parth.pandey13103447@gmail.com_8

April 25, 2020

1 Assignment 6: Apply NB

1.1 Imports

[1]:
```
import os
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import Normalizer,StandardScaler
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_curve , auc, confusion_matrix, roc_auc_score
from scipy.sparse import hstack
from sklearn.model_selection import GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np
from tqdm import tqdm

import matplotlib.pyplot as plt
import seaborn as sns
import gc
%matplotlib inline
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19:
FutureWarning: pandas.util.testing is deprecated. Use the functions in the
public API at pandas.testing instead.
  import pandas.util.testing as tm
```

1.2 Reading Data

[2]:
```
os.listdir('drive/My Drive/Colab Notebooks/AppliedAI/datasets/assignment_8/')
```

[2]:
```
['preprocessed_data.csv']
```

[3]:
```
df = pd.read_csv('drive/My Drive/Colab Notebooks/AppliedAI/datasets/
 ↪assignment_8/preprocessed_data.csv')
print(df.columns)
```

```
Index(['school_state', 'teacher_prefix', 'project_grade_category',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'price'],
      dtype='object')
```

1.3 Splitting data into Train and cross validation(or test): Stratified Sampling

[0]:
```
label = df['project_is_approved'].values
df.drop('project_is_approved',axis=1,inplace=True)
x_train, x_test, y_train, y_test = train_test_split(df,label,test_size=0.
 ↪3,stratify=label)
```

1.4 Define Reusable Functions to be Used for Encoding Categorical and Numerical Features

[0]:
```
# Defining reusable function

def encoder_for_cat_columns(col,train,test):
    if col == 'essay':
        cnt_vec = 
 ↪CountVectorizer(ngram_range=(1,5),min_df=10,max_features=5000,binary=True)
    else:
        cnt_vec = CountVectorizer()

    cnt_vec.fit(train[col].values)
    return cnt_vec.transform(train[col].values), cnt_vec.transform(test[col].
 ↪values) , cnt_vec

def encoder_for_num_columns(col, train, test):
    normal = Normalizer()
    normal.fit(train[col].values.reshape(-1,1))
    return normal.transform(train[col].values.reshape(-1,1)) , normal.
 ↪transform(test[col].values.reshape(-1,1))

def tf_encoder_for_cat_columns(col,train,test):
    if col == 'essay':
        vec = 
 ↪TfidfVectorizer(ngram_range=(1,5),min_df=10,norm='l1',max_features=5000)
    else:
        vec = CountVectorizer()

    vec.fit(train[col].values)
    return vec.transform(train[col].values), vec.transform(test[col].values) ,
 ↪vec
```

1.5 SET - 1

Convert Data

[6]:
```
print('Storing all the bow transformations in single dictionary')
temp = {}
for ele in tqdm(df.columns):
    if type(df[ele][0] ) == str:
        temp_train , temp_test, temp_vec  =␣
↪encoder_for_cat_columns(ele,x_train,x_test)
        temp['x_train_{}_bow'.format(ele)] = temp_train
        del temp_train
        temp['x_test_{}_bow'.format(ele)]  = temp_test
        del temp_test
        temp['{}_vectorizer'.format(ele)] = temp_vec
        del temp_vec
    else:
        temp_train , temp_test = encoder_for_num_columns(ele, x_train, x_test)
        temp['x_train_{}_norm '.format(ele)] = temp_train
        del temp_train
        temp['x_test_{}_norm'.format(ele)]  = temp_test
        del temp_test

print('Creating seperate key val pairs for hstacking')
hstack_temp = {}
for ele in tqdm(['train','test']):
    hstack_temp[ele] = []
    for key in temp.keys():
        if key.find(ele) >=0:
            hstack_temp[ele].append(temp[key])

x_tr = hstack(hstack_temp['train']).tocsr()
x_te = hstack(hstack_temp['test']).tocsr()
del hstack_temp
vec = temp['essay_vectorizer']
del temp
gc.collect()
```

```
  0%|          | 0/8 [00:00<?, ?it/s]

Storing all the bow transformations in single dictionary

100%|      | 8/8 [05:32<00:00, 41.62s/it]
100%|      | 2/2 [00:00<00:00, 4286.46it/s]

Creating seperate key val pairs for hstacking
```

[6]: 0

[7]:
```
# Checking shapes
print(x_tr.shape)
print(x_te.shape)
```

```
(76473, 5101)
(32775, 5101)
```

Model Training With Hyper Parameter Tuning

[0]:
```
mnb = MultinomialNB()
parameters = {'alpha':[10**-5,10**-4 ,10**-3,10**-2,10**-1,1,1.
 ↪5,2,5,10,15,20,40,80,100,120,150,200]}
clf =␣
 ↪GridSearchCV(mnb,parameters,cv=5,scoring='roc_auc',return_train_score=True)
clf.fit(x_tr.toarray(),y_train)
results = pd.DataFrame(clf.cv_results_)
results.sort_values('param_alpha',inplace=True)
```

[9]:
```
results.head()
```

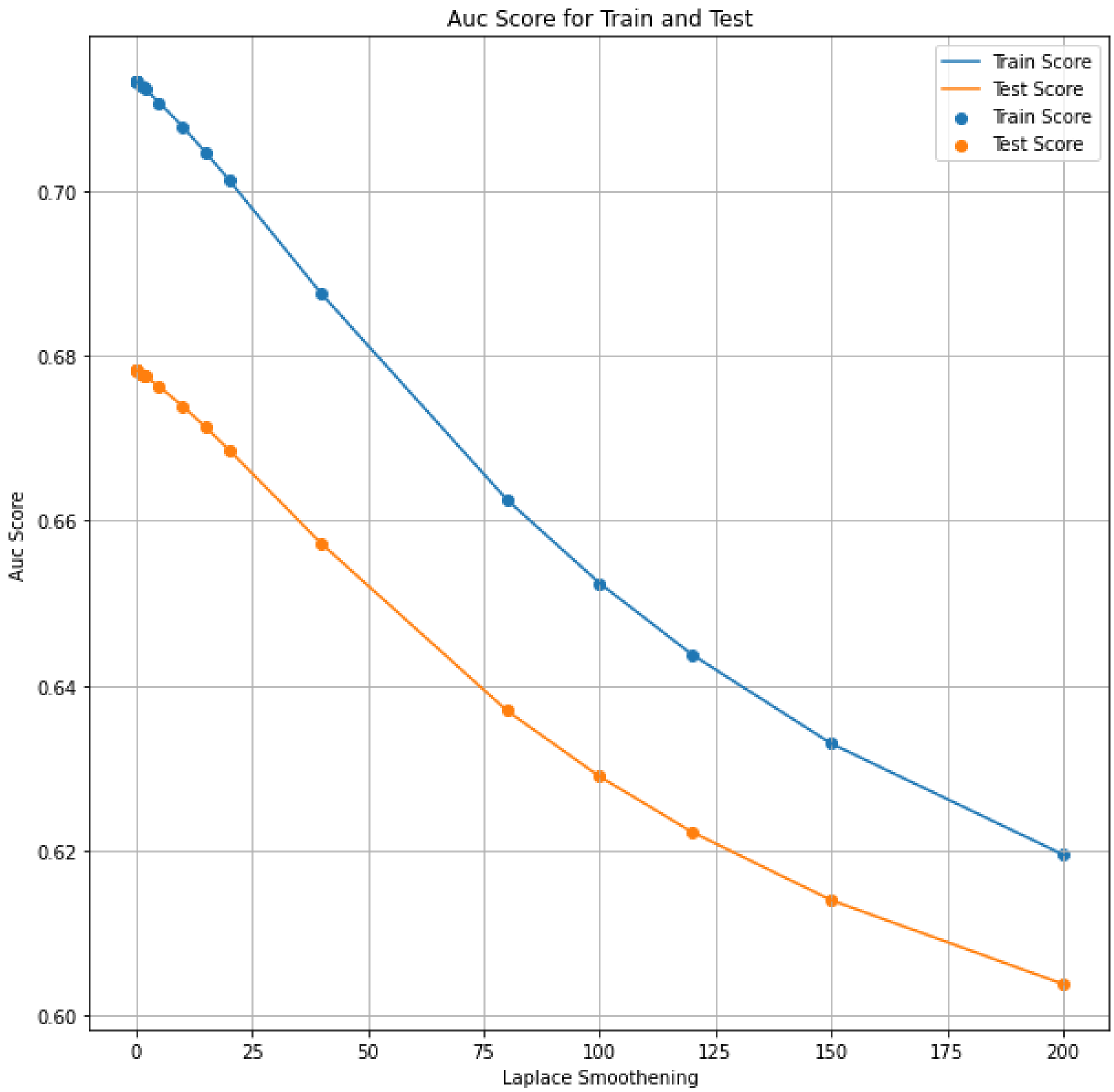
[9]:

	mean_fit_time	std_fit_time	…	mean_train_score	std_train_score
0	1.761435	0.058277	…	0.713223	0.000890
1	1.847528	0.219054	…	0.713223	0.000890
2	1.739930	0.004837	…	0.713222	0.000890
3	1.733608	0.008769	…	0.713218	0.000890
4	1.739172	0.021653	…	0.713176	0.000888

[5 rows x 21 columns]

Plotting Tuning Reaults

[10]:
```
fig , ax = plt.subplots(1,1,figsize=(10,10))
ax.plot(results['param_alpha'],results['mean_train_score'],label='Train Score')
ax.scatter(results['param_alpha'],results['mean_train_score'],label='Train␣
 ↪Score')
ax.plot(results['param_alpha'],results['mean_test_score'],label='Test Score')
ax.scatter(results['param_alpha'],results['mean_test_score'],label='Test Score')
ax.set_ylabel('Auc Score')
ax.set_xlabel('Laplace Smoothening')
ax.legend()
ax.grid()
# ax.set_xticks(map(str,results['param_alpha']))
ax.set_title('Auc Score for Train and Test')
plt.show()
```



[11]:
```
print('Alpha','Train Score','Test Score','Distance',sep=' ==> ')
for ind, row in results.iterrows():
    print(row.param_alpha,row.mean_train_score,row.mean_test_score,row.
 ↪mean_train_score-row.mean_test_score,sep=' ==> ')
```

```
Alpha ==> Train Score ==> Test Score ==> Distance
1e-05 ==> 0.7132229120620368 ==> 0.6782369339387515 ==> 0.0349859781232853
0.0001 ==> 0.713222887940761 ==> 0.6782369272632837 ==> 0.03498596067747739
0.001 ==> 0.7132223657977292 ==> 0.6782364614275054 ==> 0.034985904370223864
0.01 ==> 0.7132182691175999 ==> 0.6782326884401079 ==> 0.03498558067749202
0.1 ==> 0.7131758487699175 ==> 0.6781996566505855 ==> 0.034976192119332006
1 ==> 0.7127366733274936 ==> 0.6778540419536683 ==> 0.0348826313738253
1.5 ==> 0.712487861221214 ==> 0.6776599572974954 ==> 0.034827903923718595
2 ==> 0.7122343180673263 ==> 0.6774647680335718 ==> 0.03476955003375448
```

```
5 ==> 0.7106448725335511 ==> 0.6762204627400423 ==> 0.0344244097935088
10 ==> 0.7077473890221173 ==> 0.6738878101389041 ==> 0.033859578883213226
15 ==> 0.70460852086952 ==> 0.6713135407211824 ==> 0.03329498014833754
20 ==> 0.7013015534166469 ==> 0.6685868164077398 ==> 0.03271473700890715
40 ==> 0.6874583216194153 ==> 0.657184141377584 ==> 0.030274180241831372
80 ==> 0.6625185707858328 ==> 0.6369588623857695 ==> 0.025559708400063275
100 ==> 0.6524017929391002 ==> 0.628973320980373 ==> 0.023428471958727215
120 ==> 0.6437302231905987 ==> 0.6222296887564285 ==> 0.021500534434170215
150 ==> 0.6329585710503209 ==> 0.6139836056524934 ==> 0.018974965397827503
200 ==> 0.619508402962532 ==> 0.6038242255547062 ==> 0.015684177407825795
```

Choose Best Alpha

[0]:
```
# Taking the best alpha
alpha = 0.01
```

Retraining Final Model

[0]:
```
mnb = MultinomialNB(alpha=alpha)
mnb.fit(x_tr.toarray(),y_train)
y_te_predict = mnb.predict(x_te.toarray())
y_tr_predict = mnb.predict(x_tr.toarray())
```

Check Results

[14]:
```
print("Train Data Results")
print(roc_auc_score(y_train,y_tr_predict))
print("Test Data Results")
print(roc_auc_score(y_test,y_te_predict))
```

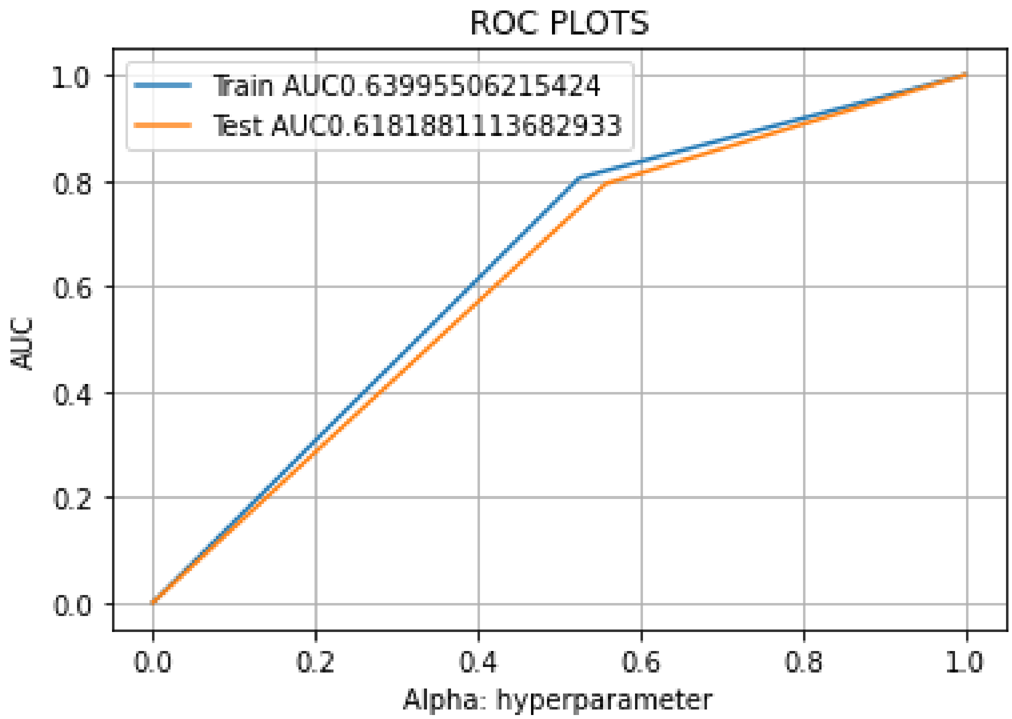
```
Train Data Results
0.63995506215424
Test Data Results
0.6181881113682933
```

Plotting ROC Curve

[0]:
```
train_fpr , train_tpr , tr_thresholds = roc_curve(y_train,y_tr_predict)
test_fpr , test_tpr , te_threshokds = roc_curve(y_test,y_te_predict)
```

[16]:
```
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("ROC PLOTS")
plt.grid()
```

```
plt.show()
```



Confusion Matrix

[0]:
```
te_conf = confusion_matrix(y_test,y_te_predict)
tr_conf = confusion_matrix(y_train,y_tr_predict)
```

[18]:
```
print('Train Data Confusion Matrix')
print(tr_conf)
print('Test Data Confusion Matrix')
print(te_conf)
```

```
Train Data Confusion Matrix
[[ 5495  6084]
 [12632 52262]]
Test Data Confusion Matrix
[[ 2196  2767]
 [ 5732 22080]]
```

[19]:
```
print('Final Confusion Matrix')
print(te_conf)
```

```
Final Confusion Matrix
[[ 2196  2767]
```

```
[ 5732 22080]]
```

Top 20 Feature Representation

[20]:
```python
# print('Top 20 Features ')
# https://stackoverflow.com/questions/29867367/
 ↪sklearn-multinomial-nb-most-informative-features
# Getting the feature names
feature_names = vec.get_feature_names()
print('\nTop 20 Features for class 0\n')
for feature_index in mnb.feature_log_prob_[0].argsort()[:20]:
    print(feature_names[feature_index],end='\n')
print('\nTop 20 Features for class 1\n')
for feature_index in mnb.feature_log_prob_[1].argsort()[:20]:
    print(feature_names[feature_index],end='\n')
```

```
Top 20 Features for class 0

4th grade
40
4th 5th
21st century learning
this technology
students constantly
time day
throughout school
11
throughout
coding
classroom focus potential growth
cold
day they
help learn
chairs allow
2nd graders
8th
century skills
classroom full

Top 20 Features for class 1

4th grade
40
4th 5th
21st century learning
8th
21st century
```

```
30 students
21st century skills
2nd graders
000
22
11
80 students
100 students receive free
20 students
4th
13
thus
this project help
90 students
```

1.6 SET - 2

Convert Data

```
[21]: print('Storing all the bow transformations in single dictionary')
temp = {}
for ele in tqdm(df.columns):
    if type(df[ele][0] ) == str:
        temp_train , temp_test, temp_vec =␣
 ↪tf_encoder_for_cat_columns(ele,x_train,x_test)
        temp['x_train_{}_bow'.format(ele)] = temp_train
        del temp_train
        temp['x_test_{}_bow'.format(ele)]  = temp_test
        del temp_test
        temp['{}_vectorizer'.format(ele)] = temp_vec
        del temp_vec
    else:
        temp_train , temp_test = encoder_for_num_columns(ele, x_train, x_test)
        temp['x_train_{}_norm '.format(ele)] = temp_train
        del temp_train
        temp['x_test_{}_norm'.format(ele)]  = temp_test
        del temp_test

print('Creating seperate key val pairs for hstacking')
hstack_temp = {}
for ele in tqdm(['train','test']):
    hstack_temp[ele] = []
    for key in temp.keys():
        if key.find(ele) >=0:
            hstack_temp[ele].append(temp[key])

x_tr = hstack(hstack_temp['train']).tocsr()
```

```
x_te = hstack(hstack_temp['test']).tocsr()
del hstack_temp
vec = temp['essay_vectorizer']
del temp
gc.collect()
```

```
  0%|          | 0/8 [00:00<?, ?it/s]

Storing all the bow transformations in single dictionary

100%|     | 8/8 [05:39<00:00, 42.41s/it]
100%|     | 2/2 [00:00<00:00, 1794.74it/s]

Creating seperate key val pairs for hstacking
```

[21]:
```
2808
```

[22]:
```
# Checking shapes
print(x_tr.shape)
print(x_te.shape)
```

```
(76473, 5101)
(32775, 5101)
```

Model Training With Hyper Parameter Tuning

[0]:
```
gnb = MultinomialNB()
parameters = {'alpha':[10**-5,10**-4 ,10**-3,10**-2,10**-1,1,1.
 ↪5,2,5,10,15,20,40,80,100,120,150,200]}
clf =␣
 ↪GridSearchCV(gnb,parameters,cv=5,scoring='roc_auc',return_train_score=True)
clf.fit(x_tr.toarray(),y_train)
results = pd.DataFrame(clf.cv_results_)
results.sort_values('param_alpha',inplace=True)
```

[24]:
```
results.head(10)
```

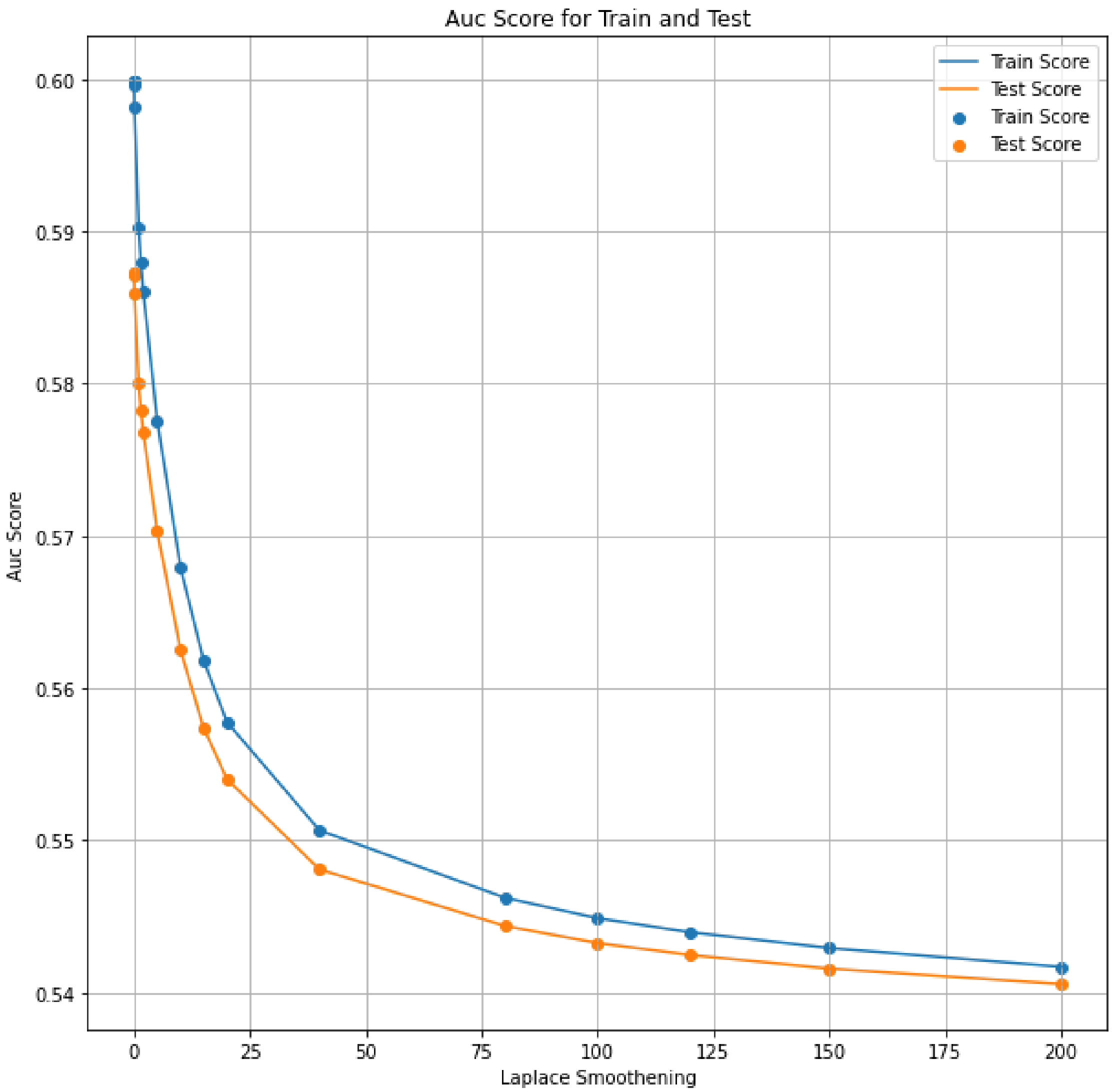
[24]:

	mean_fit_time	std_fit_time	…	mean_train_score	std_train_score
0	1.901166	0.355068	…	0.599861	0.001155
1	1.721315	0.006975	…	0.599859	0.001155
2	1.749149	0.016357	…	0.599841	0.001156
3	1.721088	0.003928	…	0.599665	0.001161
4	1.713492	0.007842	…	0.598111	0.001203
5	1.729315	0.013923	…	0.590296	0.001476
6	1.722646	0.010273	…	0.587949	0.001596
7	1.722701	0.015895	…	0.586036	0.001701
8	1.713805	0.014077	…	0.577566	0.002112
9	1.722686	0.014931	…	0.567957	0.002306

```
[10 rows x 21 columns]
```

Plotting Tuning Results

```
[25]: fig , ax = plt.subplots(1,1,figsize=(10,10))
ax.plot(results['param_alpha'],results['mean_train_score'],label='Train Score')
ax.scatter(results['param_alpha'],results['mean_train_score'],label='Train␣
 ↪Score')
ax.plot(results['param_alpha'],results['mean_test_score'],label='Test Score')
ax.scatter(results['param_alpha'],results['mean_test_score'],label='Test Score')
ax.set_ylabel('Auc Score')
ax.set_xlabel('Laplace Smoothening')
ax.legend()
ax.grid()
# ax.set_xticks(map(str,results['param_alpha']))
ax.set_title('Auc Score for Train and Test')
plt.show()
```



[26]:
```
print('Alpha','Train Score','Test Score','Distance',sep=' ==> ')
for ind, row in results.iterrows():
    print(row.param_alpha,row.mean_train_score,row.mean_test_score,row.
 ↪mean_train_score-row.mean_test_score,sep=' ==> ')
```

```
Alpha ==> Train Score ==> Test Score ==> Distance
1e-05 ==> 0.5998610246697537 ==> 0.5872647949391946 ==> 0.012596229730559072
0.0001 ==> 0.5998593029035579 ==> 0.5872634374428776 ==> 0.012595865460680367
0.001 ==> 0.5998413732228443 ==> 0.5872498564810933 ==> 0.012591516741750963
0.01 ==> 0.599665498997761 ==> 0.5871149975580321 ==> 0.012550501439728912
0.1 ==> 0.5981110216188819 ==> 0.5859428098278313 ==> 0.012168211791050543
1 ==> 0.5902958168519156 ==> 0.5799930650502707 ==> 0.01030275180164486
1.5 ==> 0.5879486018613663 ==> 0.5782138028455973 ==> 0.009734799015769013
2 ==> 0.5860357763982688 ==> 0.5767929169029637 ==> 0.009242859495305011
```

```
5 ==> 0.5775664228907667 ==> 0.5702813828251678 ==> 0.0072850400655988246
10 ==> 0.5679573955639038 ==> 0.5625259617270219 ==> 0.005431433836881938
15 ==> 0.5618315449981331 ==> 0.5573896905637405 ==> 0.0044418544343926
20 ==> 0.5577864870395592 ==> 0.554034008759778 ==> 0.0037524782797812017
40 ==> 0.5506543042547244 ==> 0.5480879817607813 ==> 0.002566322493943063
80 ==> 0.5462240995859207 ==> 0.5443761134846161 ==> 0.001847986101304544
100 ==> 0.5448818535577568 ==> 0.543246464961943 ==> 0.0016353885958138026
120 ==> 0.5439576317119622 ==> 0.542466330881209 ==> 0.0014913008307532172
150 ==> 0.5429222283756338 ==> 0.5415760332319571 ==> 0.0013461951436766206
200 ==> 0.5416892432732692 ==> 0.5405542717570091 ==> 0.0011349715162600749
```

Choose Best Alpha

[0]:
```python
alpha = 1
```

Retrain Final Model

[0]:
```python
mnb = MultinomialNB(alpha=alpha)
mnb.fit(x_tr.toarray(),y_train)
y_te_predict = mnb.predict(x_te.toarray())
y_tr_predict = mnb.predict(x_tr.toarray())
```

Check Results

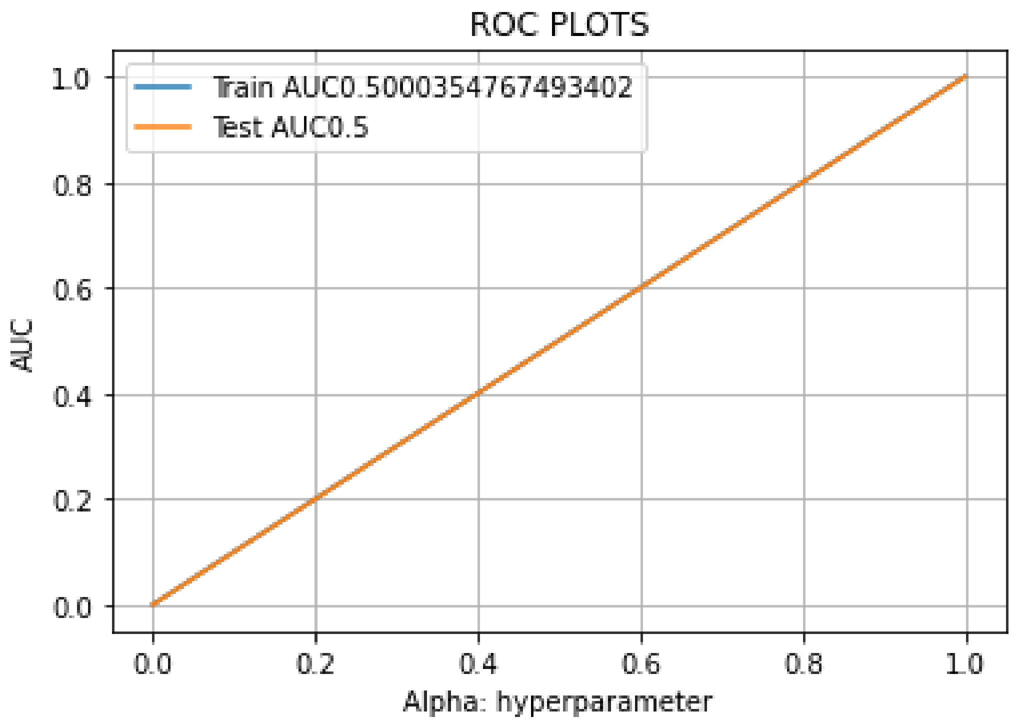
[42]:
```python
print("Train Data Results")
print(roc_auc_score(y_train,y_tr_predict))
print("Test Data Results")
print(roc_auc_score(y_test,y_te_predict))
```

```
Train Data Results
0.5000354767493402
Test Data Results
0.5
```

Plotting ROC Curve

[0]:
```python
train_fpr , train_tpr , tr_thresholds = roc_curve(y_train,y_tr_predict)
test_fpr , test_tpr , te_threshokds = roc_curve(y_test,y_te_predict)
```

[44]:
```python
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("ROC PLOTS")
plt.grid()
plt.show()
```



Confusion Matrix

[0]:
```
te_conf = confusion_matrix(y_test,y_te_predict)
tr_conf = confusion_matrix(y_train,y_tr_predict)
```

[46]:
```
print('Train Confusion Matrix\n')
print(tr_conf)
print('Test Confusion Matrix')
print(te_conf)
```

```
Train Confusion Matrix

[[    1 11578]
 [    1 64893]]
Test Confusion Matrix
[[    0  4963]
 [    0 27812]]
```

[47]:
```
print('Final Confusion Matrix')
print(te_conf)
```

```
Final Confusion Matrix
[[    0  4963]
 [    0 27812]]
```

Top 20 Features Representation

[48]:

```
feature_names = vec.get_feature_names()
print('\nTop 20 Features for class 0\n')
for feature_index in mnb.feature_log_prob_[0].argsort()[:20]:
    print(feature_names[feature_index],end='\n')
print('\nTop 20 Features for class 1\n')
for feature_index in mnb.feature_log_prob_[1].argsort()[:20]:
    print(feature_names[feature_index],end='\n')
```

```
Top 20 Features for class 0

this technology
time day
students creative meaningful
cognitive
classroom focus
day they
throughout school day
collaborate
born
throughout day
classroom focus potential
technology nannan
single parent households
things they
help kids
technology engineering
chairs allow
later
art room
century learning

Top 20 Features for class 1

informational
magnetic
income high poverty school district
day long
income high poverty school
magnet school
children learn
children love
classroom every
information
classroom environment
despite challenges
```

```
despite
second language
children come
industry
classroom despite many challenges face
ahead
day learning
day eager
```