Problem Statement

- The goal of this Kaggle competition(our project) is to predict whether or not a DonorsChoose.org project proposal submitted by a teacher will be approved, using the text of project descriptions as well as additional metadata about the project, teacher, and school.

Naive Bayes

In [331]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import math
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 collections import Counter
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
import numpy as np
```

Loading Data

In [293]:

```
import pandas
data = pandas.read_csv('preprocessed_data.csv',nrows=100000)   #taking 100k rows
```

Splitting data into Train and test

In [294]:

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

Out[294]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcategories	
0	ca	mrs	grades_prek_2	53	math_science	appliedsciences health_lifescience	fo u: st

In [295]:

```
# train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
```

Make Data Model Ready: categorical, numerical and text features

1. Encoding text feature using BOW vectorizer

In [296]:

```
feature_names_with_bow=[]  #to store the name of all features for set1
```

In [297]:

```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
print("="*100)

vectorizer = CountVectorizer(min_df=10,ngram_range=(1,2), max_features=10000)   #BOW vectorizer
X_train_essay_bow=vectorizer.fit_transform(X_train['essay'].values) # fit has to happen only on train data
# Append the items to NumPy array using numpy.append() method reference taken from: https://itsmycode.com/numpy-ndarray-object-ha
feature_names_with_bow = np.append(feature_names_with_bow,vectorizer.get_feature_names_out()) #append the all feature names to ke

X_test_essay_bow = vectorizer.transform(X_test['essay'].values) # we use the fitted CountVectorizer to convert the test text to v

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

```
(67000, 8) (67000,)
(33000, 8) (33000,)
====================================================================================================
After vectorizations
(67000, 10000) (67000,)
(33000, 10000) (33000,)
====================================================================================================
```

2. Encoding text feature using TFIDF Vectorizer

In [298]:

```
feature_names_with_tfidf=[]  #to store the name of all features for set2
```

In [299]:

```

vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,2), max_features=10000)                 #TFIDF vectorizer

X_train_essay_tfidf = vectorizer.fit_transform(X_train['essay'].values) #fitted only the train data
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values) # we use the fitted TfidfVectorizer to convert the test text to


feature_names_with_tfidf = np.append(feature_names_with_tfidf,vectorizer.get_feature_names_out())#append the all feature names to
```

In [300]:

```
print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)  #size of train and test vector
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(67000, 10000) (67000,)
(33000, 10000) (33000,)
====================================================================================================
```

3.encoding categorical features: School State

In [301]:

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer.transform(X_train['school_state'].values)

feature_names_with_bow = np.append(feature_names_with_bow,vectorizer.get_feature_names_out())#to keep track of feature names
feature_names_with_tfidf=np.append(feature_names_with_tfidf,vectorizer.get_feature_names_out())#to keep track of feature names

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_test_state_ohe.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(67000, 51) (67000,)
(33000, 51) (33000,)
====================================================================================================
```

4. encoding categorical features: teacher_prefix

In [302]:

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values)

feature_names_with_bow = np.append(feature_names_with_bow,vectorizer.get_feature_names_out()) #to keep track of feature names
feature_names_with_tfidf=np.append(feature_names_with_tfidf,vectorizer.get_feature_names_out())

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_test_teacher_ohe.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(67000, 5) (67000,)
(33000, 5) (33000,)
====================================================================================================
```

5. encoding categorical features: project_grade_category

In [303]:

```
vectorizer = CountVectorizer(binary=True)
X_train_grade_ohe=vectorizer.fit_transform(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_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].values)

feature_names_with_bow = np.append(feature_names_with_bow,vectorizer.get_feature_names_out()) #to keep track of feature names
feature_names_with_tfidf=np.append(feature_names_with_tfidf,vectorizer.get_feature_names_out())

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_test_grade_ohe.shape, y_test.shape)
#print(vectorizer.get_feature_names())             #feature names of 4 dim vector
print("="*100)
```

```
After vectorizations
(67000, 4) (67000,)
(33000, 4) (33000,)
====================================================================================================
```

6. encoding categorical features: clean_categories

In [304]:

```
vectorizer = CountVectorizer(binary=True)
X_train_category_ohe=vectorizer.fit_transform(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_test_category_ohe = vectorizer.transform(X_test['clean_categories'].values)

feature_names_with_bow = np.append(feature_names_with_bow,vectorizer.get_feature_names_out()) #to keep track of feature names
feature_names_with_tfidf=np.append(feature_names_with_tfidf,vectorizer.get_feature_names_out())


print("After vectorizations")
print(X_train_category_ohe.shape, y_train.shape)
print(X_test_category_ohe.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(67000, 9) (67000,)
(33000, 9) (33000,)
====================================================================================================
```

7. encoding categorical features: clean_subcategories

In [305]:

```
vectorizer = CountVectorizer(binary=True)
X_train_subcategory_ohe=vectorizer.fit_transform(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_test_subcategory_ohe = vectorizer.transform(X_test['clean_categories'].values)

feature_names_with_bow = np.append(feature_names_with_bow,vectorizer.get_feature_names_out())#to keep track of feature names
feature_names_with_tfidf=np.append(feature_names_with_tfidf,vectorizer.get_feature_names_out())

print("After vectorizations")
print(X_train_subcategory_ohe.shape, y_train.shape)
print(X_test_subcategory_ohe.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(67000, 30) (67000,)
(33000, 30) (33000,)
====================================================================================================
```

8. encoding numerical features: Price

In [306]:

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()      #doing minmaxscaling to numerical feature(price)
#reshape(-1, 1)--so that minmaxscaling applied on price column(feature column)
X_train_price_norm=scaler.fit_transform(X_train['price'].values.reshape(-1, 1)) #fitting train data
X_test_price_norm = scaler.transform(X_test['price'].values.reshape(-1, 1))     #converting test data using fitted minmaxscaler

feature_names_with_bow = np.append(feature_names_with_bow, ['price']) #to keep track of feature names
feature_names_with_tfidf=np.append(feature_names_with_tfidf,['price'])


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

```
After vectorizations
(67000, 1) (67000,)
(33000, 1) (33000,)
====================================================================================================
```

9. encoding numerical features: teacher_number_of_previously_posted_projects

In [307]:

```
scaler = MinMaxScaler()
#reshape(-1, 1)--so that minmaxscaling applied on teacher_number_of_previously_posted_projects column(feature column)
X_train_previous_projects_norm = scaler.fit_transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,
X_test_previous_projects_norm = scaler.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))


feature_names_with_bow = np.append(feature_names_with_bow, ['teacher_number_of_previously_posted_projects'])#to keep track of fea
feature_names_with_tfidf=np.append(feature_names_with_tfidf,['teacher_number_of_previously_posted_projects'])

print("After vectorizations")
print(X_train_previous_projects_norm.shape, y_train.shape)
print(X_test_previous_projects_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(67000, 1) (67000,)
(33000, 1) (33000,)
====================================================================================================
```

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

Set 1- categorical + numerical features + preprocessed_eassay (BOW)

In [308]:

```
from scipy.sparse import hstack
#all the necessary feature for set1 is stacked together horizontally
#stacked train features
X_tr_bow = hstack((X_train_essay_bow,X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_category_ohe, X_train_sub
#stacked test features
X_te_bow = hstack((X_test_essay_bow,X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_category_ohe, X_test_subcatego

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

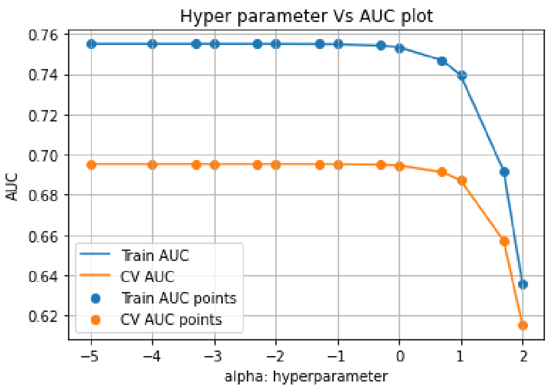
```
Final Data matrix
(67000, 10101) (67000,)
(33000, 10101) (33000,)
====================================================================================================
```

Fitting Naive bayes model on set1 features

In [309]:

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

NB = MultinomialNB(class_prior=[0.5,0.5])     #multinomial naive bayes

parameters = {'alpha':[0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,100]} #hyper parameter list for gridsearch
clf = GridSearchCV(NB, parameters, cv=3, scoring='roc_auc',return_train_score=True) #applying gridsearch to find best hyperparame
clf.fit(X_tr_bow, y_train)    #fitting the NB model with train data

results = pd.DataFrame.from_dict(clf.cv_results_)   #storing Gridsearch results

results = results.sort_values(['param_alpha'])

train_auc= results['mean_train_score']       #storing required Gridsearch results in required variable
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
alpha =  results['param_alpha'].tolist()
print(alpha)
print(np.log10(alpha))

plt.plot(np.log10(alpha), train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039

plt.plot(np.log10(alpha), cv_auc, label='CV AUC')

plt.scatter(np.log10(alpha), train_auc, label='Train AUC points')
plt.scatter(np.log10(alpha), cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```

```
[1e-05, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100]
[-5.      -4.      -3.30103 -3.      -2.30103 -2.      -1.30103 -1.
 -0.30103  0.       0.69897  1.       1.69897  2.     ]
```



Out[309]:

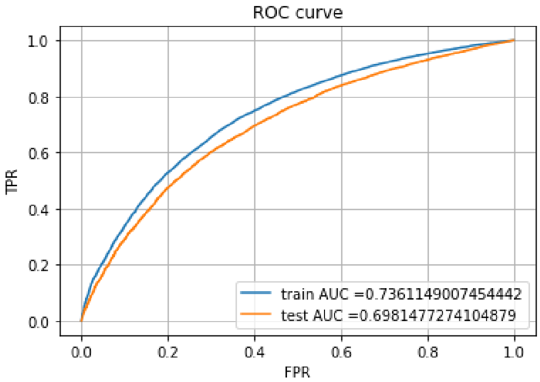
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	split2_test_score	m
0	0.080660	0.012274	0.024005	0.002153	0.00001	{'alpha': 1e-05}	0.695652	0.692786	0.697340	
2	0.086033	0.019612	0.025329	0.002627	0.0001	{'alpha': 0.0001}	0.695652	0.692790	0.697340	
1	0.072684	0.009442	0.023324	0.000471	0.0005	{'alpha': 0.0005}	0.695651	0.692794	0.697339	
4	0.069664	0.004644	0.026335	0.002503	0.001	{'alpha': 0.001}	0.695651	0.692795	0.697339	
3	0.067047	0.000816	0.023656	0.000931	0.005	{'alpha': 0.005}	0.695649	0.692796	0.697336	

In [310]:

```
best_alpha=1     #from the above"Hyper parameter Vs AUC plot" we can see that for the alpha value of 1=log10(0) we have higher te
```

In [311]:

```
# 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_alpha,class_prior=[0.5,0.5])
NB.fit(X_tr_bow, y_train)             #fitting the NB model with best alpha
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class,not the predicted output

y_train_pred_proba = NB.predict_proba(X_tr_bow)
y_test_pred_proba = NB.predict_proba(X_te_bow)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred_proba[:,[1]])        #find FPR and TPR for plotting roc cur
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred_proba[:,[1]])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))   #finding area under the ROC curve using FPR
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC curve")
plt.grid()
plt.show()
```



In [312]:

```
# 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): #for plotting confusion matrix we need to give 1 or 0 for prediction instead of proba
                                            #so this function convert prob value to 0 or 1 basen on best threshold
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)       # if the prob value is equal to or higher than threshold , append 1(consider that as 1),
        else:
            predictions.append(0)
    return predictions
```

In [313]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr) #finding the best threshold vaule for which AUC value is more
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred_proba[:,[1]], best_t)))   #plot confusion matrix
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred_proba[:,[1]], best_t)))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.46140227614281654 for threshold 0.405
Train confusion matrix
[[ 6925  3248]
 [18309 38518]]
Test confusion matrix
[[ 3130  1880]
 [ 9263 18727]]
```

In [314]:

```
feature_log_probability_values = NB.feature_log_prob_ #getting feature probability(given any feature what is the probability of p

positive_feature_log_probability_values = feature_log_probability_values[1] #probability values for each feature which helps to p
negative_feature_log_probability_values = feature_log_probability_values[0]#probability values for each feature which helps to pr

#sorting in decending order reference taken from : https://stackoverflow.com/a/16486299/17345549
positive_sorted_index=positive_feature_log_probability_values.argsort()[::-1][:20] #feature with top 20 probability score helping
negative_sorted_index=negative_feature_log_probability_values.argsort()[::-1][:20] #feature with top 20 probability score helping
```

In [315]:

```
print(feature_log_probability_values.shape)  #to make sure the dim of feature prob value vector and feature name vector are same
print(feature_names_with_bow.shape)
```

```
(2, 10101)
(10101,)
```

In [316]:

```
#printing those top 20 featuers, which helped to predict the positive classes, with the help of sorted prob index and  already st
#printing those top 20 featuers with the help of soreted prob index and  already store List of feature names
top_20_important_features_for_determining_postivite_class=[]

for i in positive_sorted_index:
    top_20_important_features_for_determining_postivite_class.append(feature_names_with_bow[i])

print(top_20_important_features_for_determining_postivite_class)
```

```
['students', 'school', 'my', 'learning', 'classroom', 'the', 'not', 'they', 'my students', 'learn', 'help', 'many',
'nannan', 'we', 'reading', 'work', 'need', 'use', 'love', 'day']
```

In [317]:

```
#printing those top 20 featuers, which helps to predict the nagative class, with the help of sorted prob index and  already store
top_20_important_features_for_determining_negative_class=[]

for i in negative_sorted_index:
    top_20_important_features_for_determining_negative_class.append(feature_names_with_bow[i])

print(top_20_important_features_for_determining_negative_class)
```

```
['students', 'school', 'learning', 'my', 'classroom', 'not', 'learn', 'they', 'help', 'the', 'my students', 'nanna
n', 'many', 'we', 'need', 'work', 'come', 'reading', 'love', 'able']
```

Set 2- categorical + numerical features + preprocessed_eassay (TFIDF)

In [318]:

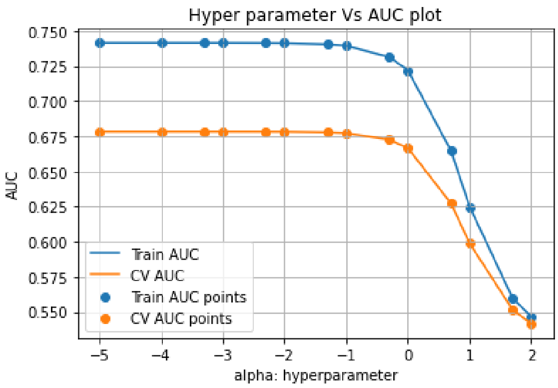
```
from scipy.sparse import hstack
#all the necessary feature for set2 is stacked together horizontally
#stacked train features
X_tr_tfidf = hstack((X_train_essay_tfidf,X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_category_ohe, X_train
#stacked test features
X_te_tfidf = hstack((X_test_essay_tfidf,X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_category_ohe, X_test_subca

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

```
Final Data matrix
(67000, 10101) (67000,)
(33000, 10101) (33000,)
====================================================================================================
```

In [319]:

```
# 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
from sklearn.naive_bayes import MultinomialNB
NB = MultinomialNB(class_prior=[0.5,0.5])                    #multinomial naive bayes

parameters = {'alpha':[0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,100]}     #hyper parameter list for gridsea
clf = GridSearchCV(NB, parameters, cv=3, scoring='roc_auc',return_train_score=True)          #applying gridsearch to find best
clf.fit(X_tr_tfidf, y_train)                              #fitting the NB model with train data

results = pd.DataFrame.from_dict(clf.cv_results_)         #storing Gridsearch results

results = results.sort_values(['param_alpha'])
train_auc= results['mean_train_score']#storing required Gridsearch results in required variable
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
alpha =  results['param_alpha'].tolist()
print(alpha)
print(np.log10(alpha))

plt.plot(np.log10(alpha), train_auc, label='Train AUC')     #in x axis "log of alpha" is used for better visualisation
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.plot(np.log10(alpha), cv_auc, label='CV AUC')
plt.scatter(np.log10(alpha), train_auc, label='Train AUC points')
plt.scatter(np.log10(alpha), cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```

```
[1e-05, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100]
[-5.      -4.      -3.30103 -3.      -2.30103 -2.      -1.30103 -1.
 -0.30103  0.       0.69897  1.       1.69897  2.     ]
```



Out[319]:

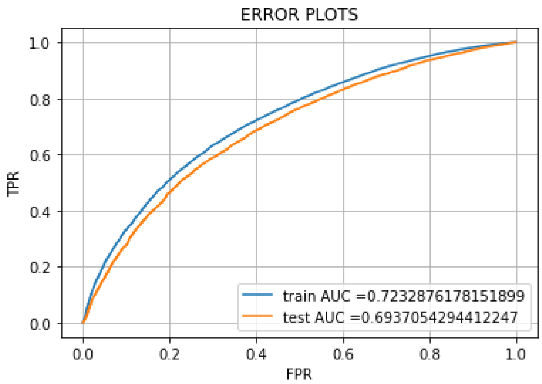
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	split2_test_score	m
0	0.072348	0.004039	0.024695	0.000926	0.00001	{'alpha': 1e-05}	0.682958	0.673593	0.678570	
2	0.075011	0.010661	0.032090	0.007037	0.0001	{'alpha': 0.0001}	0.682957	0.673592	0.678568	
1	0.074991	0.002937	0.024322	0.001892	0.0005	{'alpha': 0.0005}	0.682954	0.673588	0.678563	
4	0.076007	0.008641	0.022659	0.000471	0.001	{'alpha': 0.001}	0.682950	0.673583	0.678557	
3	0.078338	0.006778	0.027326	0.003690	0.005	{'alpha': 0.005}	0.682913	0.673540	0.678508	

In [320]:

```
best_alpha=0.01  #from the above"Hyper parameter Vs AUC plot" we can see that for the alpha value of 0.01=log10(-2) we have highe
```

In [321]:

```
# 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_alpha,class_prior=[0.5,0.5])
NB.fit(X_tr_tfidf, y_train)     #fitting the NB model with best alpha
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class,not the predicted value

y_train_pred_proba = NB.predict_proba(X_tr_tfidf)
y_test_pred_proba = NB.predict_proba(X_te_tfidf)

#how to get a perticular column in nd array:https://stackoverflow.com/a/8386737/17345549
#[:,[1]]---to take probability values for the positive class
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred_proba[:,[1]]) #find FPR and TPR for plotting roc curve
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred_proba[:,[1]])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))) #finding area under the ROC curve using FPR an
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()
```



In [322]:

```
# 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):  #for plotting confusion matrix we need to give 1 or 0 for prediction instead of prob
                                            #so this function convert prob value to 0 or 1 basen on best threshold

    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)    # if the prob value is equal to or higher than threshold , append 1(consider that as 1), els
        else:
            predictions.append(0)
    return predictions
```

In [323]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr) #finding the best threshold vaule for which AUC value is more
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred_proba[:,[1]], best_t)))   #plot confusion matrix
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred_proba[:,[1]], best_t)))    #order of confusion matrix    #TN,FP
                                                                                                                  #FN,TP
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.44293837677391185 for threshold 0.514
Train confusion matrix
[[ 7103  3070]
 [20777 36050]]
Test confusion matrix
[[ 3310  1700]
 [10486 17504]]
```

In [324]:

```
feature_log_probability_values = NB.feature_log_prob_ #getting feature probability(given any feature what is the probability of p

positive_feature_log_probability_values = feature_log_probability_values[1] #probability values for each feature which helps to p
negative_feature_log_probability_values = feature_log_probability_values[0]#probability values for each feature which helps to pr

#sorting in decending order reference taken from : https://stackoverflow.com/a/16486299/17345549
positive_sorted_index=positive_feature_log_probability_values.argsort()[::-1][:20] #feature with top 20 probability score helping
negative_sorted_index=negative_feature_log_probability_values.argsort()[::-1][:20] #feature with top 20 probability score helping
```

In [325]:

```
print(feature_log_probability_values.shape)  #to make sure the dim of feature prob value vector and feature name vector are same
print(feature_names_with_tfidf.shape)
```

```
(2, 10101)
(10101,)
```

In [326]:

```
#printing those top 20 featuers, which helped to predict the positive classes, with the help of sorted prob index and  already st
#printing those top 20 featuers with the help of soreted prob index and  already store list of feature names
top_20_important_features_for_determining_postivite_class=[]
for i in positive_sorted_index:
    top_20_important_features_for_determining_postivite_class.append(feature_names_with_tfidf[i])

print(top_20_important_features_for_determining_postivite_class)
```

```
['mrs', 'literacy_language', 'grades_prek_2', 'math_science', 'ms', 'grades_3_5', 'literacy', 'mathematics', 'liter
ature_writing', 'grades_6_8', 'ca', 'health_sports', 'specialneeds', 'specialneeds', 'students', 'appliedlearning',
'health_wellness', 'grades_9_12', 'mr', 'music_arts']
```

In [327]:

```
#printing those top 20 featuers, which helps to predict the nagative class, with the help of sorted prob index and  already store
top_20_important_features_for_determining_negative_class=[]
for i in negative_sorted_index:
    top_20_important_features_for_determining_negative_class.append(feature_names_with_tfidf[i])

print(top_20_important_features_for_determining_negative_class)
```

```
['mrs', 'literacy_language', 'math_science', 'grades_prek_2', 'ms', 'grades_3_5', 'literacy', 'mathematics', 'liter
ature_writing', 'grades_6_8', 'specialneeds', 'specialneeds', 'health_sports', 'ca', 'appliedlearning', 'appliedsci
ences', 'students', 'grades_9_12', 'mr', 'music_arts']
```

3. Summary

In [328]:

```
from prettytable import PrettyTable             # Reference Link for Pretty table:  https://pypi.org/project/prettytable/
x = PrettyTable()
```

In [329]:

```
x.field_names = ["Vectorizer", "hyperparamerter(alpha)", "train_AUC","test_AUC"]
x.add_row(["BOW", 1,0.7361149,0.6981477])
x.add_row(["TFIDF", 0.01, 0.72328761,0.69370542])
```

In [330]:

```
print(x)
```

```
+------------+------------------------+------------+------------+
| Vectorizer | hyperparamerter(alpha) | train_AUC  |  test_AUC  |
+------------+------------------------+------------+------------+
|    BOW     |           1            | 0.7361149  | 0.6981477  |
|   TFIDF    |          0.01          | 0.72328761 | 0.69370542 |
+------------+------------------------+------------+------------+
```

we have used two sets of features(set1 and set2) for training two different multinomial NB models.

- set1 had one hot encoded categorical features and minmaxscaled numerican features and Bag of words representated text features.(vectorizer=BOW)
- set2 had one hot encoded categorical features and minmaxscaled numerican features and TFIDF representated text features. (vectorizer=TFIDF)

For the set1 features

- After the stacking features(BOW vectorised) together we fitted the model with the train data and find best hyperparamer which maximizes the AUC using GridsearchCV methood
- Then we fitted the model with the best hyperparamer and printed the top20 features which helps us to classify positive and another top20 features which helps to find negative class

For the set2 features

- repeated the same step as set1

from the table we can see that Multinomial Naive Bayes model with BOW vectorizered text or TFIDF vectorizered text,it performed well with both case

In [ ]: