Assignment 6: Apply NB

1. **Apply Multinomial NB on these feature sets**

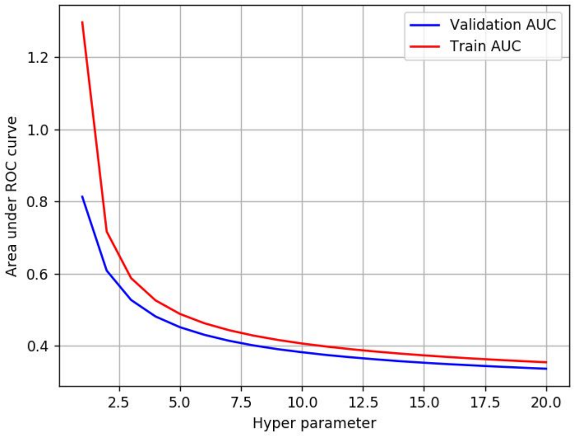
    - Set 1: categorical, numerical features + preprocessed_eassay (BOW)
    - Set 2: categorical, numerical features + preprocessed_eassay (TFIDF)

2. **The hyper paramter tuning(find best alpha:smoothing parameter)**

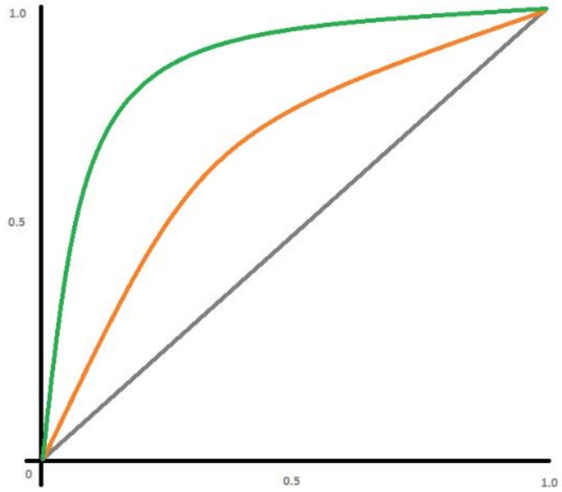
    - Find the best hyper parameter which will give the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
    - find the best hyper paramter using k-fold cross validation(use GridsearchCV or RandomsearchCV)/simple cross validation data (write for loop to iterate over hyper parameter values)
    -

3. **Representation of results**

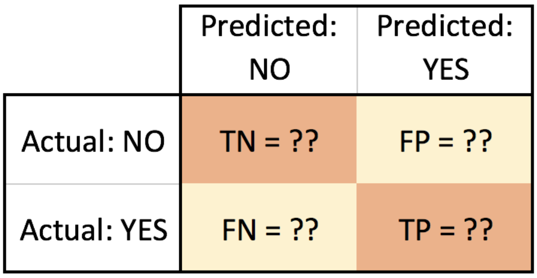
    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



    - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.

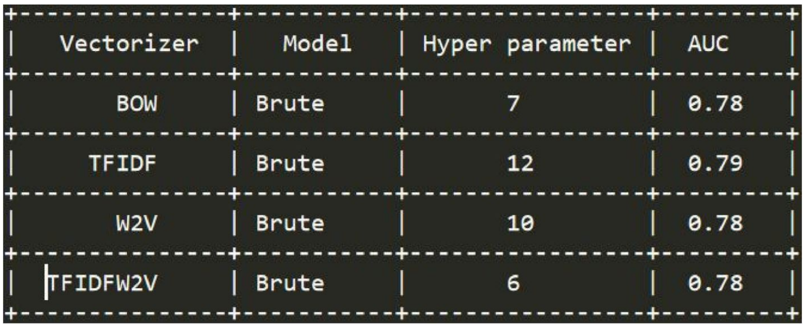


    - Along with plotting ROC curve, you need to print the confusion matrix (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points



	Predicted: NO	Predicted: YES
Actual: NO	TN = ??	FP = ??
Actual: YES	FN = ??	TP = ??

4. fine the top 20 features from either from feature Set 1 or feature Set 2 using absolute values of `feature_log_prob_ ` parameter of `MultinomialNB` (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html) and print their corresponding feature names
5. You need to summarize the results at the end of the notebook, summarize it in the table format



```
+---------------+----------+-----------------+---------+
|  Vectorizer   |  Model   | Hyper parameter |  AUC    |
+---------------+----------+-----------------+---------+
|      BOW      | Brute    |        7        |  0.78   |
+---------------+----------+-----------------+---------+
|     TFIDF     | Brute    |       12        |  0.79   |
+---------------+----------+-----------------+---------+
|      W2V      | Brute    |       10        |  0.78   |
+---------------+----------+-----------------+---------+
|   TFIDFW2V    | Brute    |        6        |  0.78   |
+---------------+----------+-----------------+---------+
```

2. Naive Bayes

In [1]:

```
#Importing libraries

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
# Tutorial about Python regular expressions: https://pymotw.com/2/re/

import pickle
from tqdm import tqdm
import os

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

1.1 Loading Data

In [36]:

```
data  = pd.read_csv('preprocessed_data.csv',nrows=60000)
data.head(5)
```

Out[36]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_pr
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	

In [39]:

```
data.columns
```

Out[39]:

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

In [4]:

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

Out[4]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_pr
0	ca	mrs	grades_prek_2	

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

In [5]:

```
# 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=0.30, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.30, stratify=
```

1.3 Make Data Model Ready: encoding eassay

Bow featurization of eassy feature

In [6]:

```
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)


vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.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.transform(X_train['essay'].values)
X_cv_essay_bow = vectorizer.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
print("="*100)
```

```
(29400, 8) (29400,)
(12600, 8) (12600,)
(18000, 8) (18000,)
============================================================================
========================
After vectorizations
(29400, 5000) (29400,)
(12600, 5000) (12600,)
(18000, 5000) (18000,)
============================================================================
========================
```

Tfidf featurization of eassy feature

In [7]:

```
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4),max_features=5000)
text_tfidf = vectorizer.fit(X_train['essay'].values)

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

print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
essay_features_tfidf = vectorizer.get_feature_names()
```

```
(29400, 8) (29400,)
(12600, 8) (12600,)
(18000, 8) (18000,)
============================================================================
========================
After vectorizations
(29400, 5000) (29400,)
(12600, 5000) (12600,)
(18000, 5000) (18000,)
============================================================================
========================
```

In [ ]:

```

```

1.4 Make Data Model Ready: encoding numerical, categorical features

1.4.1 encoding categorical features: School State

In [9]:

```
vectorizer = CountVectorizer()
vectorizer.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_ohe = vectorizer.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
school_state_features = vectorizer.get_feature_names()
```

```
After vectorizations
(29400, 51) (29400,)
(12600, 51) (12600,)
(18000, 51) (18000,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'i
a', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo',
'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'o
r', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv',
'wy']
============================================================================
========================
```

In [ ]:

```

```

1.4.2 encoding categorical features: teacher_prefix

In [10]:

```
vectorizer = CountVectorizer()
vectorizer.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_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_cv_teacher_ohe.shape, y_cv.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
teacher_prefix_features = vectorizer.get_feature_names()
```

```
After vectorizations
(29400, 5) (29400,)
(12600, 5) (12600,)
(18000, 5) (18000,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
============================================================================
========================
```

1.4.3 encoding categorical features: project_grade_category

In [11]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only on train

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].values)
X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].values)

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
project_grade_features = vectorizer.get_feature_names()
```

```
After vectorizations
(29400, 4) (29400,)
(12600, 4) (12600,)
(18000, 4) (18000,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
============================================================================
========================
```

1.4.4 encoding categorical features: clean_categories

In [12]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_categories_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_cv_categories_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_categories_ohe = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_categories_ohe.shape, y_train.shape)
print(X_cv_categories_ohe.shape, y_cv.shape)
print(X_test_categories_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
categories_features = vectorizer.get_feature_names()
```

```
After vectorizations
(29400, 9) (29400,)
(12600, 9) (12600,)
(18000, 9) (18000,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'liter
acy_language', 'math_science', 'music_arts', 'specialneeds', 'warmth']
============================================================================
========================
```

1.4.5 encoding categorical features: clean_subcategories

In [13]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen only on train dat

# we use the fitted CountVectorizer to convert the text to vector
X_train_subcategories_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
X_cv_subcategories_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_subcategories_ohe = vectorizer.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_subcategories_ohe.shape, y_train.shape)
print(X_cv_subcategories_ohe.shape, y_cv.shape)
print(X_test_subcategories_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
subcategories_features = vectorizer.get_feature_names()
```

```
After vectorizations
(29400, 30) (29400,)
(12600, 30) (12600,)
(18000, 30) (18000,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_governmen
t', 'college_careerprep', 'communityservice', 'earlydevelopment', 'economic
s', 'environmentalscience', 'esl', 'extracurricular', 'financialliteracy',
'foreignlanguages', 'gym_fitness', 'health_lifescience', 'health_wellness',
'history_geography', 'literacy', 'literature_writing', 'mathematics', 'musi
c', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 's
ocialsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
============================================================================
========================
```

1.4.6 encoding numerical features: Price

In [14]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# 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_norm = normalizer.transform(X_train['price'].values.reshape(-1,1))
X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(-1,1))



print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(29400, 1) (29400,)
(12600, 1) (12600,)
(18000, 1) (18000,)
============================================================================
========================
```

1.4.7 encoding numerical features: teacher_number_of_previously_posted_projects

In [15]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# 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['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)

X_train_projects_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_p
X_cv_projects_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_project
X_test_projects_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_pro



print("After vectorizations")
print(X_train_projects_norm.shape, y_train.shape)
print(X_cv_projects_norm.shape, y_cv.shape)
print(X_test_projects_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(29400, 1) (29400,)
(12600, 1) (12600,)
(18000, 1) (18000,)
============================================================================
========================
```

1.4.8 Concatinating all the features

Set1: Features:categorical, numerical features + preprocessed_eassay (BOW)

In [16]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_bow = hstack((X_train_essay_bow, X_train_state_ohe, X_train_teacher_ohe, X_train_grade
X_cr_bow = hstack((X_cv_essay_bow, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_c
X_te_bow = hstack((X_test_essay_bow, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe

print("Final Data matrix")
print(X_tr_bow.shape, y_train.shape)
print(X_cr_bow.shape, y_cv.shape)
print(X_te_bow.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(29400, 5101) (29400,)
(12600, 5101) (12600,)
(18000, 5101) (18000,)
============================================================================
========================
```

Set2 : Features:categorical, numerical features + preprocessed_eassay (TFIDF)

In [17]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_tfidf = hstack((X_train_essay_tfidf, X_train_state_ohe, X_train_teacher_ohe, X_train_g
X_cr_tfidf = hstack((X_cv_essay_tfidf, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_
X_te_tfidf = hstack((X_test_essay_tfidf, X_test_state_ohe, X_test_teacher_ohe, X_test_grade

print("Final Data matrix")
print(X_tr_tfidf.shape, y_train.shape)
print(X_cr_tfidf.shape, y_cv.shape)
print(X_te_tfidf.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(29400, 5101) (29400,)
(12600, 5101) (12600,)
(18000, 5101) (18000,)
============================================================================
========================
```

1.5 Appling NB on different kind of featurization as mentioned in the instructions

1.5.1 Hyper parameter tuning of Set1 features

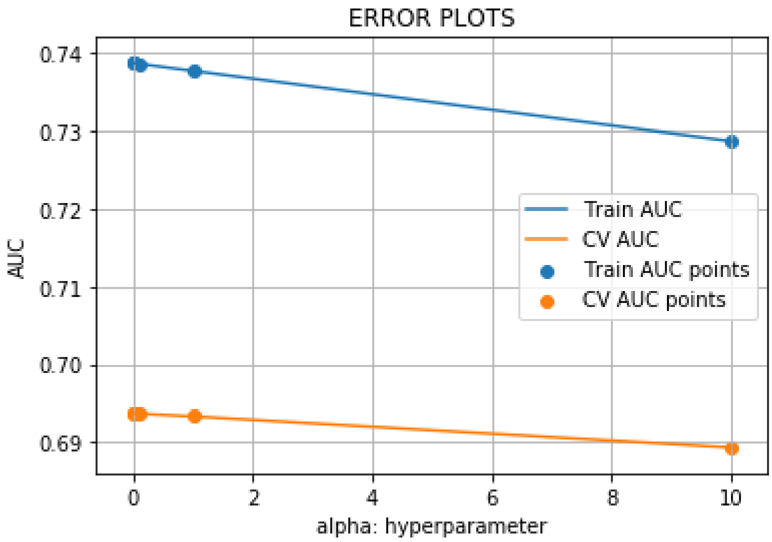
In [55]:

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # not the predicted outputs

    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 = 4900
    # in this for loop we will iterate unti 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
```

In [56]:

```
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score


train_auc = []
cv_auc = []
alpha = [0.00001, 0.0001, 0.001, 0.1, 1,10]
for i in tqdm(alpha):

    clf = MultinomialNB(alpha=i)
    clf.fit(X_tr_bow, y_train)

    y_train_pred = batch_predict(clf, X_tr_bow)
    y_cv_pred = batch_predict(clf, X_cr_bow)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(alpha, train_auc, label='Train AUC')
plt.plot(alpha, cv_auc, label='CV AUC')

plt.scatter(alpha, train_auc, label='Train AUC points')
plt.scatter(alpha, cv_auc, label='CV AUC points')

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

```
100%|██████████| 6/6 [00:01<00:00,  3.27it/s]
```

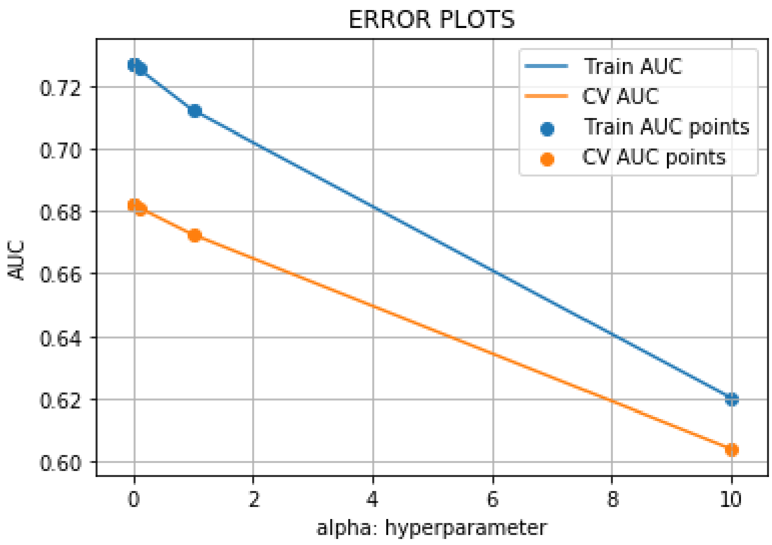


1.5.1 Hyper parameter tuning of Set2 features

In [57]:

```
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score


train_auc = []
cv_auc = []
alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10]
for i in tqdm(alpha):

    clf = MultinomialNB(alpha=i)
    clf.fit(X_tr_tfidf, y_train)
    y_train_pred = batch_predict(clf, X_tr_tfidf)
    y_cv_pred = batch_predict(clf, X_cr_tfidf)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(alpha, train_auc, label='Train AUC')
plt.plot(alpha, cv_auc, label='CV AUC')

plt.scatter(alpha, train_auc, label='Train AUC points')
plt.scatter(alpha, cv_auc, label='CV AUC points')

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

```
100%|██████████| 6/6 [00:01<00:00,  3.97it/s]
```



In [58]:

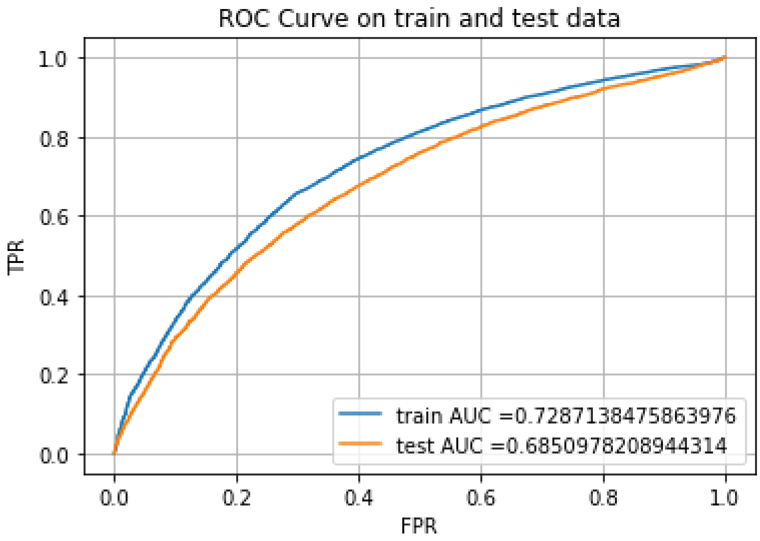
```
#from the error plot we choose K such that, we will have maximum AUC on cv data and gap bet
#Best alpha in for both set1 and set2 features is
best_alpha = 10
```

1.5.2Testing the performance of the model on test data, plotting ROC Curves

Testing the performance of the model with Set1 features

In [59]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.
from sklearn.metrics import roc_curve, auc


clf = MultinomialNB(alpha=best_alpha)
clf.fit(X_tr_bow, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the p
# not the predicted outputs

y_train_pred = batch_predict(clf, X_tr_bow)
y_test_pred = batch_predict(clf, X_te_bow)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = 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)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve on train and test data")
plt.grid()
plt.show()
```



In [60]:

```
# 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_threshold(threshould, fpr, tpr):
    t = threshould[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)", max(tpr*(1-fpr)), "for threshold", np.round(t
    return t

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

In [61]:

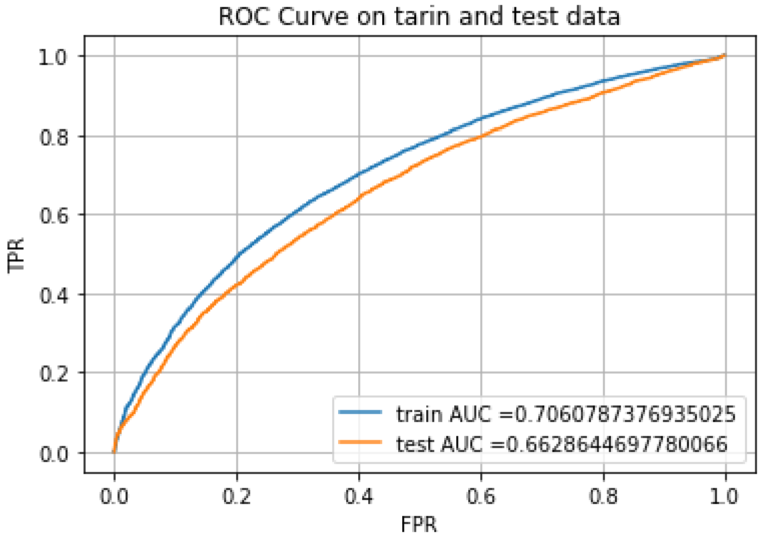
```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
============================================================================
========================
the maximum value of tpr*(1-fpr) 0.4606752373356364 for threshold 0.933
Train confusion matrix
[[ 3279  1393]
 [ 8497 16231]]
Test confusion matrix
[[1814 1047]
 [5366 9773]]
```

Testing the performance of the model with Set2 features

In [62]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.
from sklearn.metrics import roc_curve, auc


clf = MultinomialNB(alpha= best_alpha)
clf.fit(X_tr_bow, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the p
# not the predicted outputs

y_train_pred = batch_predict(clf, X_tr_tfidf)
y_test_pred = batch_predict(clf, X_te_tfidf)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = 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)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve on tarin and test data")
plt.grid()
plt.show()
```



In [63]:

```
# 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_threshold(threshould, fpr, tpr):
    t = threshould[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)", max(tpr*(1-fpr)), "for threshold", np.round(t
    return t

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

In [64]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, 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) 0.42947079291181206 for threshold 0.811
Train confusion matrix
[[ 3172  1500]
 [ 9086 15642]]
Test confusion matrix
[[1762 1099]
 [5751 9388]]
```

2. Finding Top 20 Features among Set 2 features

In [65]:

```
# Getting all feature names
All_features = list(essay_features_tfidf)
All_features.extend(list(school_state_features))
All_features.extend(list(teacher_prefix_features))
All_features.extend(list(project_grade_features))
All_features.extend(list(categories_features))
All_features.extend(list(subcategories_features))
All_features.append("previously_posted_projects")
All_features.append("price")
All_features
print(len(All_features))
```

```
5101
```

In [69]:

```
# Geeting top 20 features for class 0 and class 1
class0_probs = clf.feature_log_prob_[0, :]
class1_probs = clf.feature_log_prob_[1, :]
# Absolute values of probabilities
class0_probs = list(abs(class0_probs))
class1_probs = list(abs(class1_probs))

#Getting indices of top 20 features
top_20_indices_class0 = sorted(range(len(class0_probs)), key=lambda i: class0_probs[i], rev
top_20_indices_class0 = top_20_indices_class0[0:20]

top_20_indices_class1 = sorted(range(len(class1_probs)), key=lambda i: class1_probs[i], rev
top_20_indices_class1 = top_20_indices_class1[0:20]

# getting top 20 features of class 1
class0_top_20 = []
for i in top_20_indices_class0:
    class0_top_20.append(All_features[i])

# getting top 20 features of class 2
class1_top_20 = []
for i in top_20_indices_class1:
    class1_top_20.append(All_features[i])


print(class0_top_20)
print("*"*120)
print(class1_top_20)
```

```
['care_hunger', 'warmth', 'care_hunger', 'warmth', 'dr', 'vt', 'nd', 'boogi
e', 'wy', 'book bins', 'wobble cushions', '3d printer', 'economics', 'pro',
'the wobble', 'volleyball', 'allow students move', 'dry erase boards', 'easy
access', 'erase boards']
****************************************************************************
********************************************
['care_hunger', 'warmth', 'care_hunger', 'warmth', 'dr', 'wy', 'vt', 'nd',
'financialliteracy', 'ri', 'economics', 'mt', 'parentinvolvement', 'sd', 'a
k', 'ne', 'nh', 'de', 'hi', 'communityservice']
```

3. Summary

Vectorizer	Model	Hyperparameter	AUC
BOW	MultinomialNB	10	0.68
TFIDF	MultinomialNB	10	0.66

In [ ]:

```
```