NavieBayes_Assignment_updated

April 21, 2020

Naive Bayes

0.1 Loading Data

[1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
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

import re

import pickle
from tqdm import tqdm
import os

import chart_studio.plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

[2]:
```
import pandas
data = pandas.read_csv('preprocessed_data.csv') #loading preprocessed data
```

[3]:
```
data.head()
```

[3]:
```
  school_state teacher_prefix project_grade_category  \
0           ca            mrs          grades_prek_2
```

```
1           ut             ms             grades_3_5
2           ca            mrs          grades_prek_2
3           ga            mrs          grades_prek_2
4           wa            mrs             grades_3_5

   teacher_number_of_previously_posted_projects  project_is_approved  \
0                                            53                   1
1                                             4                   1
2                                            10                   1
3                                             2                   1
4                                             2                   1

    clean_categories                clean_subcategories  \
0       math_science  appliedsciences health_lifescience
1       specialneeds                        specialneeds
2  literacy_language                           literacy
3    appliedlearning                 earlydevelopment
4  literacy_language                           literacy

                                              essay   price
0  i fortunate enough use fairy tale stem kits cl…  725.05
1  imagine 8 9 years old you third grade classroo…  213.03
2  having class 24 students comes diverse learner…  329.00
3  i recently read article giving students choice…  481.04
4  my students crave challenge eat obstacles brea…   17.74
```

[4]:
```
data["project_is_approved"].value_counts()
```

[4]:
```
1    92706
0    16542
Name: project_is_approved, dtype: int64
```

[5]:
```
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
X.head(1)
```

[5]:
```
  school_state teacher_prefix project_grade_category  \
0           ca            mrs          grades_prek_2

   teacher_number_of_previously_posted_projects clean_categories  \
0                                            53     math_science

                clean_subcategories  \
0  appliedsciences health_lifescience

                                              essay   price
0  i fortunate enough use fairy tale stem kits cl…  725.05
```

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

[6]:
```
# train test split
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,␣
 ↪stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.
 ↪33, stratify=y_train)


print("Shape of Train, CV and Test Dataset")  #after splitting data
print(X_train.shape, y_train.shape)       #shapes of train,cv,test datsets
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)
```

```
Shape of Train, CV and Test Dataset
(49041, 8) (49041,)
(24155, 8) (24155,)
(36052, 8) (36052,)
```

Make Data Model Ready: encoding essay

0.1.1 encoding essay using BOW

[7]:
```
#BOW ON Preproccesed Essay

vectorizer_essaybow = CountVectorizer(min_df=10)
vectorizer_essaybow.fit(X_train['essay'].values) # fit has to happen only on␣
 ↪train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer_essaybow.transform(X_train['essay'].values)
X_cv_essay_bow = vectorizer_essaybow.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer_essaybow.transform(X_test['essay'].values)

print("After BOW vectorizations of Essay")
print(X_train_essay_bow.shape, y_train.shape)  #shapes of Train ,cv ,Test after␣
 ↪encoding
print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
```

```
After BOW vectorizations of Essay
(49041, 12116) (49041,)
(24155, 12116) (24155,)
(36052, 12116) (36052,)
```

0.1.2 encoding essay using TFIDF

[8]:
```
#TFIDF ON Preproccesed Essay

vectorizer_essaytfidf = TfidfVectorizer(min_df=10)
vectorizer_essaytfidf.fit(X_train['essay'].values) # fit has to happen only on␣
 ↪train data

# we use the fitted TfidfVectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer_essaytfidf.transform(X_train['essay'].values)
X_cv_essay_tfidf = vectorizer_essaytfidf.transform(X_cv['essay'].values)
X_test_essay_tfidf = vectorizer_essaytfidf.transform(X_test['essay'].values)

print("After Tfidf vectorization of Essay")
print(X_train_essay_tfidf.shape, y_train.shape)  #shapes of Train ,cv ,Test␣
 ↪after encoding
print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
```

```
After Tfidf vectorization of Essay
(49041, 12116) (49041,)
(24155, 12116) (24155,)
(36052, 12116) (36052,)
```

Make Data Model Ready: encoding numerical, categorical features

encoding categorical features: School State

[9]:
```

vectorizer_state = CountVectorizer()
vectorizer_state.fit(X_train['school_state'].values) # fit has to happen only␣
 ↪on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_state= vectorizer_state.transform(X_train['school_state'].values)
X_cv_state= vectorizer_state.transform(X_cv['school_state'].values)
X_test_state= vectorizer_state.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state.shape, y_train.shape)
print(X_cv_state.shape, y_cv.shape)
print(X_test_state.shape, y_test.shape)
print(vectorizer_state.get_feature_names())
```

```
After vectorizations
(49041, 51) (49041,)
(24155, 51) (24155,)
(36052, 51) (36052,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia',
```

```
'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms',
'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'or', 'pa',
'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'wy']
```

encoding categorical features: teacher_prefix

[10]:
```
vectorizer_prefix= CountVectorizer()
vectorizer_prefix.fit(X_train['teacher_prefix'].values) # fit has to happen␣
 ↪only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher= vectorizer_prefix.transform(X_train['teacher_prefix'].values)
X_cv_teacher= vectorizer_prefix.transform(X_cv['teacher_prefix'].values)
X_test_teacher= vectorizer_prefix.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher.shape, y_train.shape)
print(X_cv_teacher.shape, y_cv.shape)
print(X_test_teacher.shape, y_test.shape)
print(vectorizer_prefix.get_feature_names())
```

```
After vectorizations
(49041, 5) (49041,)
(24155, 5) (24155,)
(36052, 5) (36052,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
```

encoding categorical features: project_grade_category

[11]:
```
vectorizer_grade = CountVectorizer()
vectorizer_grade.fit(X_train['project_grade_category'].values) # fit has to␣
 ↪happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade= vectorizer_grade.transform(X_train['project_grade_category'].
 ↪values)
X_cv_grade= vectorizer_grade.transform(X_cv['project_grade_category'].values)
X_test_grade= vectorizer_grade.transform(X_test['project_grade_category'].
 ↪values)

print("After vectorizations")
print(X_train_grade.shape, y_train.shape)
print(X_cv_grade.shape, y_cv.shape)
print(X_test_grade.shape, y_test.shape)
print(vectorizer_grade.get_feature_names())
```

```
After vectorizations
(49041, 4) (49041,)
(24155, 4) (24155,)
```

```
(36052, 4) (36052,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
```

encoding categorical features: clean_categories

[12]:
```
vectorizer_categories=CountVectorizer()
vectorizer_categories.fit(X_train["clean_categories"])

X_train_categories=vectorizer_categories.transform(X_train["clean_categories"].
 ↪values)
X_cv_categories=vectorizer_categories.transform(X_cv["clean_categories"].values)
X_test_categories=vectorizer_categories.transform(X_test["clean_categories"].
 ↪values)

print("After vectorizations")
print(X_train_categories.shape,y_train.shape)
print(X_cv_categories.shape,y_cv.shape)
print(X_test_categories.shape,y_test.shape)
print(vectorizer_categories.get_feature_names())
```

```
After vectorizations
(49041, 9) (49041,)
(24155, 9) (24155,)
(36052, 9) (36052,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics',
'literacy_language', 'math_science', 'music_arts', 'specialneeds', 'warmth']
```

encoding categorical features: clean_subcategories

[13]:
```
vectorizer_subcategories=CountVectorizer()
vectorizer_subcategories.fit(X_train["clean_subcategories"])

X_train_subcategories=vectorizer_subcategories.
 ↪transform(X_train["clean_subcategories"].values)
X_cv_subcategories=vectorizer_subcategories.
 ↪transform(X_cv["clean_subcategories"].values)
X_test_subcategories=vectorizer_subcategories.
 ↪transform(X_test["clean_subcategories"].values)

print("After vectorizations")
print(X_train_subcategories.shape,y_train.shape)
print(X_cv_subcategories.shape,y_cv.shape)
print(X_test_subcategories.shape,y_test.shape)
print(vectorizer_subcategories.get_feature_names())
```

```
After vectorizations
(49041, 30) (49041,)
(24155, 30) (24155,)
(36052, 30) (36052,)
```

```
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics',
'environmentalscience', 'esl', 'extracurricular', 'financialliteracy',
'foreignlanguages', 'gym_fitness', 'health_lifescience', 'health_wellness',
'history_geography', 'literacy', 'literature_writing', 'mathematics', 'music',
'nutritioneducation', 'other', 'parentinvolvement', 'performingarts',
'socialsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
```

encoding numerical features: Price

[14]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

normalizer.fit(X_train['price'].values.reshape(1,-1))

X_train_price= (normalizer.transform(X_train['price'].values.reshape(1,-1))).
 ↪reshape(-1,1)
X_cv_price= (normalizer.transform(X_cv['price'].values.reshape(1,-1))).
 ↪reshape(-1,1)
X_test_price=(normalizer.transform(X_test['price'].values.reshape(1,-1))).
 ↪reshape(-1,1)

print("Shape of price matrix")
print(X_train_price.shape, y_train.shape)
print(X_cv_price.shape, y_cv.shape)
print(X_test_price.shape, y_test.shape)
```

```
Shape of price matrix
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
```

encoding numerical features: teacher_number_of_previously_posted_projects

[15]:
```
normalizer = Normalizer()

normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.
 ↪reshape(1,-1))

X_train_prev_proj=(normalizer.
 ↪transform(X_train['teacher_number_of_previously_posted_projects'].values.
 ↪reshape(1,-1))).reshape(-1,1)
```

```
X_cv_prev_proj=(normalizer.
 ↪transform(X_cv['teacher_number_of_previously_posted_projects'].values.
 ↪reshape(1,-1))).reshape(-1,1)
X_test_prev_proj=(normalizer.
 ↪transform(X_test['teacher_number_of_previously_posted_projects'].values.
 ↪reshape(1,-1))).reshape(-1,1)

print("shape of matrix")
print(X_train_prev_proj.shape, y_train.shape)
print(X_cv_prev_proj.shape, y_cv.shape)
print(X_test_prev_proj.shape, y_test.shape)
```

```
shape of matrix
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
```

0.1.3 Concatinating all the features (essay-BOW)

[16]:
```
#concatinating all features- essay(BOW)+Categorical+ Numerical Features using␣
 ↪hstack

from scipy.sparse import hstack

X_train_bow=hstack((X_train_state,X_train_teacher,X_train_grade,X_train_categories,X_train_sub
 ↪tocsr()
X_cv_bow=hstack((X_cv_state,X_cv_teacher,X_cv_grade,X_cv_categories,X_cv_subcategories,X_cv_es
 ↪tocsr()
X_test_bow=hstack((X_test_state,X_test_teacher,X_test_grade,X_test_categories,X_test_subcatego
 ↪tocsr()

print("Final Data matrix")
print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)
```

```
Final Data matrix
(49041, 12217) (49041,)
(24155, 12217) (24155,)
(36052, 12217) (36052,)
```

0.1.4 Concatinating all the features (essay-TFIDF)

[17]:
```
#concatinating all features- essay(TFIDF)+Categorical+ Numerical Features using␣
 ↪hstack
```

```
X_train_tfidf=hstack((X_train_state,X_train_teacher,X_train_grade,X_train_categories,X_train_s
 ↪tocsr()
X_cv_tfidf=hstack((X_cv_state,X_cv_teacher,X_cv_grade,X_cv_categories,X_cv_subcategories,X_cv_
 ↪tocsr()
X_test_tfidf=hstack((X_test_state,X_test_teacher,X_test_grade,X_test_categories,X_test_subcate
 ↪tocsr()

print("Final Data matrix")
print(X_train_tfidf.shape, y_train.shape)
print(X_cv_tfidf.shape, y_cv.shape)
print(X_test_tfidf.shape, y_test.shape)
```

```
Final Data matrix
(49041, 12217) (49041,)
(24155, 12217) (24155,)
(36052, 12217) (36052,)
```

Appling Multinomial NB on Set 1 (Essay-BOW)

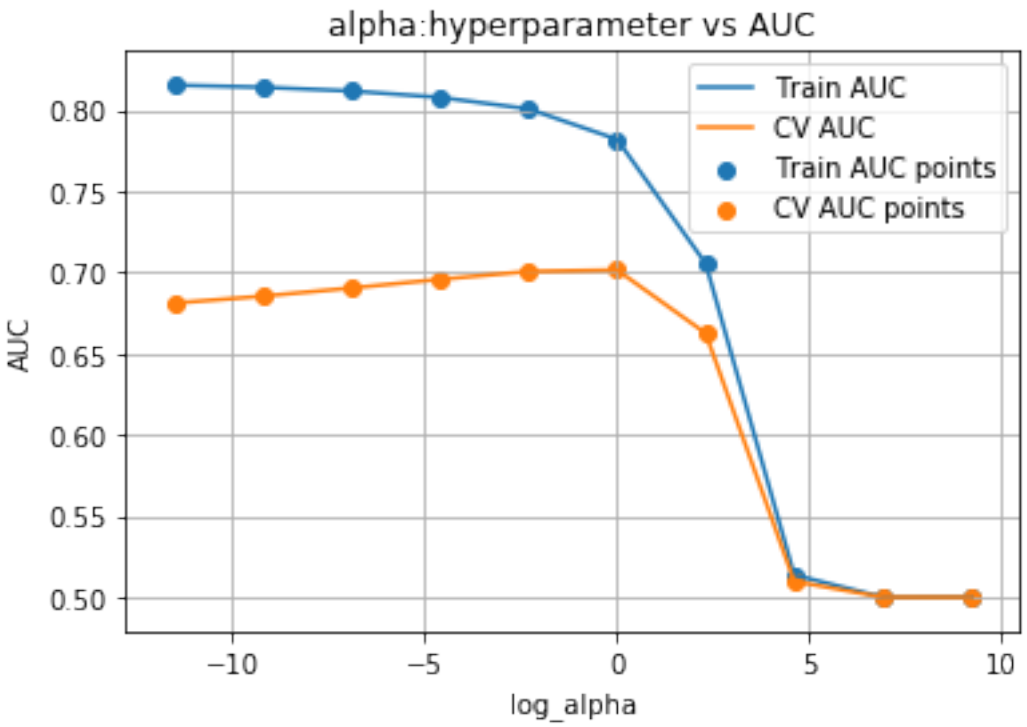
0.1.5 Hyper parameter Tuning using simple cross validation (using for loop)

[18]:
```
#function to predict probabilities batch wise
#We may get stuck if we find probality predictions of all dataponts at a time
#so,we calculate batch wise prediction of probalties
#lets the batch size be 1000 datapoints

def batch_predict(clf, data):
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 -␣
 ↪49041%1000 = 49000
    # in this for loop we will iterate until the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])
    return y_data_pred
```

[19]:
```
import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
import math
train_auc = []
cv_auc = []
alpha_values=[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000] # distinct␣
 ↪alpha values
```

```
log_alpha=[]
for i in range(len(alpha_values)):
    log_alpha.append(math.log(alpha_values[i]))
for i in tqdm(alpha_values):
    clf = MultinomialNB(alpha=i,class_prior=[0.5, 0.5])  #applying multinomial␣
 ↪NB on different alpha values
    clf.fit(X_train_bow,y_train)  #fitting the datamodel
    y_train_bow_pred = batch_predict(clf,X_train_bow)
    y_cv_bow_pred = batch_predict(clf,X_cv_bow)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability␣
 ↪estimates of the positive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_bow_pred))#roc_auc_score of␣
 ↪train and cv dataponts for different values of alpha
    cv_auc.append(roc_auc_score(y_cv,y_cv_bow_pred))


plt.plot(log_alpha,train_auc,label='Train AUC') #plotting train and cv␣
 ↪auc_scores for different values of alpha
plt.plot(log_alpha,cv_auc,label='CV AUC')
plt.scatter(log_alpha,train_auc,label='Train AUC points')
plt.scatter(log_alpha,cv_auc,label='CV AUC points')
plt.legend()
plt.xlabel("log_alpha")
plt.ylabel("AUC")
plt.title("alpha:hyperparameter vs AUC")
plt.grid()
plt.show()
```

```
100%|
    | 10/10 [00:02<00:00,  4.82it/s]
```



finding best alpha:smoothing parameter from above plot

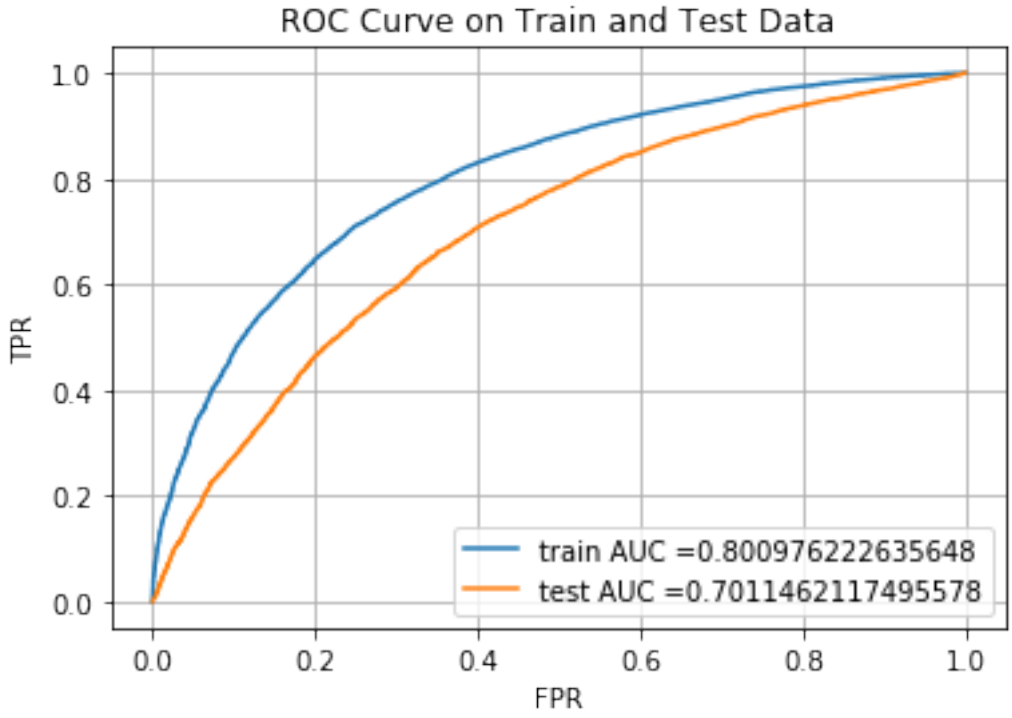
[20]:
```
best_alpha=0.1
print("best alpha(Hyper Parameter) which gives maximum AUC is {}".
 ↪format(best_alpha))
```

```
best alpha(Hyper Parameter) which gives maximum AUC is 0.1
```

0.2 Testing the performance of the model on test data , plotting ROC Curves

[21]:
```
from sklearn.metrics import roc_curve, auc


bow_bestclf=MultinomialNB(alpha=best_alpha,class_prior=[0.5,0.5]) #applying␣
 ↪multinomialNB for best alpha
bow_bestclf.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

y_train_pred = batch_predict(bow_bestclf,X_train_bow)
y_test_pred = batch_predict(bow_bestclf,X_test_bow)

train_fpr,train_tpr,train_thresolds = roc_curve(y_train, y_train_pred)
```

```
test_fpr,test_tpr, test_thresolds = roc_curve(y_test, y_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)))␣
 ↪#plotting ROC Curves for train and test datasets
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve on Train and Test Data")
plt.grid()
plt.show()
```



[22]:
```
print("AUC score on Train data is ")
print(auc(train_fpr, train_tpr))
print("AUC score on Test data is ")
print(auc(test_fpr, test_tpr))
```

```
AUC score on Train data is
0.800976222635648
AUC score on Test data is
0.7011462117495578
```

[23]:
```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_thresold(thresold, fpr, tpr):
    t = thresold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr) is ", max(tpr*(1-fpr)), "for␣
 ↪thresold", np.round(t,3))
    return t

def predict_with_best_t(proba,thresold):
    predictions = []
    for i in proba:
        if i>=thresold:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

[24]:
```
from sklearn.metrics import confusion_matrix
best_t = find_best_thresold(train_thresolds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) is  0.5336329949757144 for thresold 0.578
Train confusion matrix
[[ 5576  1850]
 [12040 29575]]
Test confusion matrix
[[ 3397  2062]
 [ 9633 20960]]
```

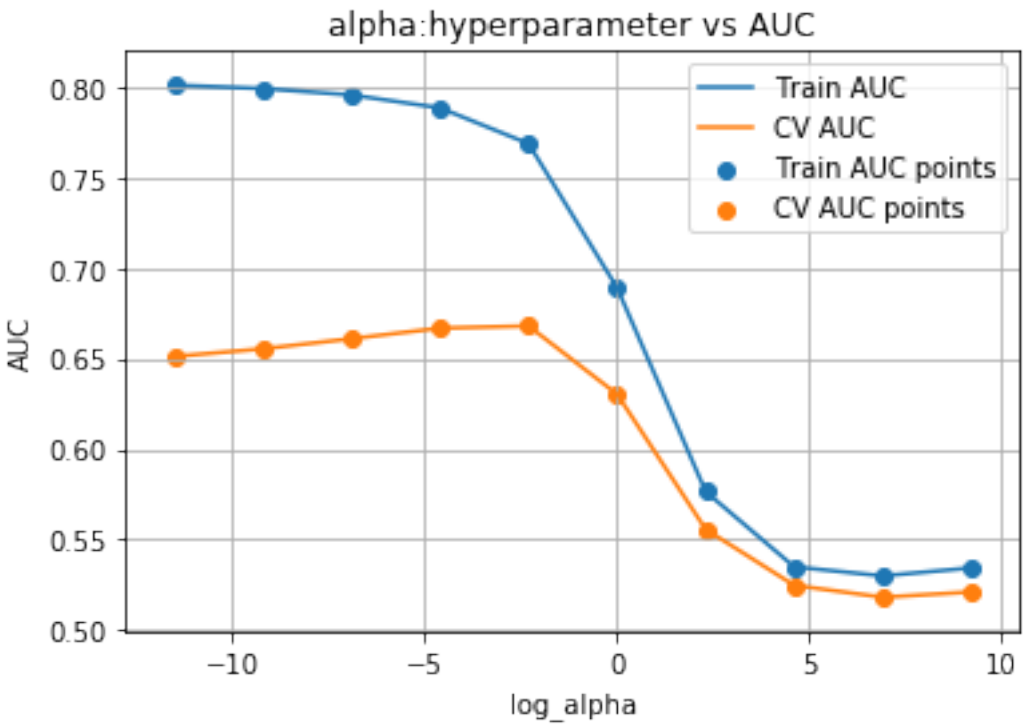
Appling Multinomial NB on Set 2 (Essay-TFIDF)

0.2.1 Hyper parameter Tuning using simple cross validation (using for loop)

[25]:
```
import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []
alpha_values =[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]  # distinct␣
 ↪alpha values
log_alpha=[]
for i in range(len(alpha_values)):
```

```
    log_alpha.append(math.log(alpha_values[i]))
for i in tqdm(alpha_values):
    clf = MultinomialNB(alpha=i,class_prior=[0.5,0.5])  #applying multinomial␣
 ↪NB on different alpha values
    clf.fit(X_train_tfidf,y_train) #fitting datamodel
    y_train_tfidf_pred = batch_predict(clf,X_train_tfidf)
    y_cv_tfidf_pred = batch_predict(clf,X_cv_tfidf)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability␣
 ↪estimates of the positive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_tfidf_pred)) #roc_auc_score␣
 ↪of train and cv dataponts for different values of alpha
    cv_auc.append(roc_auc_score(y_cv,y_cv_tfidf_pred))


plt.plot(log_alpha,train_auc,label='Train AUC') #plotting train and cv␣
 ↪auc_scores for different values of alpha
plt.plot(log_alpha,cv_auc,label='CV AUC')
plt.scatter(log_alpha,train_auc,label='Train AUC points')
plt.scatter(log_alpha,cv_auc,label='CV AUC points')
plt.legend()
plt.xlabel("log_alpha")
plt.ylabel("AUC")
plt.title("alpha:hyperparameter vs AUC")
plt.grid()
plt.show()
```

```
100%|
    | 10/10 [00:01<00:00,  5.59it/s]
```



finding best alpha:smoothing parameter from above plot

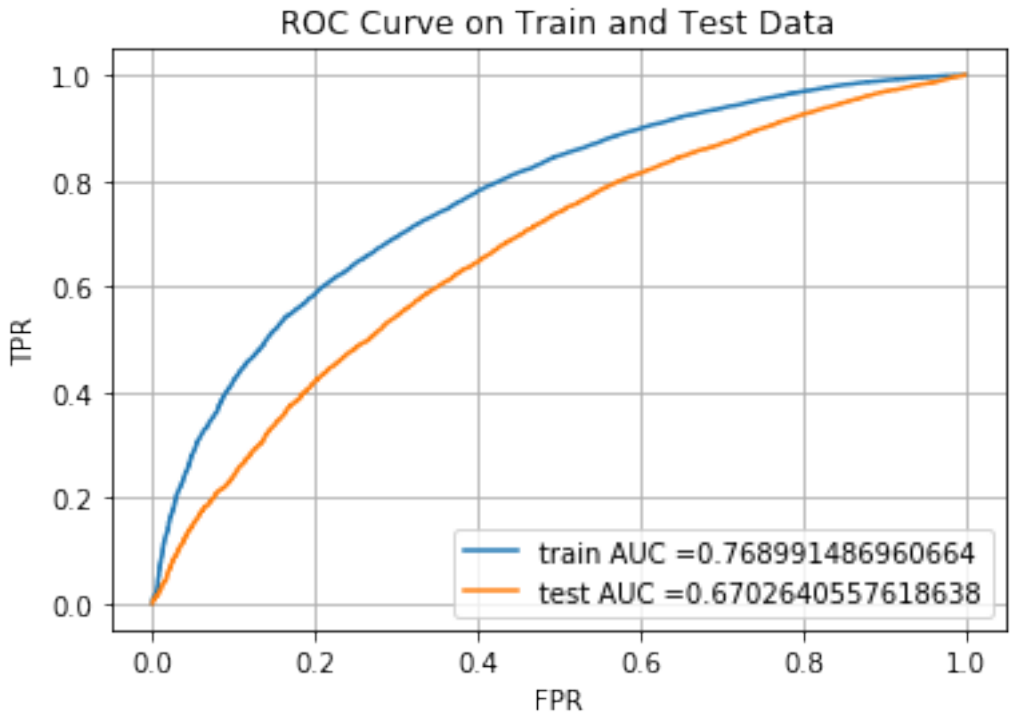
[26]:
```
best_alpha=0.1
print("best alpha(Hyper Parameter) which gives maximum AUC is {}".
 ↪format(best_alpha))
```

```
best alpha(Hyper Parameter) which gives maximum AUC is 0.1
```

0.3 Testing the performance of the model on test data , plotting ROC Curves

[27]:
```
from sklearn.metrics import roc_curve, auc


tfidf_bestclf=MultinomialNB(alpha=best_alpha,class_prior=[0.5,0.5])  #applying␣
 ↪multinomialNB for best alpha
tfidf_bestclf.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

y_train_pred = batch_predict(tfidf_bestclf,X_train_tfidf)
y_test_pred = batch_predict(tfidf_bestclf,X_test_tfidf)

train_fpr,train_tpr,train_thresolds = roc_curve(y_train, y_train_pred)
```

```
test_fpr,test_tpr, test_thresolds = roc_curve(y_test, y_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)))␣
 ↪#plotting ROC Curves for train and test datasets
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve on Train and Test Data")
plt.grid()
plt.show()
```



[28]:
```
print("AUC score on Train data is ")
print(auc(train_fpr, train_tpr))
print("AUC score on Test data is ")
print(auc(test_fpr, test_tpr))
```

```
AUC score on Train data is
0.768991486960664
AUC score on Test data is
0.6702640557618638
```

[29]:
```
from sklearn.metrics import confusion_matrix
best_t = find_best_thresold(train_thresolds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) is  0.4862196104047014 for thresold 0.503
Train confusion matrix
[[ 5230  2196]
 [12885 28730]]
Test confusion matrix
[[ 3156  2303]
 [10119 20474]]
```

finding the top 20 features Set 1 using feature_log_prob_ parameter of MultinomialNB

[30]:
```
features_names=[]
features_names.extend(vectorizer_state.get_feature_names())
features_names.extend(vectorizer_prefix.get_feature_names())
features_names.extend(vectorizer_grade.get_feature_names())
features_names.extend(vectorizer_categories.get_feature_names())
features_names.extend(vectorizer_subcategories.get_feature_names())
features_names.extend(vectorizer_essaybow.get_feature_names())
features_names.extend(['price'])
features_names.extend(['teacher_number_of_previously_posted_projects'])
print(len(features_names))
```

```
12217
```

[31]:
```
max_ind_neg=np.argsort((bow_bestclf.feature_log_prob_)[0])[::-1][0:10]
top_neg=np.take(features_names,max_ind_neg)
max_ind_pos=np.argsort((bow_bestclf.feature_log_prob_)[1])[::-1][0:10]
top_pos=np.take(features_names,max_ind_pos)
print(top_neg)
print(top_pos)
```

```
['students' 'school' 'learning' 'my' 'classroom' 'not' 'learn' 'they'
 'the' 'help']
['students' 'school' 'my' 'learning' 'classroom' 'the' 'not' 'they'
 'learn' 'help']
```

Summary

[32]:
```
#https://stackoverflow.com/questions/9535954/printing-lists-as-tabular-data
#https://pypi.python.org/pypi/PrettyTable

from prettytable import PrettyTable
t = PrettyTable(['Vectorizer', 'Model','Hyper Parameter','AUC'])
```

```
t.add_row(['BOW', 'Navie Bayes',0.1,0.7011])
t.add_row(['TFIDF','Navie Bayes',0.1,0.6702])
print(t)
```

```
+------------+-------------+-----------------+--------+
| Vectorizer |    Model    | Hyper Parameter |  AUC   |
+------------+-------------+-----------------+--------+
|    BOW     | Navie Bayes |       0.1       | 0.7011 |
|   TFIDF    | Navie Bayes |       0.1       | 0.6702 |
+------------+-------------+-----------------+--------+
```