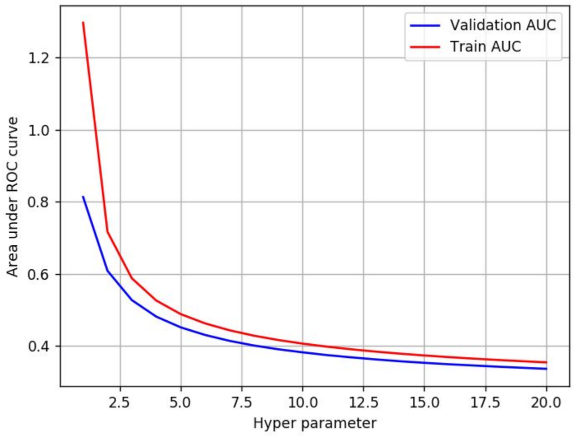
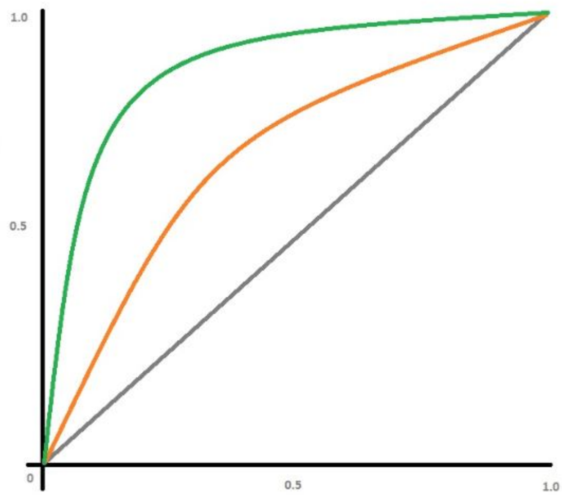
Assignment 6: Apply NB

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/)](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/)](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

1.1 Loading Data

In [1]:

```
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
from sklearn.preprocessing import Normalizer
from scipy.sparse import hstack
import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/

import pickle
from tqdm import tqdm
import os
from mlxtend.plotting import plot_confusion_matrix

#Required sklearn libraries
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB

import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
import pandas
data = pandas.read_csv('preprocessed_data.csv')
```

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

In [2]:

```
# please write all the code with proper documentation, and proper titles for each su
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging y
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

In [3]:

```
'''
    1.Data consits of 109248 records (rows) and 9 features (columns)
    2.Among 9 features 6 are categorical and 3 are numerical features
       - categorical features : school_state , teacher_prefix, project_grade_categor
       - numerical features   : teacher_number_of_previously_posted_projects,project
    3.The min,mean,max values of nuerical features are like below :
       - teacher_number_of_previously_posted_projects - (0.000000, 11.153165, 451.00
       - project_is_approved - (0.000000, 0.848583, 1.000000)
       - price - (0.660000, 298.119343, 9999.000000)
'''
data.head(10)
```

Out[3]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_proj
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	
5	ca	mrs	grades_3_5	
6	ca	mrs	grades_3_5	

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_proj
7	ca	ms	grades_3_5	
8	ca	ms	grades_prek_2	
9	hi	mrs	grades_3_5	

In [4]:

```
Y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
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)
'''
    First splitting the data in to train and test from original dataset
'''
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
'''
    Later splitting the data in to train and validation from train data
    Overal train + test + validation = total dataset
'''
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, st
print(X_train.shape, X_cv.shape, y_train.shape, y_cv.shape)
X_train.head(3)
```

```
(73196, 8) (36052, 8) (73196,) (36052,)
(49041, 8) (24155, 8) (49041,) (24155,)
```

Out[4]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted
26121	ca	ms	grades_6_8	
44153	ny	ms	grades_prek_2	
72189	ut	ms	grades_6_8	

Creating a function for encoding categorical features

In [5]:

```
'''
    Creating a custom function for getting the categorical features
    @function    : To retun the categorical features by using BOW
    @params      : [category_feature,train_data, cv_data, test_data]
    @returns     : train_data_features, cv_data_features, test_data_features
    @copyrights : Trinath Reddy
'''
def return_categorical_features(category_feature,train_data, cv_data, test_data):
    vectorizer = CountVectorizer()
    vectorizer.fit(train_data[category_feature].values)
    train_data_features = vectorizer.transform(train_data[category_feature].values)
    cv_data_features    = vectorizer.transform(cv_data[category_feature].values)
    test_data_features  = vectorizer.transform(test_data[category_feature].values)
    print('After Vectorization of {}'.format(category_feature))
    print(train_data_features.shape)
    print(cv_data_features.shape)
    print(test_data_features.shape)
    print('feature names for {}'.format(category_feature))
    print(vectorizer.get_feature_names())
    print("*"*50)
    print("\n")
    return {
             'train_data_features' : train_data_features,
             'cv_data_features'    : cv_data_features,
             'test_data_features'  : test_data_features,
             'feture_name': vectorizer.get_feature_names()
           }
```

Making categorical feature for:

- school_state
- teacher_prefix
- project_grade_category

In [6]:

```
categorical_columns = ['school_state','teacher_prefix','project_grade_category','cle
converted_categorical_features = {}
for current_feature in categorical_columns:
    print("*"*50)
    print('converting {} to categorical feature'.format(current_feature))
    converted_categorical_features[current_feature] = return_categorical_features(cu


# X_test_school_state_features, X_cv_school_state_features, X_test_school_state_feat
# converted_categorical_features.keys()
```

```
**************************************************
converting school_state to categorical feature
After Vectorization of school_state
(49041, 51)
(24155, 51)
(36052, 51)
feature names for school_state
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'h
i', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi',
'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'n
y', 'oh', 'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va',
'vt', 'wa', 'wi', 'wv', 'wy']
**************************************************


**************************************************
converting teacher_prefix to categorical feature
After Vectorization of teacher_prefix
(49041, 5)
(24155, 5)
(36052, 5)
feature names for teacher_prefix
['dr', 'mr', 'mrs', 'ms', 'teacher']
**************************************************


**************************************************
converting project_grade_category to categorical feature
After Vectorization of project_grade_category
(49041, 4)
(24155, 4)
(36052, 4)
feature names for project_grade_category
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
**************************************************


**************************************************
converting clean_categories to categorical feature
After Vectorization of clean_categories
(49041, 9)
(24155, 9)
(36052, 9)
feature names for clean_categories
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics',
'literacy_language', 'math_science', 'music_arts', 'specialneeds', 'wa
rmth']
**************************************************
```

```
**************************************************
converting clean_subcategories to categorical feature
After Vectorization of clean_subcategories
(49041, 30)
(24155, 30)
(36052, 30)
feature names for clean_subcategories
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_gover
nment', 'college_careerprep', 'communityservice', 'earlydevelopment',
'economics', 'environmentalscience', 'esl', 'extracurricular', 'financ
ialliteracy', 'foreignlanguages', 'gym_fitness', 'health_lifescience',
'health_wellness', 'history_geography', 'literacy', 'literature_writin
g', 'mathematics', 'music', 'nutritioneducation', 'other', 'parentinvo
lvement', 'performingarts', 'socialsciences', 'specialneeds', 'teamspo
rts', 'visualarts', 'warmth']
**************************************************
```

Encoding numerical features for:

- price
- teacher_number_of_previously_posted_projects

In [7]:

```
def return_numerical_features(numerical_feature,train_data, cv_data, test_data):
    normalizer = Normalizer()
    print(train_data[numerical_feature].values)
    normalizer.fit(train_data[numerical_feature].values.reshape(1,-1))
    train_data_features = normalizer.transform(train_data[numerical_feature].values.
    cv_data_features    = normalizer.transform(cv_data[numerical_feature].values.res
    test_data_features  = normalizer.transform(test_data[numerical_feature].values.r
    print('After Vectorization of {}'.format(numerical_feature))
    print(train_data_features.shape)
    print(cv_data_features.shape)
    print(test_data_features.shape)
    print('After Vectorization Reshape of {}'.format(numerical_feature))
    train_data_features = train_data_features.reshape(-1,1)
    cv_data_features    = cv_data_features.reshape(-1,1)
    test_data_features  = test_data_features.reshape(-1,1)
    print(train_data_features.shape)
    print(cv_data_features.shape)
    print(test_data_features.shape)
    print("*"*50)
    print("\n")
    return {
            'train_data_features' : train_data_features,
            'cv_data_features'    : cv_data_features,
            'test_data_features'  : test_data_features
           }
```

In [8]:

```
numerical_columns = ['price','teacher_number_of_previously_posted_projects']
converted_numerical_features = {}
for current_feature in numerical_columns:
    print("*"*50)
    print('converting {} to categorical feature'.format(current_feature))
    converted_numerical_features[current_feature] = return_numerical_features(currer


print(converted_numerical_features.keys())
```

```
**************************************************
converting price to categorical feature
[179.   339.94 103.98 ... 193.41 293.97 234.95]
After Vectorization of price
(1, 49041)
(1, 24155)
(1, 36052)
After Vectorization Reshape of price
(49041, 1)
(24155, 1)
(36052, 1)
**************************************************


**************************************************
converting teacher_number_of_previously_posted_projects to categorical
feature
[ 1  0 11 ...  7  1  0]
After Vectorization of teacher_number_of_previously_posted_projects
(1, 49041)
(1, 24155)
(1, 36052)
After Vectorization Reshape of teacher_number_of_previously_posted_pro
jects
(49041, 1)
(24155, 1)
(36052, 1)
**************************************************


dict_keys(['price', 'teacher_number_of_previously_posted_projects'])
```

In [ ]:

```

```

In [9]:

```
''' Calculating BOW for essay'''
def custum_vectorizer_calculate(vectorizer_type , custum_feature, train_data, cv_dat
    if vectorizer_type == 'BOW':
        vectorizer = CountVectorizer(min_df=10)
        print('Using BOW vectorizer')
    else:
        vectorizer = TfidfVectorizer(min_df=10)
        print('Using TFIDF vectorizer')
    vectorizer.fit_transform(train_data[custum_feature].values)
    train_data_features = vectorizer.transform(train_data[custum_feature].values)
    cv_data_features    = vectorizer.transform(cv_data[custum_feature].values)
    test_data_features  = vectorizer.transform(test_data[custum_feature].values)
    print('After Vectorization of {}'.format(custum_feature))
    print(train_data_features.shape)
    print(cv_data_features.shape)
    print(test_data_features.shape)
    print('feature names for {}'.format(custum_feature))
    print(vectorizer.get_feature_names())
    print("*"*50)
    print("\n")
    return {
            'train_data_features' : train_data_features,
            'cv_data_features'    : cv_data_features,
            'test_data_features'  : test_data_features,
            'feture_name'         : vectorizer.get_feature_names()
        }
```

In [10]:

```
''' Calculating BOW for essay'''
eassy_bow_features = custum_vectorizer_calculate('BOW','essay', X_train, X_cv, X_tes
```

```
Using BOW vectorizer
After Vectorization of essay
(49041, 12114)
(24155, 12114)
(36052, 12114)
feature names for essay
['00', '000', '10', '100', '1000', '100th', '101', '103', '10th', '1
1', '110', '1100', '115', '11th', '12', '120', '1200', '125', '12th',
'13', '130', '1300', '14', '140', '1400', '14th', '15', '150', '1500',
'16', '160', '1600', '165', '17', '170', '175', '17th', '18', '180',
'1800', '19', '1950', '1950s', '1960', '1980', '1999', '19th', '1st',
'20', '200', '2000', '2001', '2002', '2003', '2004', '2005', '2006',
'2007', '2008', '2009', '2010', '2011', '2012', '2013', '2014', '201
5', '2016', '2017', '2018', '2020', '20th', '21', '21st', '22', '225',
'23', '24', '240', '25', '250', '26', '260', '27', '28', '280', '29',
'2d', '2nd', '30', '300', '3000', '31', '32', '320', '33', '34', '35',
'350', '36', '360', '365', '37', '375', '38', '380', '39', '3d', '3doo
dler', '3doodlers', '3rd', '40', '400', '4000', '41', '42', '425', '4
3', '430', '44', '440', '45', '450', '46', '47', '48', '480', '49', '4
```

In [11]:

```
''' Calculating TFIDF for essay'''
eassy_tfidf_features = custum_vectorizer_calculate('TFIDF','essay', X_train, X_cv, X
```

```
Using TFIDF vectorizer
After Vectorization of essay
(49041, 12114)
(24155, 12114)
(36052, 12114)
feature names for essay
['00', '000', '10', '100', '1000', '100th', '101', '103', '10th', '1
1', '110', '1100', '115', '11th', '12', '120', '1200', '125', '12th',
'13', '130', '1300', '14', '140', '1400', '14th', '15', '150', '1500',
'16', '160', '1600', '165', '17', '170', '175', '17th', '18', '180',
'1800', '19', '1950', '1950s', '1960', '1980', '1999', '19th', '1st',
'20', '200', '2000', '2001', '2002', '2003', '2004', '2005', '2006',
'2007', '2008', '2009', '2010', '2011', '2012', '2013', '2014', '201
5', '2016', '2017', '2018', '2020', '20th', '21', '21st', '22', '225',
'23', '24', '240', '25', '250', '26', '260', '27', '28', '280', '29',
'2d', '2nd', '30', '300', '3000', '31', '32', '320', '33', '34', '35',
'350', '36', '360', '365', '37', '375', '38', '380', '39', '3d', '3doo
dler', '3doodlers', '3rd', '40', '400', '4000', '41', '42', '425', '4
3', '430', '44', '440', '45', '450', '46', '47', '48', '480', '49', '4
```

In [12]:

```
converted_categorical_features.keys()
type(converted_categorical_features['school_state']['train_data_features'][0][0])
```

Out[12]:

```
scipy.sparse.csr.csr_matrix
```

set 1 features

In [13]:

```
print('Train features shapes are:')
print(converted_categorical_features['school_state']['train_data_features'].shape, c
set_one_train_features = hstack((converted_categorical_features['school_state']['tra
set_one_cv_features    = hstack((converted_categorical_features['school_state']['cv_
set_one_test_features  = hstack((converted_categorical_features['school_state']['tes

print("Final Data matrix")
print(set_one_train_features.shape, y_train.shape)
print(set_one_cv_features.shape, y_cv.shape)
print(set_one_test_features.shape, y_test.shape)
print("="*100)

set_one_features = {
                        'train_data' : [set_one_train_features, y_train],
                        'cv_data'    : [set_one_cv_features, y_cv],
                        'test_data'  : [set_one_test_features, y_test]
                     }
```

```
Train features shapes are:
(49041, 51) (49041, 5) (49041, 4) (49041, 9) (49041, 30) (49041, 1211
4)
Final Data matrix
(49041, 12215) (49041,)
(24155, 12215) (24155,)
(36052, 12215) (36052,)
======================================================================
==============================
```

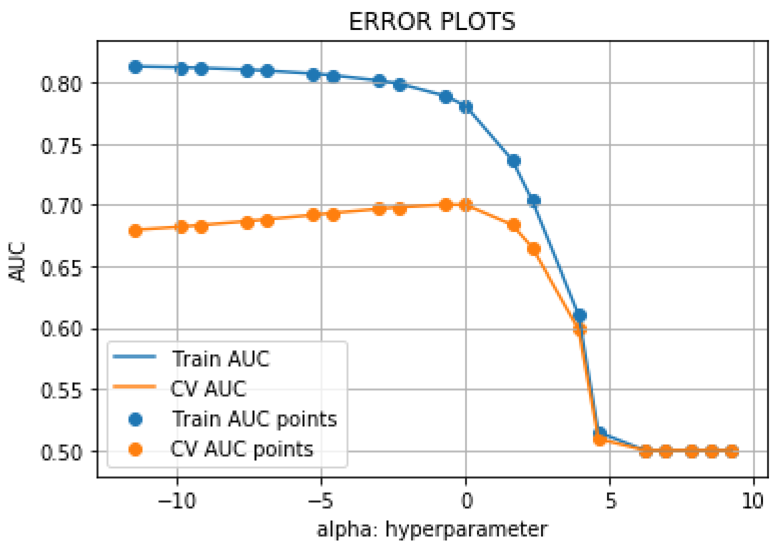
In [14]:

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
train_auc = []
cv_auc = []
alpha = [0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5,
#alpha = [1,5,10,15,20,25,30,37,45,50,60,70,80,100,110,120,125,140,150,160,175,190,2
for each_alpha in tqdm(alpha):
    clf = MultinomialNB(alpha=each_alpha,class_prior = [0.5, 0.5])
    clf.fit(set_one_train_features, y_train)
#     print(clf.predict_proba(set_one_train_features)[:,1])
    y_train_pred = clf.predict_proba(set_one_train_features)[:,1]
    y_cv_pred = clf.predict_proba(set_one_cv_features)[:,1]
    print(y_cv_pred, y_cv_pred)
    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()
# plt.show()

print('='*50)
print('alpha values with log on x-axis')
plt.plot(np.log(alpha), train_auc, label='Train AUC')
plt.plot(np.log(alpha), cv_auc, label='CV AUC')

plt.scatter(np.log(alpha), train_auc, label='Train AUC points')
plt.scatter(np.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()
```

```
  5%|▌         | 1/20 [00:00<00:02,  6.78it/s]

[1.         0.99987617 0.48531608 ... 0.35740365 0.02561981 1.
] [1.         0.99987617 0.48531608 ... 0.35740365 0.02561981 1.
]

 10%|█         | 2/20 [00:00<00:02,  7.16it/s]

[1.         0.99987616 0.48532224 ... 0.35740679 0.02561924 1.
] [1.         0.99987616 0.48532224 ... 0.35740679 0.02561924 1.
]

 15%|█▌        | 3/20 [00:00<00:02,  7.40it/s]

[0.99999999 0.99987616 0.48532994 ... 0.35741073 0.02561853 0.9999999
```

```
9] [0.99999999 0.99987616 0.48532994 ... 0.35741073 0.02561853 0.99999
999]

 20%|██        | 4/20 [00:00<00:02,  7.54it/s]

[0.99999979 0.99987615 0.4853915  ... 0.35744219 0.02561281 0.9999999
5] [0.99999979 0.99987615 0.4853915  ... 0.35744219 0.02561281 0.99999
995]

 25%|██▌       | 5/20 [00:00<00:01,  7.68it/s]

[0.99999917 0.99987613 0.48546847 ... 0.35748152 0.02560566 0.9999999
1] [0.99999917 0.99987613 0.48546847 ... 0.35748152 0.02560566 0.99999
991]

 30%|███       | 6/20 [00:00<00:01,  7.78it/s]

[0.99997917 0.99987601 0.48608424 ... 0.35779664 0.02554866 0.9999995
5] [0.99997917 0.99987601 0.48608424 ... 0.35779664 0.02554866 0.99999
955]

 35%|███▌      | 7/20 [00:00<00:01,  7.32it/s]

[0.99991627 0.99987586 0.48685415 ... 0.35819161 0.02547778 0.9999991
] [0.99991627 0.99987586 0.48685415 ... 0.35819161 0.02547778 0.999999
1 ]

 40%|████      | 8/20 [00:01<00:01,  7.08it/s]

[0.99782671 0.99987468 0.49302097 ... 0.36139427 0.02492547 0.9999955
4] [0.99782671 0.99987468 0.49302097 ... 0.36139427 0.02492547 0.99999
554]

 45%|████▌     | 9/20 [00:01<00:01,  7.20it/s]

[0.99095115 0.99987332 0.50074607 ... 0.3655042  0.02427011 0.9999912
] [0.99095115 0.99987332 0.50074607 ... 0.3655042  0.02427011 0.999991
2 ]

 50%|█████     | 10/20 [00:01<00:01,  7.24it/s]

[0.76495825 0.999867   0.56274116 ... 0.40252593 0.02014621 0.9999618
3] [0.76495825 0.999867   0.56274116 ... 0.40252593 0.02014621 0.99996
183]

 55%|█████▌    | 11/20 [00:01<00:01,  7.41it/s]

[0.3916344  0.99986889 0.6384497  ... 0.45837878 0.01687579 0.9999411
1] [0.3916344  0.99986889 0.6384497  ... 0.45837878 0.01687579 0.99994
111]

 60%|██████    | 12/20 [00:01<00:01,  7.42it/s]

[0.03098619 0.99997187 0.96135542 ... 0.92187101 0.01326813 0.9999915
9] [0.03098619 0.99997187 0.96135542 ... 0.92187101 0.01326813 0.99999
159]

 65%|██████▌   | 13/20 [00:01<00:00,  7.53it/s]

[0.0605437  0.99999884 0.99866244 ... 0.99879271 0.03477764 0.9999999
4] [0.0605437  0.99999884 0.99866244 ... 0.99879271 0.03477764 0.99999
994]

 70%|███████   | 14/20 [00:01<00:00,  7.78it/s]

[0.99999999 1.         1.         ... 1.         0.99998609 1.
```

```
] [0.99999999 1.         1.         ... 1.         0.99998609 1.
]

 75%|███████▌  | 15/20 [00:02<00:00,  7.32it/s]

[1. 1. 1. ... 1. 1. 1.] [1. 1. 1. ... 1. 1. 1.]

 80%|████████  | 16/20 [00:02<00:00,  7.68it/s]

[1. 1. 1. ... 1. 1. 1.] [1. 1. 1. ... 1. 1. 1.]

 85%|████████▌ | 17/20 [00:02<00:00,  7.95it/s]

[1. 1. 1. ... 1. 1. 1.] [1. 1. 1. ... 1. 1. 1.]

 90%|█████████ | 18/20 [00:02<00:00,  7.88it/s]

[1. 1. 1. ... 1. 1. 1.] [1. 1. 1. ... 1. 1. 1.]

 95%|█████████▌| 19/20 [00:02<00:00,  8.08it/s]

[1. 1. 1. ... 1. 1. 1.] [1. 1. 1. ... 1. 1. 1.]

100%|██████████| 20/20 [00:02<00:00,  7.67it/s]

[1. 1. 1. ... 1. 1. 1.] [1. 1. 1. ... 1. 1. 1.]
==================================================
alpha values with log on x-axis
```




```
'''

    - i tried with couple of aplha values from 0.00001 - 15,000
    - i found for more than 5000 epocs the AUC is falling below 51%
    - i found in between 0.5- 5 the epocs and AUC is closer, so i tried with mul
    tiple values.
    - based on the observations i took 1 best hyper-paramter for set-1

'''
```

In [15]:

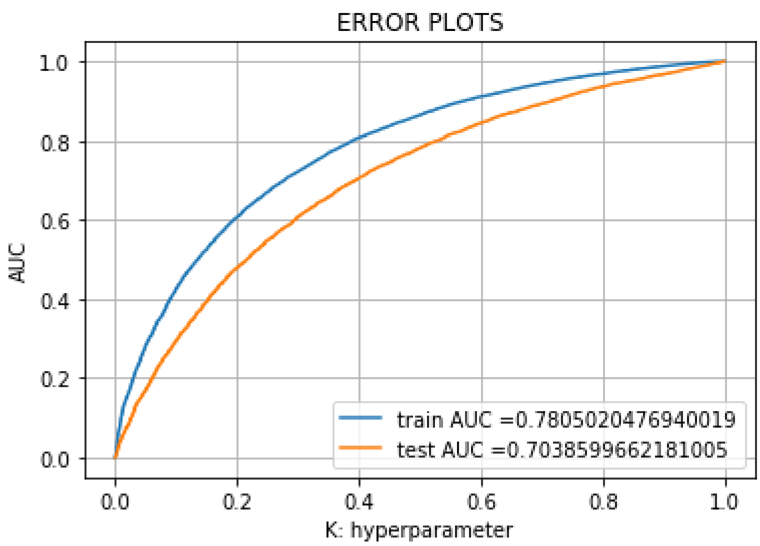
```
set_one_train_features.shape, y_train.shape
```

Out[15]:

```
((49041, 12215), (49041,))
```

In [48]:

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


SET_ONE_NB_MODEL = MultinomialNB(alpha=1,class_prior = [0.5, 0.5])
SET_ONE_NB_MODEL = SET_ONE_NB_MODEL.fit(set_one_train_features, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates c
# not the predicted outputs
set_one_y_train_pred = SET_ONE_NB_MODEL.predict_proba(set_one_train_features)[:,1]
set_one_y_test_pred  = SET_ONE_NB_MODEL.predict_proba(set_one_test_features)[:,1]
# y_train_pred = batch_predict(clf, set_one_train_features)
# y_test_pred = batch_predict(clf, set_one_test_features)

set_one_train_fpr, set_one_train_tpr, set_one_tr_thresholds = roc_curve(y_train, set
set_one_test_fpr, set_one_test_tpr, set_one_te_thresholds = roc_curve(y_test, set_or

plt.plot(set_one_train_fpr, set_one_train_tpr, label="train AUC ="+str(auc(set_one_t
plt.plot(set_one_test_fpr, set_one_test_tpr, label="test AUC ="+str(auc(set_one_test
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
print("Best alpha values is {} and test auc is {} ".format("1","0.703"))
```

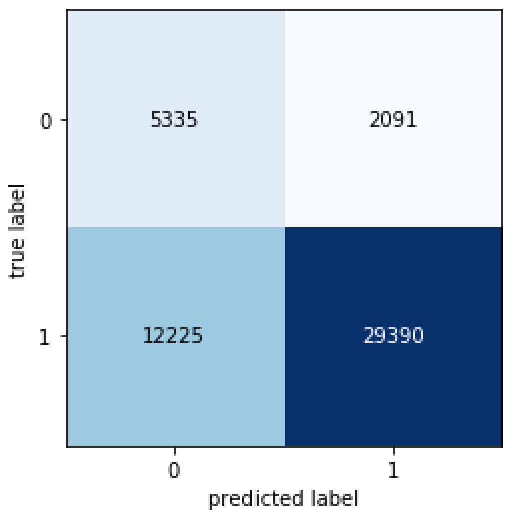



```
Best alpha values is 1 and test auc is 0.703
```

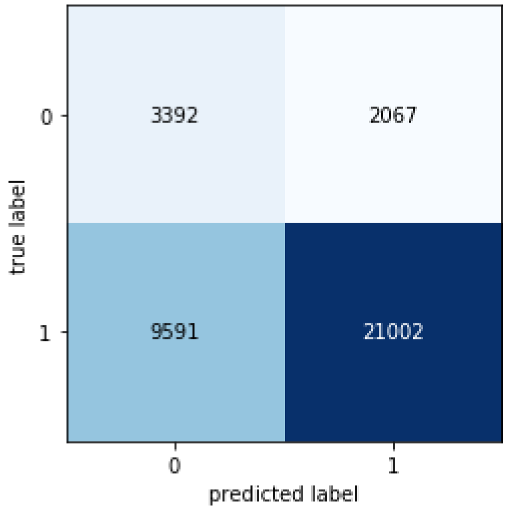
In [17]:

```
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.
    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

print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(set_one_tr_thresholds, set_one_train_fpr, set_one_trair
print("Train confusion matrix")
first_set_train_confusion_matrix = confusion_matrix(y_train, predict_with_best_t(set
plot_confusion_matrix(first_set_train_confusion_matrix)
plt.show()
print("Test confusion matrix")
first_set_test_confusion_matrix  = confusion_matrix(y_test, predict_with_best_t(set_
plot_confusion_matrix(first_set_test_confusion_matrix)
plt.show()
```

```
======================================================================
==============================
the maximum value of tpr*(1-fpr) 0.5073751187535027 for threshold 0.53
Train confusion matrix
```




```
Test confusion matrix
```



''' Getting feature name as shown in the below list '''

```
- school_state
- teacher_prefix
- project_grade_category
- clean_categories
- clean_subcategories
- price
- teacher_number_of_previously_posted_projects
```

In [18]:

```
set_one_total_feture_names = []
for each_category_feature in converted_categorical_features.keys():
    print(each_category_feature)
    for each_name in converted_categorical_features[each_category_feature]['feture_r
        set_one_total_feture_names.append(each_name)
```

```
school_state
teacher_prefix
project_grade_category
clean_categories
clean_subcategories
```

In [19]:

```
len(converted_categorical_features['school_state']['feture_name'])
```

Out[19]:

```
51
```

In [20]:

```
set_one_total_feture_names = list(set(set_one_total_feture_names))
len(set_one_total_feture_names)
```

Out[20]:

```
95
```

In [21]:

```
set_one_total_feture_names.append("price")
set_one_total_feture_names.append("teacher_number_of_previously_posted_projects")
for each_bow_name in list(set(eassy_bow_features['feture_name'])):
    set_one_total_feture_names.append(each_bow_name)


len(set_one_total_feture_names)
```

Out[21]:

```
12211
```

In [22]:

```
set_one_top_30_postive_propabilties = SET_ONE_NB_MODEL.feature_log_prob_[1, :].argsc
set_one_top_30_postive_classes = []
for i in set_one_top_30_postive_propabilties:
    set_one_top_30_postive_classes.append(set_one_total_feture_names[i])
```

In [23]:

```
set_one_top_30_negative_propabilties = SET_ONE_NB_MODEL.feature_log_prob_[0, :].args
set_one_top_30_negative_classes = []
for i in set_one_top_30_negative_propabilties:
    set_one_top_30_negative_classes.append(set_one_total_feture_names[i])
```

In [24]:

```
''' BOW vector of NB on set -1 '''
set_one_top_30_postive_classes
```

Out[24]:

```
['writes',
 'realizing',
 'scenarios',
 'precocious',
 'persevered',
 'parking',
 'contest',
 'invitation',
 'needing',
 'titled',
 'sc',
 'history',
 'calculus',
 'ngss',
 'grader',
 'responds',
 'detectives',
 'heterogeneous',
 'dropped',
 'tunnels']
```

In [25]:

```
''' BOW vector of NB on set -1 '''
set_one_top_30_negative_classes
```

Out[25]:

```
['writes',
 'realizing',
 'precocious',
 'scenarios',
 'persevered',
 'needing',
 'contest',
 'invitation',
 'parking',
 'titled',
 'history',
 'sc',
 'calculus',
 'grader',
 'ngss',
 'honorable',
 'heterogeneous',
 'responds',
 'dropped',
 'because']
```

In [26]:

```
''' Non common words in top postive and neagtive'''
set(set_one_top_30_postive_classes) ^ set(set_one_top_30_negative_classes)
```

Out[26]:

```
{'because', 'detectives', 'honorable', 'tunnels'}
```

In [27]:

```
'''common words in top postive and neagtive'''
set(set_one_top_30_postive_classes) & set(set_one_top_30_negative_classes)
```

Out[27]:

```
{'calculus',
 'contest',
 'dropped',
 'grader',
 'heterogeneous',
 'history',
 'invitation',
 'needing',
 'ngss',
 'parking',
 'persevered',
 'precocious',
 'realizing',
 'responds',
 'sc',
 'scenarios',
 'titled',
 'writes'}
```

Set 2 features

In [28]:

```
# print('Train features shapes are:')
# print(converted_categorical_features['school_state']['train_data_features'].shape,
set_two_train_features = hstack((converted_categorical_features['school_state']['tra
set_two_cv_features    = hstack((converted_categorical_features['school_state']['cv_
set_two_test_features  = hstack((converted_categorical_features['school_state']['tes

print("Final Data matrix")
print(set_two_train_features.shape, y_train.shape)
print(set_two_cv_features.shape, y_cv.shape)
print(set_two_test_features.shape, y_test.shape)
print("="*100)

set_two_features = {
                        'train_data'  : [set_two_train_features, y_train],
                        'cv_data'     : [set_two_cv_features, y_cv],
                        'test_data'   : [set_two_test_features, y_test]
                     }
```

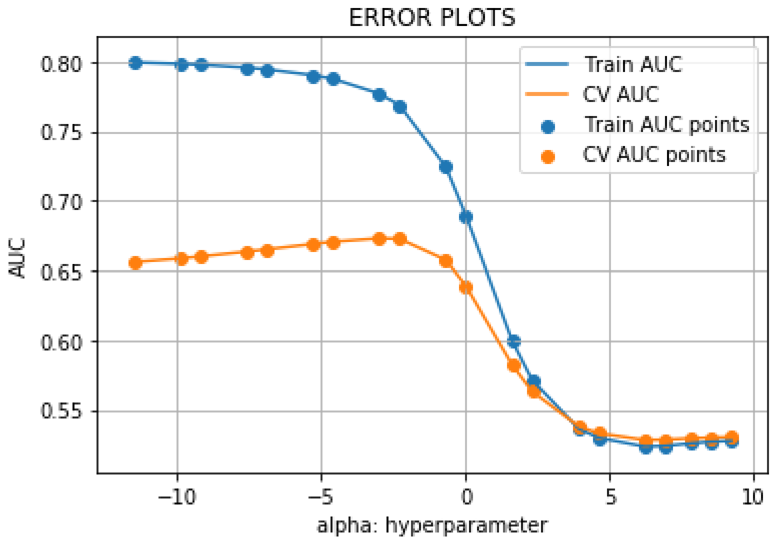
```
Final Data matrix
(49041, 12215) (49041,)
(24155, 12215) (24155,)
(36052, 12215) (36052,)
======================================================================
==============================
```

In [30]:

```
''' code referenced from AAIC '''
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
train_auc = []
cv_auc = []
alpha = [0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5,
# alpha = [0.00001,0.0001,0.001,0.01,0.1,1,10,25,50,62,75,100,110,150,1000,10000]
for each_alpha in tqdm(alpha):
    clf = MultinomialNB(alpha=each_alpha)
    clf.fit(set_two_train_features, y_train)
#     print(clf.predict_proba(set_two_train_features)[:,1])
    y_train_pred = clf.predict_proba(set_two_train_features)[:,1]
    y_cv_pred = clf.predict_proba(set_two_cv_features)[:,1]
    print(y_cv_pred, y_cv_pred)
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

print('alpha values with log on x-axis')

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

plt.scatter(np.log(alpha), train_auc, label='Train AUC points')
plt.scatter(np.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()
```

```
 10%|█         | 2/20 [00:00<00:02,  6.92it/s]

[0.98923788 0.9431281  0.8228707  ... 0.91104156 0.74062891 0.9763989
1] [0.98923788 0.9431281  0.8228707  ... 0.91104156 0.74062891 0.97639
891]
[0.98363841 0.94312719 0.82287492 ... 0.91104185 0.74062494 0.9712270
2] [0.98363841 0.94312719 0.82287492 ... 0.91104185 0.74062494 0.97122
702]

 20%|██        | 4/20 [00:00<00:02,  7.64it/s]

[0.98041921 0.94312604 0.82288021 ... 0.91104222 0.74061999 0.9686743
7] [0.98041921 0.94312604 0.82288021 ... 0.91104222 0.74061999 0.96867
437]
[0.97035824 0.94311693 0.82292248 ... 0.9110452  0.74058044 0.9618726
3] [0.97035824 0.94311693 0.82292248 ... 0.9110452  0.74058044 0.96187
263]

 30%|███       | 6/20 [00:00<00:01,  8.07it/s]

[0.96460441 0.94310559 0.82297531 ... 0.91104894 0.74053112 0.9585212
5] [0.96460441 0.94310559 0.82297531 ... 0.91104894 0.74053112 0.95852
125]
[0.94667638 0.94301672 0.82339771 ... 0.91107985 0.7401418  0.9495900
8] [0.94667638 0.94301672 0.82339771 ... 0.91107985 0.7401418  0.94959
008]

 40%|████      | 8/20 [00:01<00:01,  7.43it/s]
```

```
[0.93636873 0.94291017 0.82392509 ... 0.91112091 0.7396679  0.9451718
3] [0.93636873 0.94291017 0.82392509 ... 0.91112091 0.7396679  0.94517
183]
[0.90306664 0.94222577 0.82811831 ... 0.91154094 0.73634551 0.9333003
3] [0.90306664 0.94222577 0.82811831 ... 0.91154094 0.73634551 0.93330
033]

 50%|█████     | 10/20 [00:01<00:01,  7.61it/s]

[0.88282719 0.94173333 0.83328979 ... 0.91227129 0.73319975 0.9276485
] [0.88282719 0.94173333 0.83328979 ... 0.91227129 0.73319975 0.927648
5 ]
[0.82489237 0.94571191 0.87113323 ... 0.92300161 0.73134071 0.9228600
1] [0.82489237 0.94571191 0.87113323 ... 0.92300161 0.73134071 0.92286
001]

 60%|██████    | 12/20 [00:01<00:01,  7.97it/s]

[0.81574581 0.95702836 0.90842533 ... 0.9400813  0.75525057 0.9364931
8] [0.81574581 0.95702836 0.90842533 ... 0.9400813  0.75525057 0.93649
318]
[0.94849476 0.99620414 0.99319852 ... 0.99503498 0.94764616 0.9944686
3] [0.94849476 0.99620414 0.99319852 ... 0.99503498 0.94764616 0.99446
863]

 70%|███████   | 14/20 [00:01<00:00,  8.05it/s]

[0.99089975 0.99963434 0.99934761 ... 0.99957802 0.99146019 0.9995030
8] [0.99089975 0.99963434 0.99934761 ... 0.99957802 0.99146019 0.99950
308]
[0.9999337  0.99999927 0.99999857 ... 0.99999937 0.99994575 0.9999987
4] [0.9999337  0.99999927 0.99999857 ... 0.99999937 0.99994575 0.99999
874]

 80%|████████  | 16/20 [00:02<00:00,  7.82it/s]

[0.9999736  0.99999982 0.99999962 ... 0.99999982 0.99998102 0.9999994
7] [0.9999736  0.99999982 0.99999962 ... 0.99999982 0.99998102 0.99999
947]
[0.99987665 0.99999961 0.99999886 ... 0.99999888 0.99993772 0.9999926
] [0.99987665 0.99999961 0.99999886 ... 0.99999888 0.99993772 0.999992
6 ]

 90%|█████████ | 18/20 [00:02<00:00,  8.18it/s]

[0.99948034 0.99999836 0.99999419 ... 0.99999289 0.99975202 0.9999448
2] [0.99948034 0.99999836 0.99999419 ... 0.99999289 0.99975202 0.99994
482]
[0.99644179 0.9999805  0.99991978 ... 0.99989965 0.99824073 0.9992654
9] [0.99644179 0.9999805  0.99991978 ... 0.99989965 0.99824073 0.99926
549]

100%|██████████| 20/20 [00:02<00:00,  7.99it/s]

[0.98755973 0.99983363 0.99937442 ... 0.99929732 0.9932016  0.9960431
2] [0.98755973 0.99983363 0.99937442 ... 0.99929732 0.9932016  0.99604
312]
[0.96735404 0.99864058 0.99605213 ... 0.99600723 0.9793512  0.9848931
5] [0.96735404 0.99864058 0.99605213 ... 0.99600723 0.9793512  0.98489
315]
alpha values with log on x-axis
```



In [31]:

```
#  1. Based on different comparision of alpha value i absorved the train AUC falling
# 2. As to get best hyper paramater, i am seeing a training auc and cv auc compping
# 3. But by comparing all values in range fron 0.5 to 1 i saw train and test AUC sco
# 4. By all above absorvations i am taking alpha (best hyper paramter) as 0.1
```

In [32]:

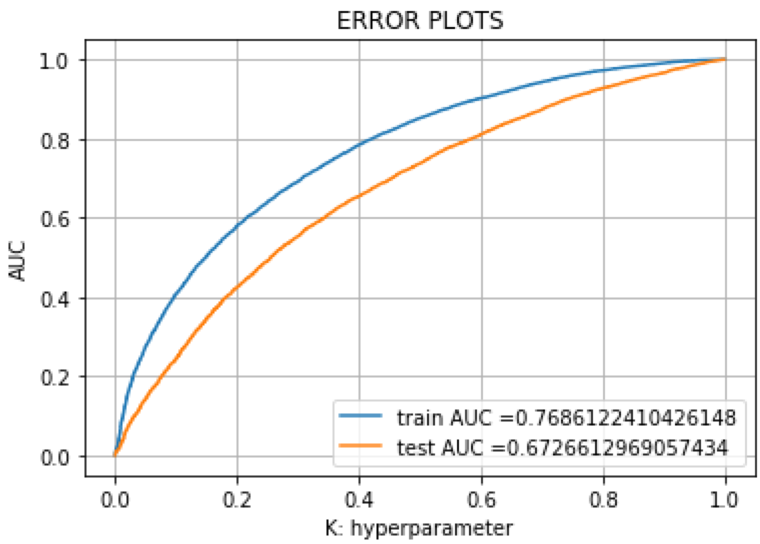
```
set_two_train_features.shape, y_train.shape
```

Out[32]:

```
((49041, 12215), (49041,))
```

In [49]:

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


NB_MODEL = MultinomialNB(alpha=0.1)
NB_MODEL_FIT = NB_MODEL.fit(set_two_train_features, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates c
# not the predicted outputs
y_train_pred = NB_MODEL.predict_proba(set_two_train_features)[:,1]
y_test_pred  = NB_MODEL.predict_proba(set_two_test_features)[:,1]
# y_train_pred = batch_predict(clf, set_two_train_features)
# y_test_pred = batch_predict(clf, set_two_test_features)

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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
print("Best alpha values is {} and test auc is {} ".format("0.1","0.672"))
```




```
Best alpha values is 0.1 and test auc is 0.672
```

In [34]:

```
get_feture_details = NB_MODEL_FIT.feature_log_prob_
# get_feture_details = sorted(get_feture_details[0])
len(get_feture_details)
```

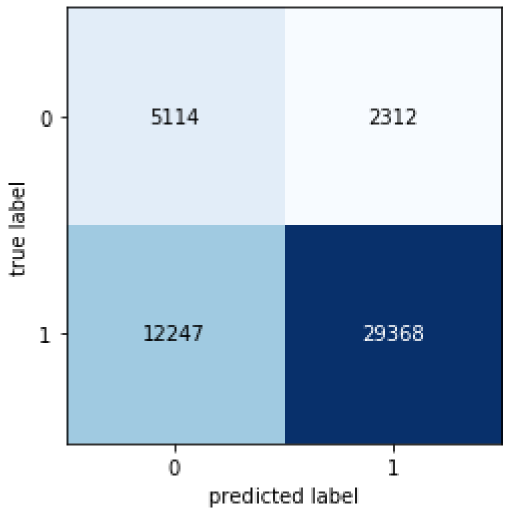
Out[34]:

```
2
```

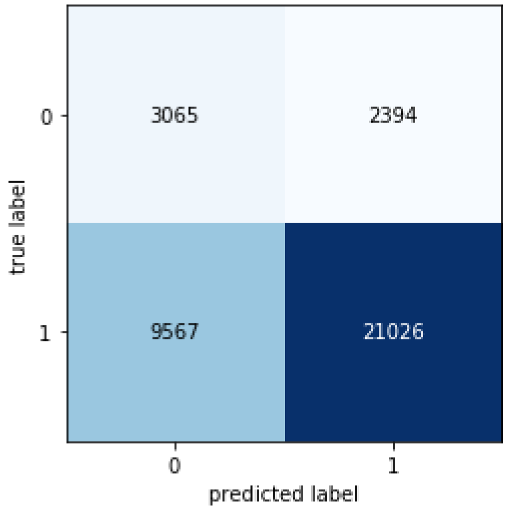
In [35]:

```
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.
    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

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

```
============================================================================
==============================
the maximum value of tpr*(1-fpr) 0.48599326563807965 for threshold 0.8
47
Train confusion matrix
```




```
Test confusion matrix
```



''' Getting feature name as shown in the below list '''

```
- school_state
- teacher_prefix
- project_grade_category
- clean_categories
- clean_subcategories
- price
- teacher_number_of_previously_posted_projects
```

In [36]:

```
total_feture_names = []
for each_category_feature in converted_categorical_features.keys():
    print(each_category_feature)
    for each_name in converted_categorical_features[each_category_feature]['feture_r
        total_feture_names.append(each_name)
```

```
school_state
teacher_prefix
project_grade_category
clean_categories
clean_subcategories
```

In [37]:

```
len(converted_categorical_features['school_state']['feture_name'])
```

Out[37]:

```
51
```

In [38]:

```
total_feture_names = list(set(total_feture_names))
len(total_feture_names)
```

Out[38]:

```
95
```

In [39]:

```
total_feture_names.append("price")
total_feture_names.append("teacher_number_of_previously_posted_projects")
for each_bow_name in list(set(eassy_tfidf_features['feture_name'])):
    total_feture_names.append(each_bow_name)


len(total_feture_names)
```

Out[39]:

```
12211
```

In [40]:

```
len(set(eassy_tfidf_features['feture_name']))
```

Out[40]:

```
12114
```

In [41]:

```
top_30_postive_propabilties = NB_MODEL.feature_log_prob_[1, :].argsort()[::-1][:20]
top_30_postive_classes = []
for i in top_30_postive_propabilties:
    top_30_postive_classes.append(total_feture_names[i])
```

In [42]:

```
top_30_negative_propabilties = NB_MODEL.feature_log_prob_[0, :].argsort()[::-1][:20]
top_30_negative_classes = []
for i in top_30_negative_propabilties:
    top_30_negative_classes.append(total_feture_names[i])
```

In [43]:

```
''' tfidf vector of NB on set -2 '''
top_30_postive_classes
```

Out[43]:

```
['mr',
 'teacher',
 'sd',
 'civics_government',
 'ny',
 'ga',
 'mathematics',
 'music',
 'wi',
 'ks',
 'nm',
 'parentinvolvement',
 'writes',
 'la',
 'price',
 'environmentalscience',
 'mt',
 'history_civics',
 'health_wellness',
 'nutritioneducation']
```

In [44]:

```
''' tfidf vector of NB on set -2 '''
top_30_negative_classes
```

Out[44]:

```
['mr',
 'teacher',
 'sd',
 'civics_government',
 'ny',
 'ga',
 'music',
 'mathematics',
 'wi',
 'ks',
 'price',
 'la',
 'parentinvolvement',
 'nm',
 'writes',
 'environmentalscience',
 'health_wellness',
 'history_civics',
 'mt',
 'nj']
```

In [45]:

```
''' Non common words in top postive and neagtive'''
set(top_30_postive_classes) ^ set(top_30_negative_classes)
```

Out[45]:

```
{'nj', 'nutritioneducation'}
```

In [46]:

```
''' common words in top postive and neagtive'''
set(top_30_postive_classes) & set(top_30_negative_classes)
```

Out[46]:

```
{'civics_government',
 'environmentalscience',
 'ga',
 'health_wellness',
 'history_civics',
 'ks',
 'la',
 'mathematics',
 'mr',
 'mt',
 'music',
 'nm',
 'ny',
 'parentinvolvement',
 'price',
 'sd',
 'teacher',
 'wi',
 'writes'}
```

In [50]:

```
# Reference from : https://stackoverflow.com/questions/36423259/how-to-use-pretty-ta
from prettytable import PrettyTable

NB_RESULTS_TABLE = PrettyTable()
NB_RESULTS_TABLE.field_names = ["VECTORIZER", "MODEL", "HYPER-PARAMETER", "AUC"]

NB_RESULTS_TABLE.add_row(["BOW", "NB", 1, 0.703])
NB_RESULTS_TABLE.add_row(["TFIDF", "NB", 0.1, 0.672])

print(NB_RESULTS_TABLE)
```

```
+------------+-------+-----------------+-------+
| VECTORIZER | MODEL | HYPER-PARAMETER |  AUC  |
+------------+-------+-----------------+-------+
|    BOW     |   NB  |        1        | 0.703 |
|   TFIDF    |   NB  |       0.1       | 0.672 |
+------------+-------+-----------------+-------+
```