Importing Libraries

In [2]:

```
%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.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc

import re

import pickle
from tqdm import tqdm
import os

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

2. Naive Bayes

1.1 Loading Data

In [3]:

```
import pandas
data = pandas.read_csv('preprocessed_data.csv', nrows=50000)
```

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

In [4]:

```
#Segregating output label and feature set

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_projects	clean_categories
0	ca	mrs	grades_prek_2	53	math_science

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=0.33, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

Declaring feature list

In [6]:

```
# These list will contain all feature names and will get updated after every encoding

feature_list_bow = []
feature_list_tfidf = []
```

1.3 Make Data Model Ready: encoding eassay, and project_title

Set 1: Bag of Words

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

feature_list_bow.extend(vectorizer.get_feature_names())#Update feature list

# 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)
```

```
(22445, 8) (22445,)
(11055, 8) (11055,)
(16500, 8) (16500,)
====================================================================================================

After vectorizations
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
====================================================================================================
```

Set 2: TFIDF

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)
```

```
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data

feature_list_tfidf.extend(vectorizer.get_feature_names())#Update feature list

# 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)
```

```
(22445, 8) (22445,)
(11055, 8) (11055,)
(16500, 8) (16500,)
====================================================================================================

After vectorizations
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
====================================================================================================
```

1.4 Make Data Model Ready: encoding numerical, categorical features

1.4.1 encoding categorical features: School State

In [8]:

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

#Update feature list
feature_list_bow.extend(vectorizer.get_feature_names())
feature_list_tfidf.extend(vectorizer.get_feature_names())

# 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)
```

```
After vectorizations
(22445, 51) (22445,)
(11055, 51) (11055,)
(16500, 51) (16500,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'k
s', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm',
'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv
', 'wy']
====================================================================================================
```

1.4.2 encoding categorical features: teacher_prefix

In [9]:

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

```
#Update feature list
feature_list_bow.extend(vectorizer.get_feature_names())
feature_list_tfidf.extend(vectorizer.get_feature_names())

# 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)
```

```
After vectorizations
(22445, 5) (22445,)
(11055, 5) (11055,)
(16500, 5) (16500,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
====================================================================================================
```

1.4.3 encoding categorical features: project_grade_category

In [10]:

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

#Update feature list
feature_list_bow.extend(vectorizer.get_feature_names())
feature_list_tfidf.extend(vectorizer.get_feature_names())

# 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)
```

```
After vectorizations
(22445, 4) (22445,)
(11055, 4) (11055,)
(16500, 4) (16500,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
====================================================================================================
```

1.4.4 encoding categorical features: clean_categories

In [11]:

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

#Update feature list
feature_list_bow.extend(vectorizer.get_feature_names())
feature_list_tfidf.extend(vectorizer.get_feature_names())

# 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)
```

```
After vectorizations
(22445, 9) (22445,)
(11055, 9) (11055,)
(16500, 9) (16500,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
====================================================================================================
```

1.4.5 encoding categorical features: clean_subcategories

In [12]:

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

#Update feature list
feature_list_bow.extend(vectorizer.get_feature_names())
feature_list_tfidf.extend(vectorizer.get_feature_names())

# 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)
```

```
After vectorizations
(22445, 30) (22445,)
(11055, 30) (11055,)
(16500, 30) (16500,)
['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']
====================================================================================================
```

1.4.6 encoding numerical features: price

In [13]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
normalizer.fit(X_train['price'].values.reshape(1,-1))

#Update feature list
feature_list_bow.append('price')
feature_list_tfidf.append('price')

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
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

1.4.7 encoding numerical features: teacher_number_of_previously_posted_projects

In [14]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))

#Update feature list
feature_list_bow.append('teacher_number_of_previously_posted_projects')
feature_list_tfidf.append('teacher_number_of_previously_posted_projects')


X_train_previously_posted_norm =
normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_cv_previously_posted_norm =
normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_test_previously_posted_norm =
normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_previously_posted_norm.shape, y_train.shape)
print(X_cv_previously_posted_norm.shape, y_cv.shape)
print(X_test_previously_posted_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Concatinating all the features

Set 1: BOW

In [15]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_1 = hstack((X_train_essay_bow, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe,
X_train_categories_ohe, X_train_subcategories_ohe, X_train_price_norm,
X_train_previously_posted_norm)).tocsr()
X_cr_1 = hstack((X_cv_essay_bow, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe,
X_cv_categories_ohe, X_cv_subcategories_ohe, X_cv_price_norm, X_cv_previously_posted_norm)).tocsr(
)
X_te_1 = hstack((X_test_essay_bow, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_c
ategories_ohe, X_test_subcategories_ohe, X_test_price_norm, X_test_previously_posted_norm)).tocsr(
)

print("Final Data matrix")
print(X_tr_1.shape, y_train.shape)
print(X_cr_1.shape, y_cv.shape)
print(X_te_1.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 5101) (22445,)
(11055, 5101) (11055,)
(16500, 5101) (16500,)
====================================================================================================
```

Set 2: TFIDF

In [16]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_2 = hstack((X_train_essay_tfidf, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_
train_categories_ohe, X_train_subcategories_ohe, X_train_price_norm,
X_train_previously_posted_norm)).tocsr()
X_cr_2 = hstack((X_cv_essay_tfidf, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_categorie
s_ohe, X_cv_subcategories_ohe, X_cv_price_norm, X_cv_previously_posted_norm)).tocsr()
X_te_2 = hstack((X_test_essay_tfidf, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test
_categories_ohe, X_test_subcategories_ohe, X_test_price_norm,
X_test_previously_posted_norm)).tocsr()

print("Final Data matrix")
print(X_tr_2.shape, y_train.shape)
print(X_cr_2.shape, y_cv.shape)
print(X_te_2.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 5101) (22445,)
(11055, 5101) (11055,)
(16500, 5101) (16500,)
====================================================================================================
```

Batch Wise prediction method

In [19]:

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # 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 = 49000
    # 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
```

Utility methods to find best probability threshold and predictions accordingly

In [20]:

```
# 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
```

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

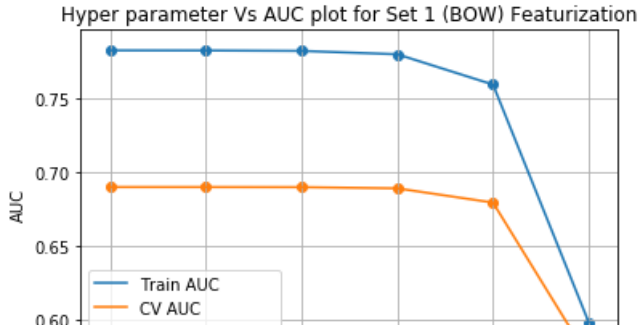
Apply NB on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instrucations

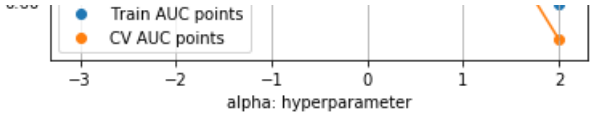
1.5.1 Applying Multinomial NB on BOW featurization (Set 1)

In [24]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import RandomizedSearchCV

naive = MultinomialNB()
parameters = {'alpha': [0.001, 0.01, 0.1, 1, 10, 100]}
clf = RandomizedSearchCV(naive, parameters, cv=3, scoring='roc_auc', return_train_score=True)
clf.fit(X_tr_1, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_alpha'])

# print(results)

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['param_alpha']

K_log = np.log10([0.001, 0.01, 0.1, 1, 10, 100]) #Keeping the hyperparameter axis as log(base 10)
scale.
plt.plot(K_log, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.2,color='darkblue')

plt.plot(K_log, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

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


plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot for Set 1 (BOW) Featurization")
plt.grid()
plt.show()
```

```
C:\Users\prakhar.raj.HIGHRADIUS\AppData\Roaming\Python\Python36\site-
packages\sklearn\model_selection\_search.py:266: UserWarning:

The total space of parameters 6 is smaller than n_iter=10. Running 6 iterations. For exhaustive
searches, use GridSearchCV.
```





In [25]:

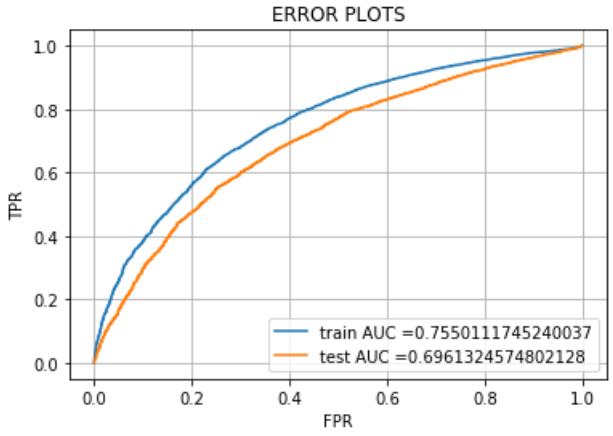
```
best_alpha_set_1 = clf.best_params_['alpha']
print('Best alpha hyperparameter for Set 1(BOW):',clf.best_params_['alpha'])
```

```
Best alpha hyperparameter for Set 1(BOW): 0.001
```

ROC Curve on Train and Test for Set 1 (BOW)

In [21]:

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

naive = MultinomialNB(alpha=best_alpha_set_1)
naive.fit(X_tr_1, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(naive, X_tr_1)
y_test_pred = batch_predict(naive, X_te_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)

auc_set1 = auc(test_fpr, test_tpr)

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("ERROR PLOTS")
plt.grid()
plt.show()
```



Confusion Matrix for Set 1 (BOW)

In [22]:

```
from sklearn.metrics import confusion_matrix

best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)

print("Test confusion matrix for Set 1 (BOW)")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
Test confusion matrix for Set 1 (BOW)
[[1707  935]
 [4792 9066]]
```

Top 20 features from Set 1 (BOW)

In [25]:

```
index_list = list(naive.feature_log_prob_[1, :].argsort()[::-1][:X_tr_1.shape[1]])[:20] #This list contains index of top 20 features
```

In [26]:

```
values = list(map(list(naive.feature_log_prob_[1, :]).__getitem__, index_list))
features = list(map(feature_list_bow.__getitem__, index_list))

from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Feature", "Log Probability Value"]

for i in range(0,20):
    x.add_row([features[i], values[i]])

print("Top 20 features in the Set 1 (BOW) approach for positive class are: \n", x)
```

```
Top 20 features in the Set 1 (BOW) approach for positive class are:
 +-------------+------------------------+
|   Feature   | Log Probability Value |
+-------------+------------------------+
|   students  |   -3.220302734326877  |
|    school   |   -4.364122846210263  |
|      my     |   -4.685208027077222  |
|  classroom  |   -4.715195327744031  |
|   learning  |   -4.720993745239435  |
|     the     |  -4.9634797799478605  |
|     not     |   -5.020992307973163  |
|     they    |   -5.025687152073283  |
|    learn    |   -5.054777217586389  |
| my students |   -5.070784793580673  |
|     help    |  -5.0827071646295465  |
|    price    |   -5.222492165208404  |
|     many    |   -5.259398315651353  |
|    nannan   |  -5.3115988101402785  |
|     work    |   -5.368453648151515  |
|      we     |  -5.3713430238505815  |
|   reading   |   -5.385978943115594  |
|     need    |   -5.398235923475276  |
|     use     |   -5.463364883628138  |
|     day     |   -5.509714555556178  |
+-------------+------------------------+
```

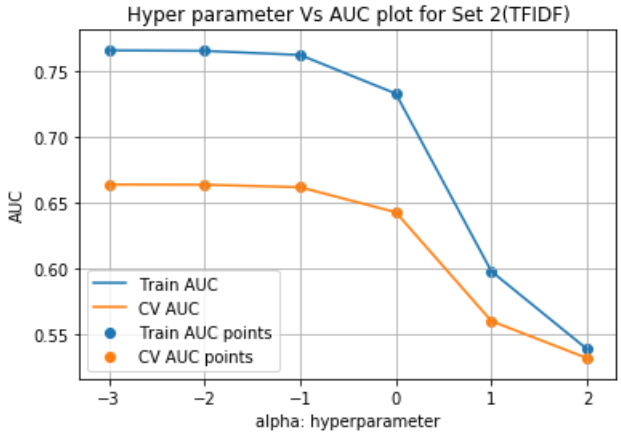
1.5.2 Applying Multinomial NB on TFIDF featurization (Set 2)

In [27]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import RandomizedSearchCV

naive = MultinomialNB()
parameters = {'alpha': [0.001, 0.01, 0.1, 1, 10, 100]}
clf = RandomizedSearchCV(naive, parameters, cv=3, scoring='roc_auc', return_train_score=True)
clf.fit(X_tr_2, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_alpha'])

# print(results)

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']
```

```
cv_auc_std= results['std_test_score']
K =  results['param_alpha']

K_log = np.log10([0.001, 0.01, 0.1, 1, 10, 100]) #Keeping the hyperparameter axis as log(base 10)
scale.
plt.plot(K_log, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.2,color='darkblue')

plt.plot(K_log, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

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


plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot for Set 2(TFIDF)")
plt.grid()
plt.show()
```

```
C:\Users\prakhar.raj.HIGHRADIUS\AppData\Roaming\Python\Python36\site-
packages\sklearn\model_selection\_search.py:266: UserWarning:

The total space of parameters 6 is smaller than n_iter=10. Running 6 iterations. For exhaustive
searches, use GridSearchCV.
```



In [28]:

```
best_alpha_set_2 = clf.best_params_['alpha']
print('Best alpha hyperparameter for Set 2(TFIDF):',clf.best_params_['alpha'])
```

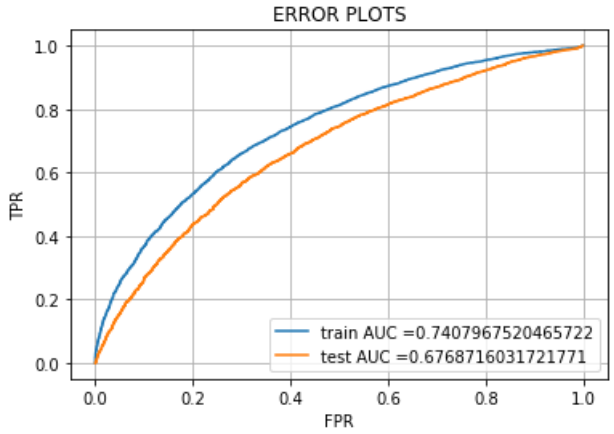
```
Best alpha hyperparameter for Set 2(TFIDF): 0.001
```

ROC Curve on Train and Test for Set 2 (TFIDF)

In [29]:

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

naive = MultinomialNB(alpha=best_alpha_set_2)
naive.fit(X_tr_2, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(naive, X_tr_2)
y_test_pred = batch_predict(naive, X_te_2)
```

```
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)

auc_set2 = auc(test_fpr, test_tpr)

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("ERROR PLOTS")
plt.grid()
plt.show()
```



Confusion Matrix for Set 2 (TFIDF)

In [30]:

```
from sklearn.metrics import confusion_matrix

best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)

print("Test confusion matrix for Set 2 (TFIDF)")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
Test confusion matrix for Set 2 (TFIDF)
[[1603 1039]
 [4775 9083]]
```

Top 20 features from Set 2 (TFIDF)

In [31]:

```
index_list = list(naive.feature_log_prob_[1, :].argsort()[::-1][:X_tr_2.shape[1]])[:20] #This list contains index of top 20 features
```

In [32]:

```
values = list(map(list(naive.feature_log_prob_[1, :]).__getitem__, index_list))
features = list(map(feature_list_tfidf.__getitem__, index_list))

x = PrettyTable()
x.field_names = ["Feature", "Log Probability Value"]

for i in range(0,20):
    x.add_row([features[i], values[i]])

print("Top 20 features in the Set 2 (TFIDF) approach for positive class are: \n", x)
```

```
Top 20 features in the Set 2 (TFIDF) approach for positive class are:
 +-----------------------------------------------+-----------------------+
|                    Feature                    | Log Probability Value |
+-----------------------------------------------+-----------------------+
```

```
|                    price                     |  -2.8868823980470406  |
| teacher_number_of_previously_posted_projects |  -3.2612961422309166  |
|                     mrs                      |  -3.5020865598542557  |
|               literacy_language              |  -3.6138030291011916  |
|                 grades_prek_2                |  -3.7778983500486483  |
|                 math_science                 |   -3.91340836631643   |
|                      ms                      |   -3.940424863572497  |
|                  grades_3_5                  |  -3.9532867752454877  |
|                   literacy                   |   -4.054222223962295  |
|                 mathematics                  |   -4.297773721164628  |
|               literature_writing             |   -4.451453256481999  |
|                 health_sports                |   -4.568349005418847  |
|                  grades_6_8                  |   -4.782058791160436  |
|                       ca                     |   -4.794847593245778  |
|               health_wellness                |  -4.8070779571247995  |
|                   students                   |  -4.9772845987822905  |
|                 specialneeds                 |   -5.075759525662705  |
|                 specialneeds                 |   -5.075759525662705  |
|               appliedlearning                |   -5.166393112664681  |
|                 grades_9_12                  |   -5.238389785948306  |
+----------------------------------------------+-----------------------+
```

3. Summary

In [33]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyperparameter", "AUC"]


x.add_row(['Bag of Words', 'Brute', best_alpha_set_1, auc_set1])
x.add_row(['TFIDF', 'Brute', best_alpha_set_2, auc_set2])
print(x)
```

```
+--------------+-------+----------------+--------------------+
|  Vectorizer  | Model | Hyperparameter |        AUC         |
+--------------+-------+----------------+--------------------+
| Bag of Words | Brute |     0.001      | 0.6961324574802128 |
|    TFIDF     | Brute |     0.001      | 0.6768716031721771 |
+--------------+-------+----------------+--------------------+
```