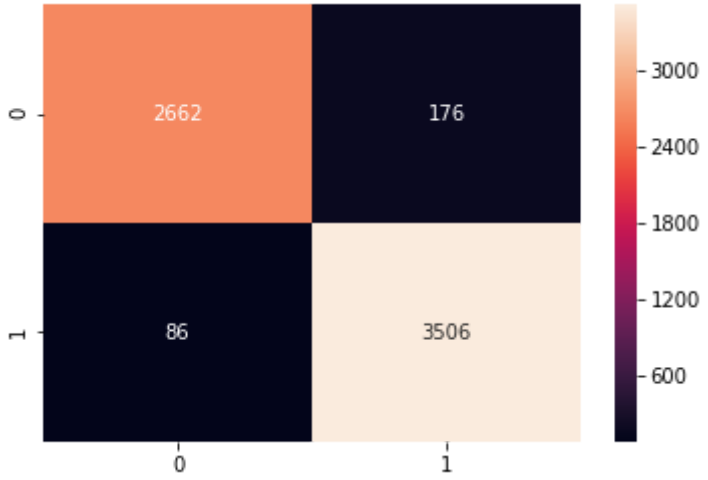
Out[40]:
```
{'alpha': 0.0001}
```

In [41]:
```
from sklearn.metrics import roc_curve, auc

best_alpha = 0.0001
clf_nb_tfidf = MultinomialNB(alpha=best_alpha, class_prior=None)
clf_nb_tfidf.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, clf_nb_tfidf.predict_proba(X_train_TFIDF)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, clf_nb_tfidf.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("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
uniform_data = confusion_matrix(y_train, clf_nb_tfidf.predict(X_train_TFIDF))
ax = sns.heatmap(uniform_data,annot= True, fmt= "d")
```



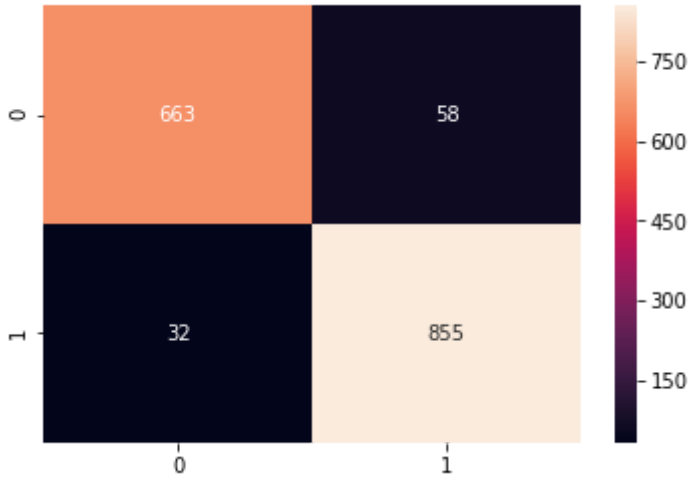
==========

Train confusion matrix



In [42]:
```
print("Test confusion matrix")
uniform_data = confusion_matrix(y_test, clf_nb_tfidf.predict(X_test_TFIDF))
ax = sns.heatmap(uniform_data,annot= True, fmt= "d")
```

```
Test confusion matrix
```



Logistic Regression

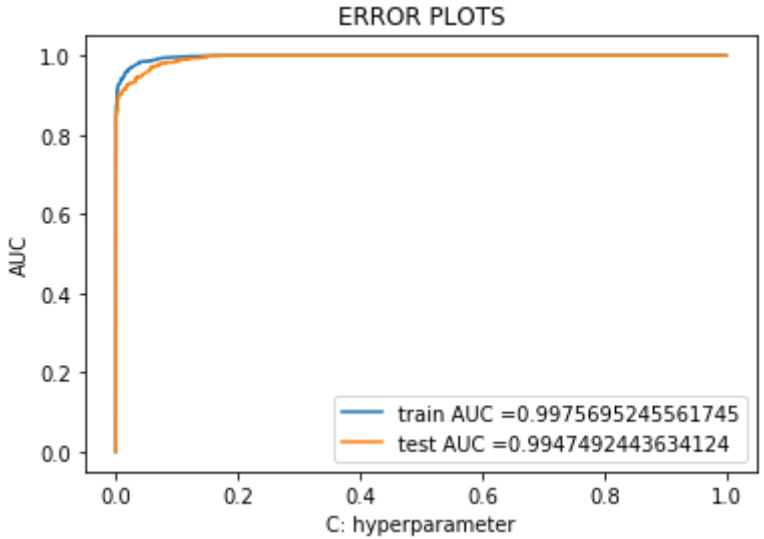
In [43]:
```
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
#import GridSearchCV
from sklearn.model_selection import GridSearchCV
param_grid ={"C" : [1, 0.1, 0.01, 0.001, 0.0001]}
clf = LogisticRegression(penalty='l1', random_state=0, solver='saga',multi_class='multinomial')
#clf = RandomForestClassifier(random_state=0)
clf_cv_LR_tfidf= GridSearchCV(clf,param_grid,cv=5)
clf_cv_LR_tfidf.fit(X_train_TFIDF,y_train)
```

Out[43]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100,
                                          multi_class='multinomial',
                                          n_jobs=None, penalty='l1',
                                          random_state=0, solver='saga',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'C': [1, 0.1, 0.01, 0.001, 0.0001]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [44]:
```
clf_cv_LR_tfidf.best_params_
```

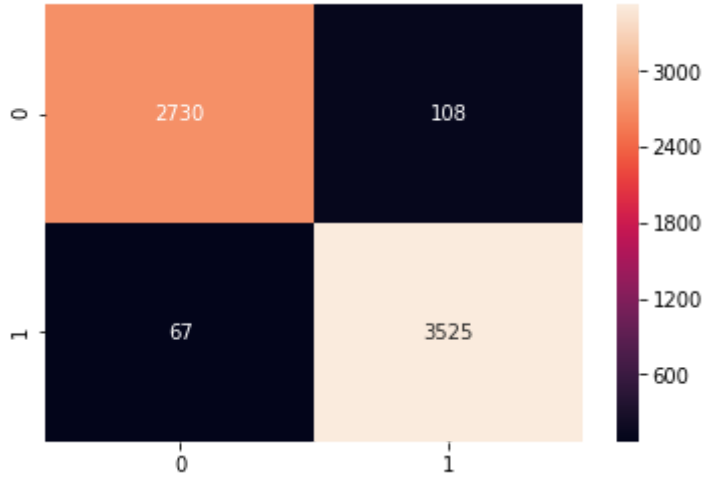
Out[44]:
```
{'C': 1}
```

In [45]:
```
#Testing with Test data
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

best_C = 1
clf_LR_tfidf = LogisticRegression(penalty='12', C=best_C, random_state=0, solver='saga',multi_class='multinomial')
clf_LR_tfidf.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, clf_LR_tfidf.predict_proba(X_train_TFIDF)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, clf_LR_tfidf.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("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
uniform_data = confusion_matrix(y_train, clf_LR_tfidf.predict(X_train_TFIDF))
ax = sns.heatmap(uniform_data,annot= True, fmt= "d")
```



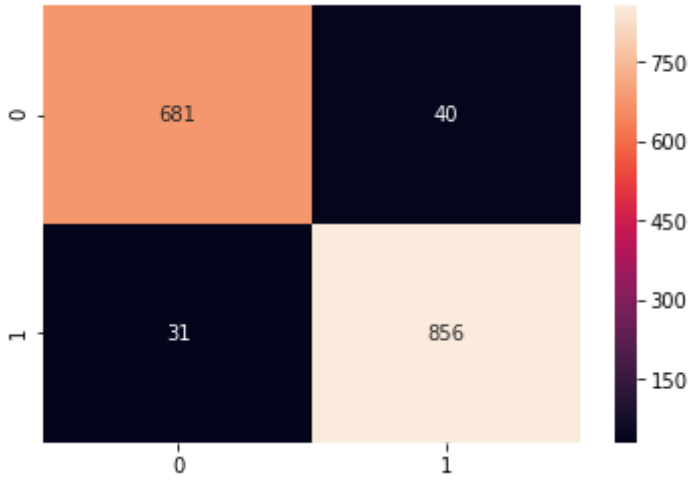
==========

Train confusion matrix



In [46]:
```
print("Test confusion matrix")
uniform_data = confusion_matrix(y_test, clf_LR_tfidf.predict(X_test_TFIDF))
ax = sns.heatmap(uniform_data,annot= True, fmt= "d")
```

```
Test confusion matrix
```

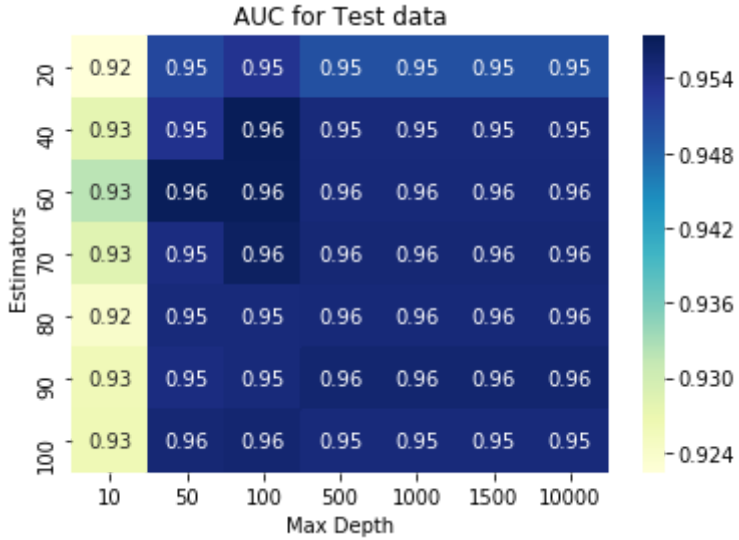


Random Forest Classifier

In [47]:
```
from sklearn.ensemble import RandomForestClassifier
#import GridSearchCV
from sklearn.model_selection import GridSearchCV
param_grid = {'max_depth': [10, 50, 100, 500, 1000, 1500, 10000], 'n_estimators': [20, 40, 60, 70, 80, 90, 100]}
clf = RandomForestClassifier(random_state=0)
clf_cv_RF_tfidf= GridSearchCV(clf,param_grid,cv=5)
clf_cv_RF_tfidf.fit(X_train_TFIDF,y_train)
```

Out[47]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators='warn', n_jobs=None,
                                              oob_score=False, random_state=0,
                                              verbose=0, warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'max_depth': [10, 50, 100, 500, 1000, 1500, 10000],
                         'n_estimators': [20, 40, 60, 70, 80, 90, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

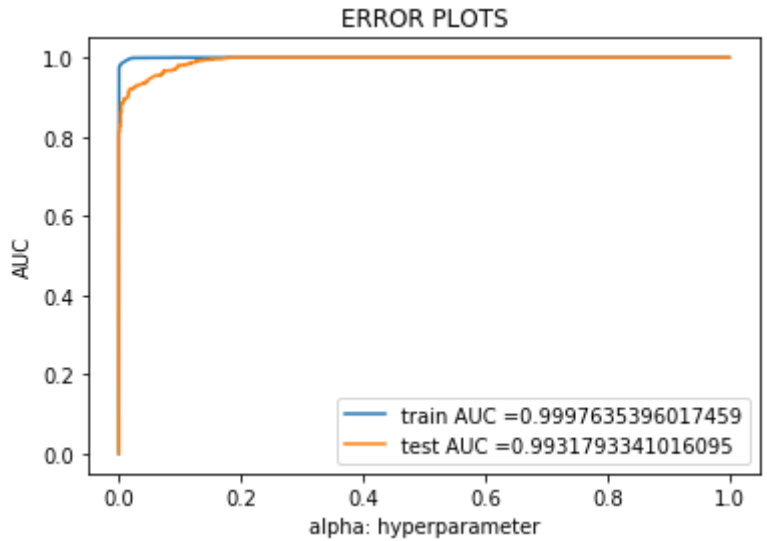
In [48]:
```
max_depth_list = list(clf_cv_RF_tfidf.cv_results_['param_max_depth'].data)
estimators_list = list(clf_cv_RF_tfidf.cv_results_['param_n_estimators'].data)
data = pd.DataFrame(data={'Estimators':estimators_list, 'Max Depth':max_depth_list, 'AUC':clf_cv_RF_tfidf.cv_results_[
'mean_test_score']})
data = data.pivot(index='Estimators', columns='Max Depth', values='AUC')
sns.heatmap(data, annot=True, cmap="YlGnBu").set_title('AUC for Test data')
plt.show()
```



In [49]:
```
clf_cv_RF_tfidf.best_params_
```

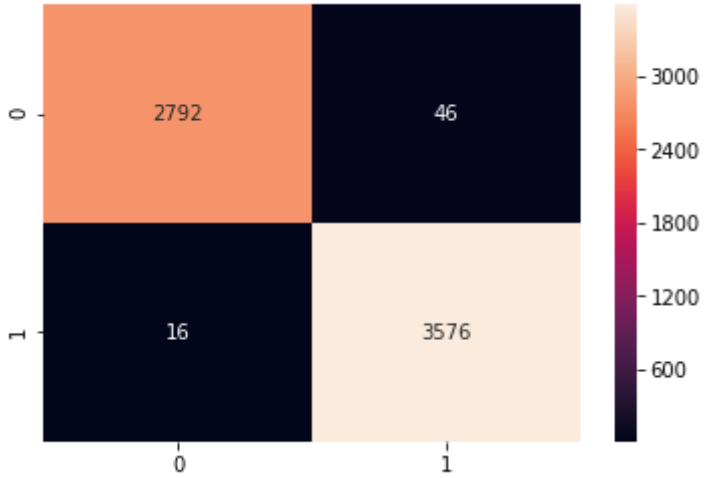
Out[49]:
```
{'max_depth': 100, 'n_estimators': 40}
```

In [50]:
```
#Testing with Test data
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

clf_RF_tfidf = RandomForestClassifier(random_state=0,max_depth=100, n_estimators=40)
clf_RF_tfidf.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, clf_RF_tfidf.predict_proba(X_train_TFIDF)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, clf_RF_tfidf.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("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
uniform_data = confusion_matrix(y_train, clf_RF_tfidf.predict(X_train_TFIDF))
ax = sns.heatmap(uniform_data,annot= True, fmt= "d")
```



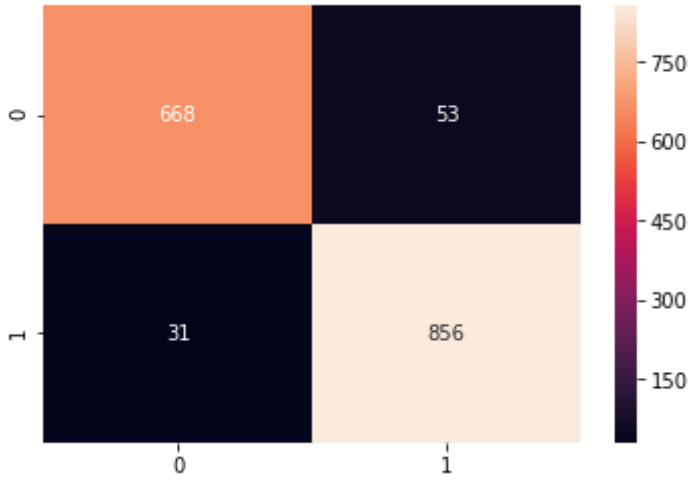
==========

Train confusion matrix



In [51]:
```
print("Test confusion matrix")
uniform_data = confusion_matrix(y_test, clf_RF_tfidf.predict(X_test_TFIDF))
ax = sns.heatmap(uniform_data,annot= True, fmt= "d")
```

```
Test confusion matrix
```



Applying Random Forests on AVG W2V

In [52]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
from sklearn.model_selection import train_test_split

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=Flase)# this is for time series sp
lit
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2) # this is random splitting
#X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33) # this is random splitting

#Preparing Reviews for gensim model
i=0
list_of_sentance_train=[]
for sentance in X_train:
    list_of_sentance_train.append(sentance.split())
 #Training w2v model

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

# this line of code trains your w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=4)

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  505
sample words  ['ok', 'mother', 'talking', 'given', 'coming', 'fact', 'absolutely', 'mom', 'gary', 'budget', 'perfec
t', 'solid', 'word', 'better', 'becoming', 'highly', 'experience', 'made', 'give', 'wanted', 'mind', 'dragged', 'tw
o', 'scenes', 'opinion', 'pretty', 'often', 'turned', 'told', 'insanely', 'might', 'moving', 'not', 'knows', 'see',
'dialogue', 'recommended', 'seen', 'wotshisface', 'racism', 'oscar', 'work', 'keys', 'theater', 'loved', 'songs', 're
ad', 'begin', 'community', 'acne']
```

In [53]:
```
#Converting Reviews into Numerical Vectors using W2V vectors

#Algorithm: Avg W2V

from tqdm import tqdm
import numpy as np

#Converting Train data text

# average Word2Vec
# compute average word2vec for each review.
sent_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance_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 g
oogle'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_train.append(sent_vec)
sent_vectors_train = np.array(sent_vectors_train)
print(sent_vectors_train.shape)
print(sent_vectors_train[0])
```

```
100%|██████████| 6430/6430 [00:00<00:00, 15826.07it/s]

(6430, 50)
[ 4.31584986e-03  4.82470654e-02 -1.10908598e-03 -7.52001815e-03
 -1.00640301e-03 -9.97795686e-02 -3.07208318e-02 -1.02930270e-01
 -1.17968127e-01  1.62688885e-02  1.97248794e-02  9.66700315e-02
  5.76948747e-02 -2.53514517e-02  1.04601741e-01  9.25860927e-02
 -5.42466491e-02  7.32812285e-02 -6.78649731e-03  2.09226385e-02
 -9.02480260e-02 -5.02421074e-02  4.78475774e-03  1.17753170e-01
  1.23303406e-01  3.62860672e-02 -1.07852608e-01 -7.30782598e-02
  1.18716873e-01  2.31619421e-02 -2.00667661e-02  1.13027997e-01
  1.70237347e-02  2.52799392e-02  6.48770258e-02  1.17845029e-01
  3.07397321e-02 -2.33457815e-02 -2.67884601e-02 -1.84447155e-03
  1.36103434e-02  1.36160985e-01  4.43706736e-02  1.14947960e-01
 -1.07768318e-02 -1.20386267e-02 -2.87235808e-02  6.01061038e-05
  1.30376086e-01 -1.23426318e-02]
```

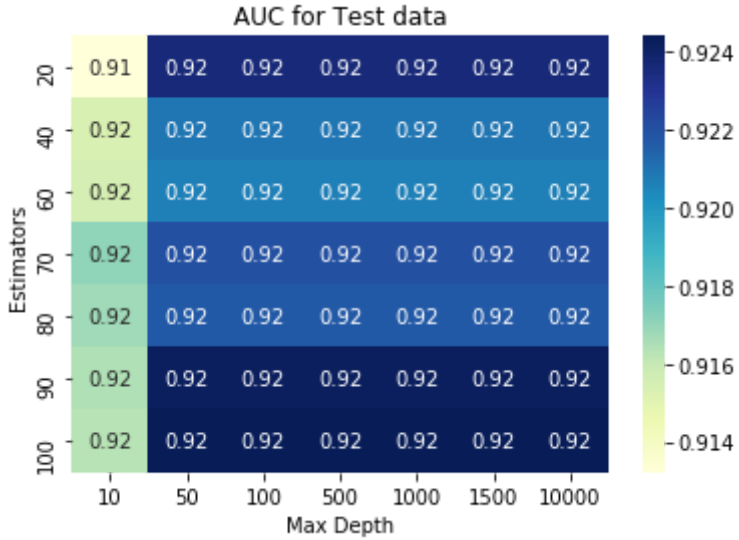
In [54]:
```
#Converting Test data text
i=0
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sent 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 change this to 300 if you use g
oogle'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)
sent_vectors_test = np.array(sent_vectors_test)
print(sent_vectors_test.shape)
print(sent_vectors_test[0])
```

```
100%|██████████| 1608/1608 [00:00<00:00, 14659.12it/s]

(1608, 50)
[ 0.30674833  0.13991447  0.03827324  0.23929965  0.13574131 -0.22908914
 -0.51812851 -0.63726533  0.17094047 -0.19487083  0.08865481  0.74904361
  0.09410995 -0.48561508  0.52406213  0.47606938 -0.24679037  0.05479717
  0.03344516  0.1722329  -0.76238429 -0.02975634  0.26907431  0.90394043
  0.97730556  0.43129858 -0.3363911  -0.16615126  0.56679229  0.18027803
  0.46246117  0.62612372 -0.08286493  0.37897206  0.39206419  0.46528977
  0.2052499  -0.34322818  0.38745052  0.10768376 -0.26053902  0.04791026
  0.21844842  0.4488855  -0.15636532 -0.14415916 -0.12990334 -0.02839463
  0.53074998 -0.10725935]
```

In [55]:
```
from sklearn.ensemble import RandomForestClassifier
#import GridSearchCV
from sklearn.model_selection import GridSearchCV
param_grid = {'max_depth': [10, 50, 100, 500, 1000, 1500, 10000], 'n_estimators': [20, 40, 60, 70, 80, 90, 100]}
clf = RandomForestClassifier(random_state=0)
clf_cv_RF_w2v= GridSearchCV(clf,param_grid,cv=5)
clf_cv_RF_w2v.fit(sent_vectors_train,y_train)
#clf = DecisionTreeClassifier(random_state=0,max_depth=None, min_samples_split=2)
```

Out[55]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators='warn', n_jobs=None,
                                              oob_score=False, random_state=0,
                                              verbose=0, warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'max_depth': [10, 50, 100, 500, 1000, 1500, 10000],
                         'n_estimators': [20, 40, 60, 70, 80, 90, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

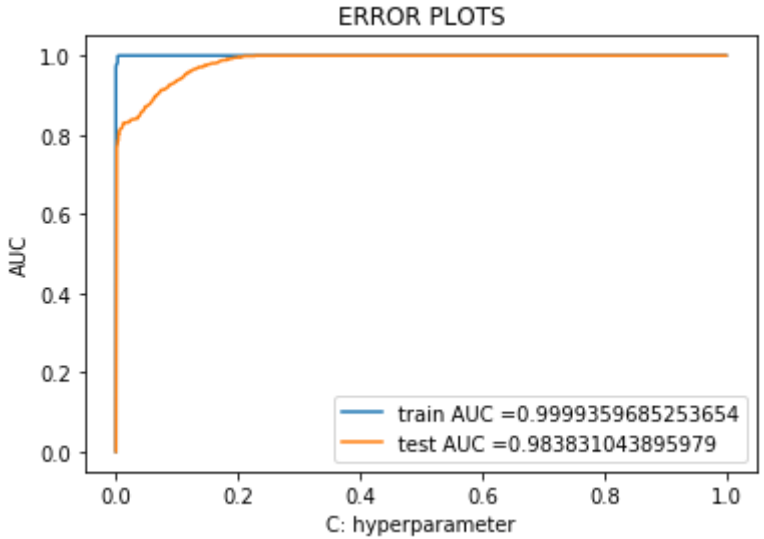
In [56]:
```
max_depth_list = list(clf_cv_RF_w2v.cv_results_['param_max_depth'].data)
estimators_list = list(clf_cv_RF_w2v.cv_results_['param_n_estimators'].data)
data = pd.DataFrame(data={'Estimators':estimators_list, 'Max Depth':max_depth_list, 'AUC':clf_cv_RF_w2v.cv_results_['mean_test_score']})
data = data.pivot(index='Estimators', columns='Max Depth', values='AUC')
sns.heatmap(data, annot=True, cmap="YlGnBu").set_title('AUC for Test data')
plt.show()
```



In [57]:
```
clf_cv_RF_w2v.best_params_
```

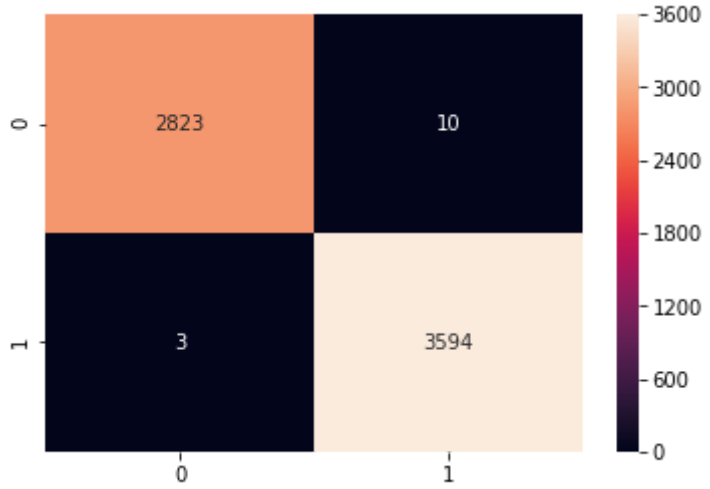
Out[57]:
```
{'max_depth': 50, 'n_estimators': 100}
```

In [58]:
```
#Testing with Test data
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

clf_RF_w2v = RandomForestClassifier(random_state=0,max_depth=50, n_estimators=100)
clf_RF_w2v.fit(sent_vectors_train, 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, clf_RF_w2v.predict_proba(sent_vectors_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, clf_RF_w2v.predict_proba(sent_vectors_test)[:,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("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
uniform_data = confusion_matrix(y_train, clf_RF_w2v.predict(sent_vectors_train))
ax = sns.heatmap(uniform_data,annot= True, fmt= "d")
```



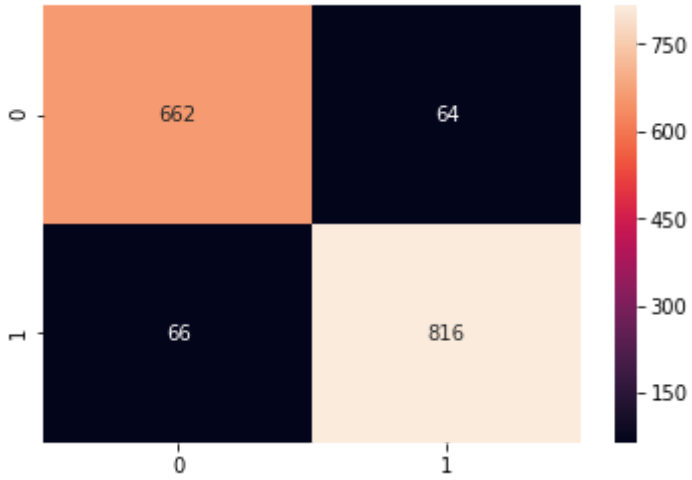
==========

Train confusion matrix



In [59]:
```
print("Test confusion matrix")
uniform_data = confusion_matrix(y_test, clf_RF_w2v.predict(sent_vectors_test))
ax = sns.heatmap(uniform_data,annot= True, fmt= "d")
```

```
Test confusion matrix
```



Applying Random Forests on TFIDF W2V

In [60]:
```
from sklearn.model_selection import train_test_split

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=Flase)# this is for time series sp
lit
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33) # this is random splitting
#X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33) # this is random splitting

#Preparing Reviews for gensim model

 #Training w2v model

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

# this line of code trains your w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=4)

w2v_words = list(w2v_model.wv.vocab)
```

In [61]:
```
#Converting Reviews into Numerical Vectors using W2V vectors

i=0
list_of_sentance_train=[]
for sentance in X_train:
    list_of_sentance_train.append(sentance.split())
#Algorithm: ifidf W2V
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_)))

from tqdm import tqdm
import numpy as np

#Converting Train data text

# 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

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

```
100%|██████████| 5385/5385 [00:01<00:00, 3625.09it/s]
```

In [62]:
```
i=0
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())

#Converting Reviews into Numerical Vectors using W2V vectors


#Algorithm: ifidf W2V
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_)))

from tqdm import tqdm
import numpy as np

#Converting Train data text

# 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

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
```

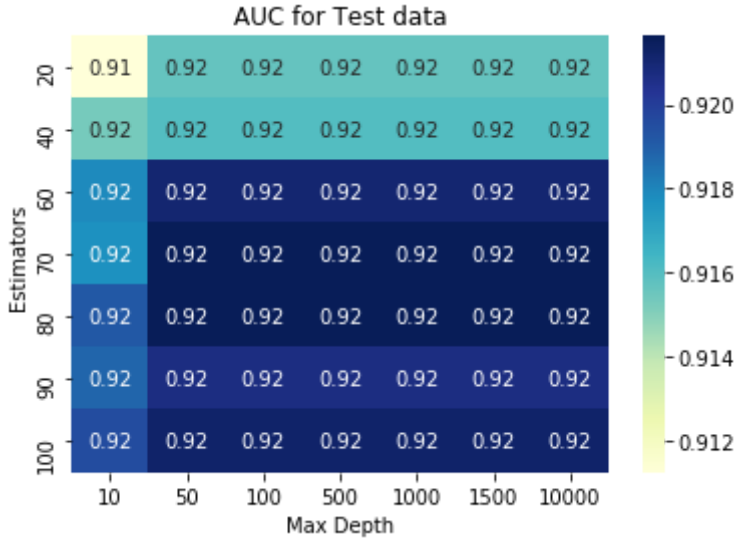
```
    sent_vectors_test.append(sent_vec)
    row += 1
```

```
100%|██████████| 2653/2653 [00:00<00:00, 3505.63it/s]
```

In [63]:
```
from sklearn.tree import DecisionTreeClassifier
#import GridSearchCV
from sklearn.model_selection import GridSearchCV
param_grid = {'max_depth': [10, 50, 100, 500, 1000, 1500, 10000], 'n_estimators': [20, 40, 60, 70, 80, 90, 100]}
clf = RandomForestClassifier(random_state=0)
clf_cv_RF_TfidfW2V= GridSearchCV(clf,param_grid,cv=5)
clf_cv_RF_TfidfW2V.fit(sent_vectors_train,y_train)
#clf = DecisionTreeClassifier(random_state=0,max_depth=None, min_samples_split=2)
```

Out[63]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators='warn', n_jobs=None,
                                              oob_score=False, random_state=0,
                                              verbose=0, warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'max_depth': [10, 50, 100, 500, 1000, 1500, 10000],
                         'n_estimators': [20, 40, 60, 70, 80, 90, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

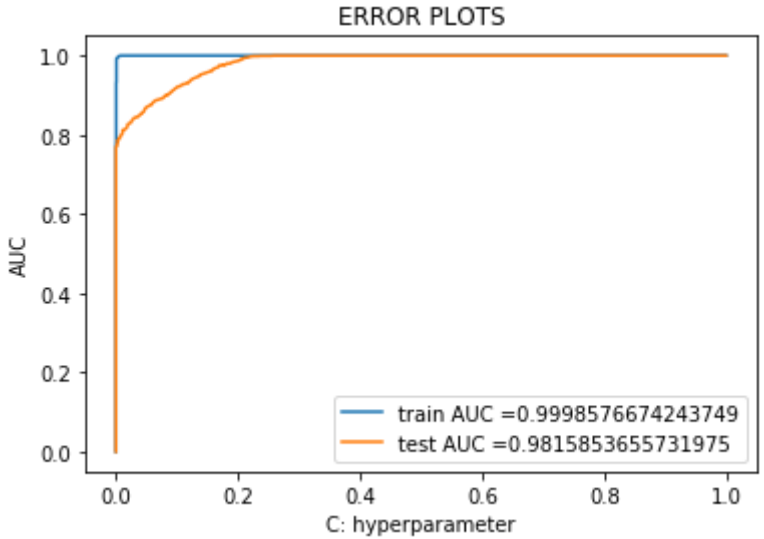
In [64]:
```
max_depth_list = list(clf_cv_RF_TfidfW2V.cv_results_['param_max_depth'].data)
estimators_list = list(clf_cv_RF_TfidfW2V.cv_results_['param_n_estimators'].data)
data = pd.DataFrame(data={'Estimators':estimators_list, 'Max Depth':max_depth_list, 'AUC':clf_cv_RF_TfidfW2V.cv_result
s_['mean_test_score']})
data = data.pivot(index='Estimators', columns='Max Depth', values='AUC')
sns.heatmap(data, annot=True, cmap="YlGnBu").set_title('AUC for Test data')
plt.show()
```



In [65]:
```
clf_cv_RF_TfidfW2V.best_params_
```

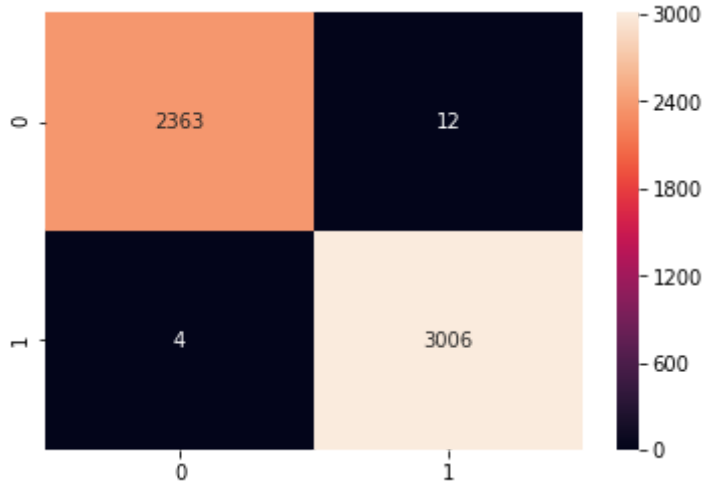
Out[65]:
```
{'max_depth': 50, 'n_estimators': 70}
```

In [66]:
```
#Testing with Test data
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

clf_RF_TfidfW2V = RandomForestClassifier(random_state=0,max_depth=50, n_estimators=70)
clf_RF_TfidfW2V.fit(sent_vectors_train, 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, clf_RF_TfidfW2V.predict_proba(sent_vectors_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, clf_RF_TfidfW2V.predict_proba(sent_vectors_test)[:,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("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
uniform_data = confusion_matrix(y_train, clf_RF_TfidfW2V.predict(sent_vectors_train))
ax = sns.heatmap(uniform_data,annot= True, fmt= "d")
```



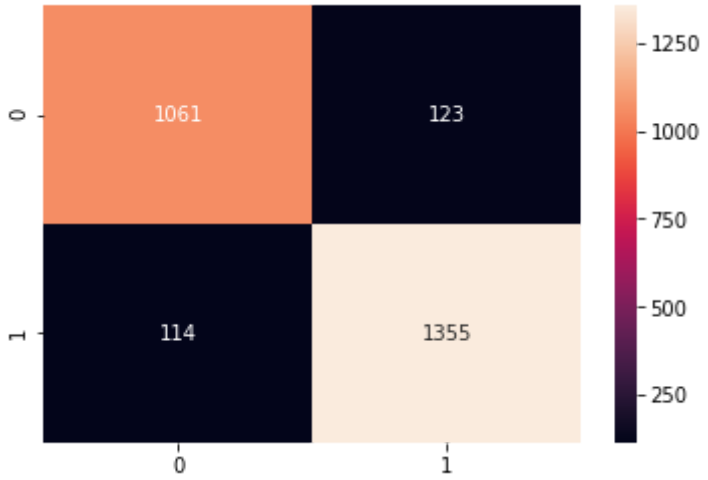
==========

Train confusion matrix



In [67]:
```
print("Test confusion matrix")
uniform_data = confusion_matrix(y_test, clf_RF_TfidfW2V.predict(sent_vectors_test))
ax = sns.heatmap(uniform_data,annot= True, fmt= "d")
```

```
Test confusion matrix
```



Conclusions

In [74]:
```
from prettytable import PrettyTable

x = PrettyTable()
y = PrettyTable()

y.field_names = ["Vectorizer", "Model", "hyperparameter", "best hyperparameter", "AUC"]

y.add_row(["BOW", "NaiveBayes", "alpha", 0.1, 0.99142])
y.add_row(["BOW", "Logistic Regression", "C", 1, 0.99407])

y.add_row(["TFIDF", "Naive Bayes ", "alpha", 0.0001, 0.99205])
y.add_row(["TFIDF", "Logistic Regression", "C", 1, 0.99474])

print(y)

x.field_names = ["Vectorizer", "Random Forests", "depth", "estimators", "AUC"]
x.add_row(["BOW", "Random Forests", 500 ,70, 0.99546])
x.add_row(["TFIDF", "Random Forests", 100, 40, 0.99417])
x.add_row(["W2v", "Random Forests", 100,40, 0.99317])
x.add_row(["TFIDFW2v", "Random Forests", 50,100 ,0.98383])
print(x)
```

```
+------------+---------------------+----------------+----------------------+---------+
| Vectorizer |        Model        | hyperparameter | best hyperparameter |   AUC   |
+------------+---------------------+----------------+----------------------+---------+
|    BOW     |      NaiveBayes     |     alpha      |         0.1          | 0.99142 |
|    BOW     | Logistic Regression |       C        |          1           | 0.99407 |
|   TFIDF    |     Naive Bayes     |     alpha      |        0.0001        | 0.99205 |
|   TFIDF    | Logistic Regression |       C        |          1           | 0.99474 |
+------------+---------------------+----------------+----------------------+---------+
+------------+----------------+-------+------------+---------+
| Vectorizer | Random Forests | depth | estimators |   AUC   |
+------------+----------------+-------+------------+---------+
|    BOW     | Random Forests |  500  |     70     | 0.99546 |
|   TFIDF    | Random Forests |  100  |     40     | 0.99417 |
|    W2v     | Random Forests |  100  |     40     | 0.99317 |
|  TFIDFW2v  | Random Forests |   50  |    100     | 0.98383 |
+------------+----------------+-------+------------+---------+
```

For Bow and Tfidf we get good AUC so by looking at confusion matrix, I am using Naive Bayes by BoW Vectorizer

applying model on text 3

In [29]:
```
from collections import defaultdict
final_new1 = defaultdict(list)
text1=[]
f=open('3.txt')
for line in f:
    final_new1['Text'].append(line[4:])

final_new =  pd.DataFrame(final_new1)
final_new = final_new.iloc[1:]
final_new.head()
```

Out[29]:

	Text
1	I exchanged the sony ericson z500a for this an...
2	Oh and I forgot to also mention the weird colo...
3	"Verizon tech support walked my through a few ...
4	Better than you'd expect.,\n
5	This is a great little item.,\n

In [30]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_new = []
# tqdm is for printing the status bar
for sentance in tqdm(final_new['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_new.append(sentance.strip())
```

```
100%|██████████| 932/932 [00:00<00:00, 2818.19it/s]
```

In [31]:
```
preprocessed_new[100]
```

Out[31]:
```
'not upload ringtones third party'
```

In [48]:
```
X = preprocessed
Y = final['label']
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
vectorizer.fit(X_train) # fit has to happen only on train data

X_train_final = vectorizer.transform(X_train)
X_test_final  = vectorizer.transform(X_test)
best_alpha = 0.1
clf_final = MultinomialNB(alpha=best_alpha, class_prior=None)
clf_final.fit(X_train_final, y_train)

from sklearn.metrics import classification_report
target_names = ['class 0', 'class 1']
print("classification report")
print(classification_report(y_train, clf_final.predict(X_train_final), target_names=target_names))
```

```
classification report
              precision    recall  f1-score   support

     class 0       0.99      0.99      0.99      2819
     class 1       0.99      0.99      0.99      3611

    accuracy                           0.99      6430
   macro avg       0.99      0.99      0.99      6430
weighted avg       0.99      0.99      0.99      6430
```

In [35]:
```
X_new_bow = vectorizer.transform(preprocessed_new)
```

In [36]:
```
ynew = clf_final.predict(X_new_bow)
```

In [41]:
```
print("X=%s, Predicted=%s" % (preprocessed_new[0], ynew[0]))
print("X=%s, Predicted=%s" % (final_new['Text'].values[0], ynew[0]))
```

```
X=exchanged sony ericson pretty happy decision, Predicted=1
X=I exchanged the sony ericson z500a for this and I'm pretty happy with that decision.,
, Predicted=1
```

In [42]:
```
final_new["Predicted"] = ynew
```

In [43]:
```
final_new.head()
```

Out[43]:

	Text	Predicted
1	I exchanged the sony ericson z500a for this an...	1
2	Oh and I forgot to also mention the weird colo...	0
3	"Verizon tech support walked my through a few ...	0
4	Better than you'd expect.,\n	0
5	This is a great little item.,\n	1

In [44]:
```
final_new.to_csv("DigitMainfile.csv")
```

In [ ]:
```

```