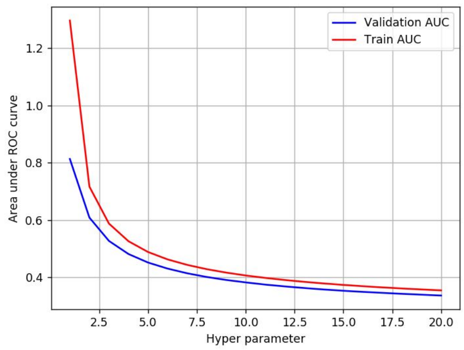
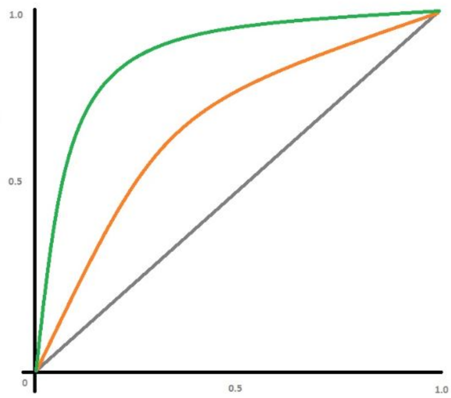
Assignment 6: Apply NB

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 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 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

1.1 Loading Data

In [52]:

```
%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
# 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
```

In [53]:

```
import pandas
data = pandas.read_csv('preprocessed_data.csv')
```

In [54]:

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

Out[54]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcate
0	ca	mrs	grades_prek_2	53	math_science	appliedsc health_lifes

In [55]:

```
print(data.shape)
print(X.shape)
print(y.shape)
```

```
(109248, 9)
(109248, 8)
(109248,)
```

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

In [56]:

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

1.3 Make Data Model Ready: encoding eassay, and project_title

preprocessed_eassay (BOW)

In [57]:

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

```
(49041, 8) (49041,)
(24155, 8) (24155,)
(36052, 8) (36052,)
```

In [63]:

```
vectorizer_essay = CountVectorizer(min_df=10,ngram_range=(1,4))
vectorizer_essay.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_essay.transform(X_train['essay'].values)
X_cv_essay_bow = vectorizer_essay.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer_essay.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)
```

```
After vectorizations
(49041, 166174) (49041,)
(24155, 166174) (24155,)
(36052, 166174) (36052,)
====================================================================================================
```

1.4 Make Data Model Ready: encoding numerical, categorical features

1.4.1 encoding categorical features: School State

In [64]:

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

```
After vectorizations
(49041, 51) (49041,)
(24155, 51) (24155,)
(36052, 51) (36052,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
====================================================================================================
```

1.4.2 encoding categorical features: teacher_prefix

In [65]:

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

```
After vectorizations
(49041, 5) (49041,)
(24155, 5) (24155,)
(36052, 5) (36052,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
====================================================================================================
```

1.4.3 encoding categorical features: project_grade_category

In [66]:

```
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_ohe = vectorizer_grade.transform(X_train['project_grade_category'].values)
X_cv_grade_ohe = vectorizer_grade.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer_grade.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)
```

```
After vectorizations
(49041, 4) (49041,)
(24155, 4) (24155,)
(36052, 4) (36052,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
====================================================================================================
```

1.4.4 encoding categorical features: clean_categories

In [69]:

```
vectorizer_cat = CountVectorizer()
vectorizer_cat.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_cat.transform(X_train['clean_categories'].values)
X_cv_categories_ohe = vectorizer_cat.transform(X_cv['clean_categories'].values)
X_test_categories_ohe = vectorizer_cat.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)
```

```
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']
====================================================================================================
```

1.4.4 encoding subcategorical features: clean_subcategories

In [68]:

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_subcategories_ohe = vectorizer_subcat.transform(X_train['clean_subcategories'].values)
X_cv_subcategories_ohe = vectorizer_subcat.transform(X_cv['clean_subcategories'].values)
X_test_subcategories_ohe = vectorizer_subcat.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_subcat.get_feature_names())
print("="*100)
```

```
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', 'm
athematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socia
lsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
====================================================================================================
```

1.4.5 encoding numerical features: Price

In [70]:

```
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
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
====================================================================================================
```

1.4.7 encoding numerical features: teacher_number_of_previously_posted_projects

In [71]:

```
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
X_train_prev_proj_norm =
normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_cv_prev_proj_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].va
lues.reshape(-1,1))
X_test_prev_proj_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'
].values.reshape(-1,1))


print("After vectorizations")
print(X_train_prev_proj_norm.shape, y_train.shape)
print(X_cv_prev_proj_norm.shape, y_cv.shape)
print(X_test_prev_proj_norm.shape, y_test.shape)

# only one sample of price
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
```

Concatinating all the features

In [72]:

```
# 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_ohe,
X_train_price_norm,X_train_categories_ohe,X_train_subcategories_ohe,X_train_prev_proj_norm)).tocsr
()
X_cv_bow = hstack((X_cv_essay_bow, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_price_nor
m,X_cv_categories_ohe,X_cv_subcategories_ohe,X_cv_prev_proj_norm)).tocsr()
X_te_bow = hstack((X_test_essay_bow, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test
_price_norm, X_test_categories_ohe,X_test_subcategories_ohe,X_test_prev_proj_norm)).tocsr()

print("Final Data matrix")
print(X_tr_bow.shape, y_train.shape)
```

```
print(X_cv_bow.shape, y_cv.shape)
print(X_te_bow.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(49041, 166275) (49041,)
(24155, 166275) (24155,)
(36052, 166275) (36052,)
====================================================================================================
```

preprocessed_eassay (TFIDF)

In [74]:

```
vectorizer_tfidf = TfidfVectorizer(min_df=10,ngram_range=(1,4))
vectorizer_tfidf.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_tfidf = vectorizer_tfidf.transform(X_train['essay'].values)
X_cv_essay_tfidf = vectorizer_tfidf.transform(X_cv['essay'].values)
X_test_essay_tfidf = vectorizer_tfidf.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)
```

```
After vectorizations
(49041, 166174) (49041,)
(24155, 166174) (24155,)
(36052, 166174) (36052,)
====================================================================================================
```

Concatinating all the features

In [77]:

```
# 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_grade_ohe,
X_train_price_norm,X_train_categories_ohe,X_train_subcategories_ohe,X_train_prev_proj_norm)).tocsr
()
X_cv_tfidf = hstack((X_cv_essay_tfidf, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_price
_norm,X_cv_categories_ohe,X_cv_subcategories_ohe,X_cv_prev_proj_norm)).tocsr()
X_te_tfidf = hstack((X_test_essay_tfidf, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_
test_price_norm, X_test_categories_ohe,X_test_subcategories_ohe,X_test_prev_proj_norm)).tocsr()

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

```
Final Data matrix
(49041, 166275) (49041,)
(24155, 166275) (24155,)
(36052, 166275) (36052,)
====================================================================================================
```

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

Apply NB on different kind of featurization as mentioned in the instructions

For Every model that you work on make sure you do the step 2 and step 3 of instrucations

1.5.3.1 Hyper parameter Tuning

1.5.1.1.1 Method 1: Simple for loop (if you are having memory limitations use this)

In [78]:

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

train_auc = []
cv_auc = []
alpha = [0.0001,0.001,0.01,0.1,1,10,100,1000] # the typical range lies between 10^-4 to 10^4 in mul
tiples of 10
for i in tqdm(alpha):
    nb = MultinomialNB(alpha=i, class_prior=[0.5,0.5]) # sum of the priors should be 1
    nb.fit(X_tr_bow, y_train)

    y_train_pred = nb.predict_proba(X_tr_bow)[:,1]  # since we want only positive values[:,1]
    y_cv_pred = nb.predict_proba(X_cv_bow)[:,1]

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # 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))

# How to apply class_prior and fit_prior for an imbalanced dataset
# class_prior = [0.5,0.5] and fit_prior = False should be the setting in order to tackle the data
imbalance

# we dont have to do over sampling or undersampling as we have already given class prior?
# we can give class prior [1,1] or [2,2] or anything just both values should be same so that it in
directly cancels the effect
# of prior probability right?
# fit_prior calculates prior probabilities for us
# class_prior is not provided by user explicitly
```

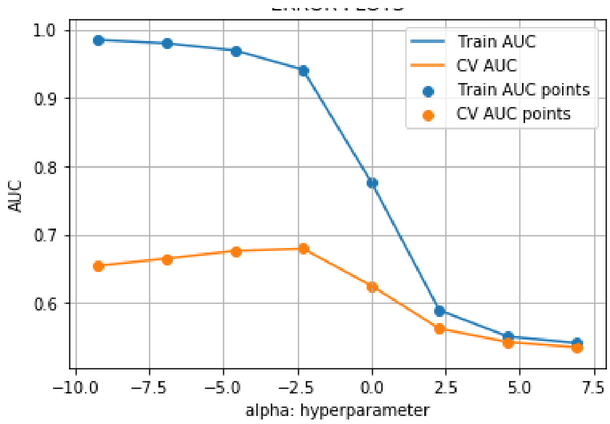
```
100%|██████████| 8/8 [00:01<00:00,  5.64it/s]
```

In [79]:

```
import math
log_alpha = []
for i in tqdm(alpha):
    log_alpha.append(math.log(i)) # if you plot directly values overlap each other since they are
so less
                                # in order to see distinctly we use log on x axis

plt.plot(log_alpha, train_auc, label='Train AUC')
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("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████| 8/8 [00:00<?, ?it/s]
```

ERROR PLOTS



15.1.1.2 Method 2: Random Search or Grid Search

In [80]:

```
# RandomSearchCv
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV

nb = MultinomialNB()
parameters = {'alpha':[0.0001,0.001,0.01,0.1,1,10,100,1000] }
clf = GridSearchCV(nb, parameters, cv=3, scoring='roc_auc',return_train_score=True)

clf.fit(X_tr_bow, y_train)
```

Out[80]:

```
GridSearchCV(cv=3, error_score=nan,
             estimator=MultinomialNB(alpha=1.0, class_prior=None,
                                     fit_prior=True),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [84]:

```
results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_alpha'])
print('best alpha value:',clf.best_params_['alpha'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
K =  results['params']
```

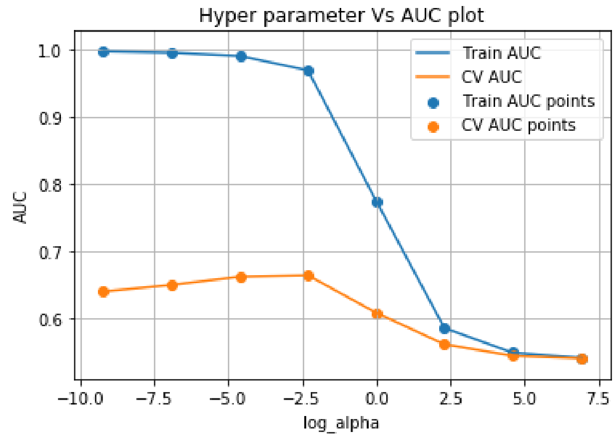
```
best alpha value: 0.1
```

In [85]:

```
import math
log_alpha = []
for i in tqdm(alpha):
    log_alpha.append(math.log(i))

plt.plot(log_alpha, train_auc, label='Train AUC')
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("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```

```
100%|██████████| 8/8 [00:00<00:00, 8017.79it/s]
```



Out[85]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	split2_test_
0	0.114353	0.005561	0.028177	0.004281	0.0001	{'alpha': 0.0001}	0.640799	0.638729	0.6
1	0.106432	0.000931	0.024743	0.000258	0.001	{'alpha': 0.001}	0.650775	0.647289	0.6
2	0.109715	0.000805	0.025920	0.000015	0.01	{'alpha': 0.01}	0.663085	0.657560	0.6
3	0.111047	0.006180	0.024938	0.000008	0.1	{'alpha': 0.1}	0.665952	0.656733	0.6
4	0.107730	0.001435	0.025924	0.000814	1	{'alpha': 1}	0.614305	0.600155	0.6

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

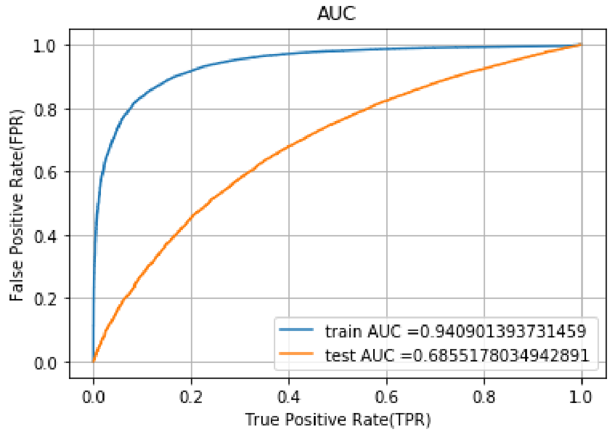
In [88]:

```
best_a = 0.1
```

In [118]:

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


nb = MultinomialNB(alpha=best_a, class_prior=[0.5,0.5])
nb.fit(X_tr_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 = nb.predict_proba(X_tr_bow)[:,1]
y_test_pred = nb.predict_proba(X_te_bow)[:,1]

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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



In [90]:

```
# 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,3))
    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 [91]:

```
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) 0.7562537190608679 for threshold 0.505
Train confusion matrix
[[ 6538   888]
 [ 5869 35746]]
Test confusion matrix
[[ 2240  3219]
 [ 5593 25000]]
```

Hyper Parameter tuning for TFIDF

Method : 1 : Simple for loop (if you are having memory limitations use this)

In [92]:

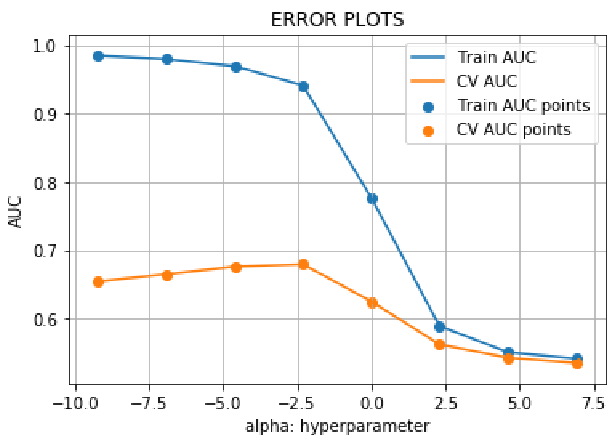
```
import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []
alpha = [0.0001,0.001,0.01,0.1,1,10,100,1000] # the typical range lies between 10^-4 to 10^4 in mul
tiples of 10
for i in tqdm(alpha):
    nb = MultinomialNB(alpha=i, class_prior=[0.5,0.5]) # sum of the priors should be 1
    nb.fit(X_tr_tfidf, y_train)

    y_train_pred = nb.predict_proba(X_tr_tfidf)[:,1]  # since we want only positive values[:,1]
    y_cv_pred = nb.predict_proba(X_cv_tfidf)[:,1]

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # 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))

# How to apply class_prior and fit_prior for an imbalanced dataset
# class_prior = [0.5,0.5] and fit_prior = False should be the setting in order to tackle the data
imbalance

# we dont have to do over sampling or undersampling as we have already given class prior?
# we can give class prior [1,1] or [2,2] or anything just both values should be same so that it in
directly cancels the effect
# of prior probability right?
# fit_prior calculates prior probabilities for us
# class_prior is not provided by user explicitly
```

```
100%|██████████| 8/8 [00:01<00:00,  5.62it/s]
```

In [93]:

```
from math import log
log_alpha = []
for i in tqdm(alpha):
    log_alpha.append(math.log(i))

plt.plot(log_alpha, train_auc, label='Train AUC')
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("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████| 8/8 [00:00<?, ?it/s]
```



Method 2: Random Search or Grid Search

In [94]:

```
# RandomSearchCv
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV

nb = MultinomialNB()
parameters = {'alpha':[0.0001,0.001,0.01,0.1,1,10,100,1000] }
clf = GridSearchCV(nb, parameters, cv=3, scoring='roc_auc',return_train_score=True)

clf.fit(X_tr_tfidf, y_train)
```

Out[94]:

```
GridSearchCV(cv=3, error_score=nan,
             estimator=MultinomialNB(alpha=1.0, class_prior=None,
                                     fit_prior=True),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [95]:

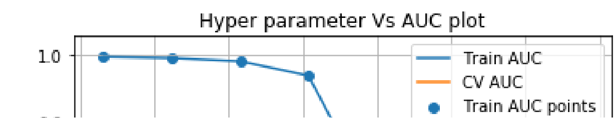
```
results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_alpha'])
print('best alpha value:',clf.best_params_['alpha'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
K =  results['params']
```

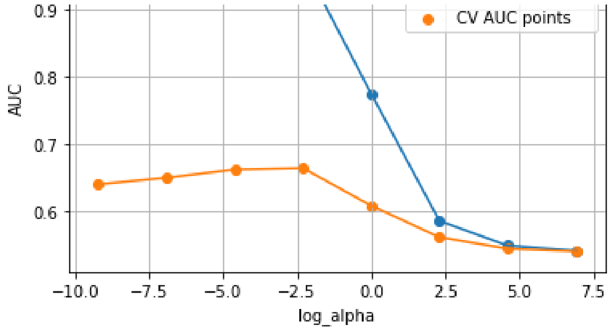
```
best alpha value: 0.1
```

In [96]:

```
import math
log_alpha = []
for i in tqdm(alpha):
    log_alpha.append(math.log(i))

plt.plot(log_alpha, train_auc, label='Train AUC')
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("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```

```
100%|██████████| 8/8 [00:00<?, ?it/s]
```





Out[96]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	split2_test_
0	0.106395	0.002364	0.024931	0.000019	0.0001	{'alpha': 0.0001}	0.640799	0.638729	0.6
1	0.105730	0.000821	0.025261	0.001245	0.001	{'alpha': 0.001}	0.650775	0.647289	0.6
2	0.110732	0.004957	0.024935	0.000001	0.01	{'alpha': 0.01}	0.663085	0.657560	0.6
3	0.106871	0.000622	0.024901	0.000023	0.1	{'alpha': 0.1}	0.665952	0.656733	0.6
4	0.109689	0.004218	0.025937	0.001406	1	{'alpha': 1}	0.614305	0.600155	0.6

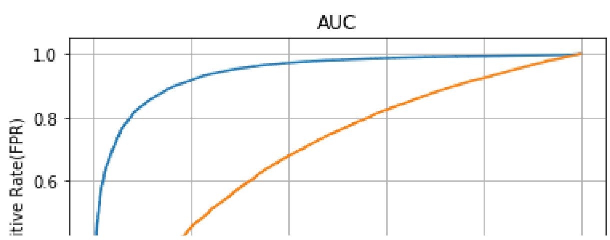
Testing the performance of the model on test data, plotting ROC Curves

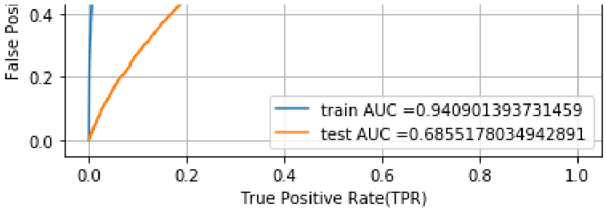
In [97]:

```
best_a = 0.1
```

In [117]:

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


nb = MultinomialNB(alpha=best_a, class_prior=[0.5,0.5])
nb.fit(X_tr_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 = nb.predict_proba(X_tr_tfidf)[:,1]
y_test_pred = nb.predict_proba(X_te_tfidf)[:,1]

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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```





In [99]:

```
# 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,3))
    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 [100]:

```
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) 0.7562537190608679 for threshold 0.505
Train confusion matrix
[[ 6538   888]
 [ 5869 35746]]
Test confusion matrix
[[ 2240  3219]
 [ 5593 25000]]
```

find the top 20 features from either from feature Set 1 or feature Set 2 using absolute values of `feature_log_prob_` parameter of `MultinomialNB`

In [101]:

```
print("Final Data-matrix:")
print(X_tr_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_te_bow.shape, y_test.shape)
```

```
Final Data-matrix:
(49041, 166275) (49041,)
(24155, 166275) (24155,)
(36052, 166275) (36052,)
```

In [102]:

```
#Step 2: Train the model with MultinomialNB

# Put the optimal alpha value you found
nb = MultinomialNB(alpha=1.0, class_prior=[0.5,0.5])
nb.fit(X_tr_bow, y_train)
```

Out[102]:

```
MultinomialNB(alpha=1.0, class_prior=[0.5, 0.5], fit_prior=True)
```

In [103]:

```
#Step 3: Get the features indices sorted by log-probability of features

# Here .argsort() will give indexes of features sorted with their log-probabilities

# For positive class
sorted_prob_class_1_ind = nb.feature_log_prob_[1, :].argsort()
# For negative class
sorted_prob_class_0_ind = nb.feature_log_prob_[0, :].argsort()
```

In [104]:

```
#Step 4: We got indexes of features, but we want names of features. So first get the list of all f
eatures from concatenating all the previously obtained vectorizers' features.

features_lst = list(vectorizer_essay.get_feature_names() + vectorizer_state.get_feature_names() +
\
                    vectorizer_prefix.get_feature_names() + vectorizer_grade.get_feature_names() +
\
                    ["teacher_number_of_previously_posted_projects"] +
vectorizer_cat.get_feature_names() + \
                   vectorizer_subcat.get_feature_names() + ["Price"])
```

In [105]:

```
Most_imp_words_1 = []
Most_imp_words_0 = []

for index in sorted_prob_class_1_ind[-20:-1]:
    Most_imp_words_1.append(features_lst[index])

for index in sorted_prob_class_0_ind[-20:-1]:
    Most_imp_words_0.append(features_lst[index])

print("20 most imp features for positive class:\n")
print(Most_imp_words_1)

print("\n" + "-"*100)

print("\n20 most imp features for negative class:\n")
print(Most_imp_words_0)
```

```
20 most imp features for positive class:

['mr', 'appliedsciences', 'grades_9_12', 'appliedlearning', 'specialneeds', 'specialneeds',
'health_sports', 'ca', 'grades_6_8', 'literature_writing', 'mathematics', 'literacy',
'grades_3_5', 'ms', 'math_science', 'grades_prek_2', 'literacy_language', 'mrs', 'Price']

----------------------------------------------------------------------------------------------------

20 most imp features for negative class:

['mr', 'appliedsciences', 'grades_9_12', 'appliedlearning', 'health_sports', 'ca', 'specialneeds',
'specialneeds', 'grades_6_8', 'literature_writing', 'mathematics', 'literacy', 'grades_3_5', 'ms',
'math_science', 'grades_prek_2', 'literacy_language', 'mrs', 'Price']
```

In [109]:

```
sorted_prob_class_1_ind = nb.feature_log_prob_[:].argsort()
```

In [110]:

```
#Step 4: We got indexes of features, but we want names of features. So first get the list of all f
eatures from concatenating all the previously obtained vectorizers' features.
```

```
features_lst = list(vectorizer_essay.get_feature_names() + vectorizer_state.get_feature_names() +
\
                    vectorizer_prefix.get_feature_names() + vectorizer_grade.get_feature_names() +
\
                    ["teacher_number_of_previously_posted_projects"] +
vectorizer_cat.get_feature_names() + \
                   vectorizer_subcat.get_feature_names() + ["Price"])
```

In [112]:

```
Most_imp_words = []
for index in sorted_prob_class_0_ind[-20:-1]:
    Most_imp_words.append(features_lst[index])
```

In [114]:

```
Most_imp_words
```

Out[114]:

```
['mr',
 'appliedsciences',
 'grades_9_12',
 'appliedlearning',
 'health_sports',
 'ca',
 'specialneeds',
 'specialneeds',
 'grades_6_8',
 'literature_writing',
 'mathematics',
 'literacy',
 'grades_3_5',
 'ms',
 'math_science',
 'grades_prek_2',
 'literacy_language',
 'mrs',
 'Price']
```

In [115]:

```
Most_imp_words[::-1]
```

Out[115]:

```
['Price',
 'mrs',
 'literacy_language',
 'grades_prek_2',
 'math_science',
 'ms',
 'grades_3_5',
 'literacy',
 'mathematics',
 'literature_writing',
 'grades_6_8',
 'specialneeds',
 'specialneeds',
 'ca',
 'health_sports',
 'appliedlearning',
 'grades_9_12',
 'appliedsciences',
 'mr']
```

Summary

In [119]:

```
# Compare all your models using Prettytable library
```

```python
#http://zetcode.com/python/prettytable/

from prettytable import PrettyTable

x_pretty_table = PrettyTable()
x_pretty_table.field_names = ["Model Type","Vectorizer", "Hyper Parameter - a","Train-AUC","Test-AUC"]

x_pretty_table.add_row(["Naive Bayes","BOW",0.1,0.94,0.68])
x_pretty_table.add_row([ "Naive Bayes", "TFIDF",0.1,0.94,0.68])

print(x_pretty_table)
```

```
+-------------+------------+---------------------+-----------+----------+
|  Model Type | Vectorizer | Hyper Parameter - a | Train-AUC | Test-AUC |
+-------------+------------+---------------------+-----------+----------+
| Naive Bayes |    BOW     |         0.1         |    0.94   |   0.68   |
| Naive Bayes |   TFIDF    |         0.1         |    0.94   |   0.68   |
+-------------+------------+---------------------+-----------+----------+
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```