Assignment 6: Apply NB

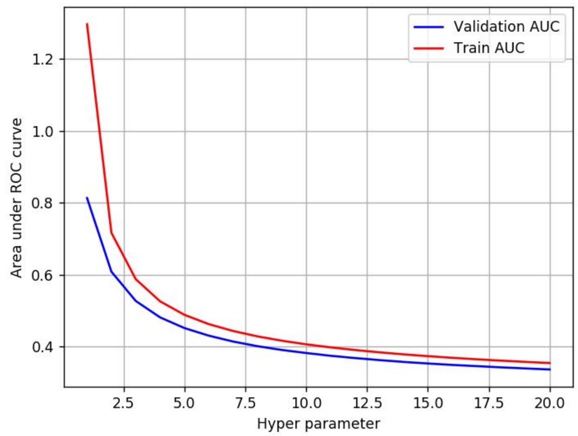
1. Minimum data points need to be considered for people having 4GB RAM is **50k** and for 8GB RAM is **100k**
2. When you are using ramdomsearchcv or gridsearchcv you need not split the data into X_train,X_cv,X_test. As the above methods use kfold. The model will learn better if train data is more so splitting to X_train,X_test will suffice.
3. If you are writing for loops to tune your model then you need split the data into X_train,X_cv,X_test.
4. While splitting the data explore stratify parameter.
5. **Apply Multinomial NB on these feature sets**
    - Features that need to be considered

      essay
        while encoding essay, try to experiment with the max_features and n_grams parameter of vectorizers and see if it increases AUC score.

      categorical features
        - teacher_prefix
        - project_grade_category
        - school_state
        - clean_categories
        - clean_subcategories

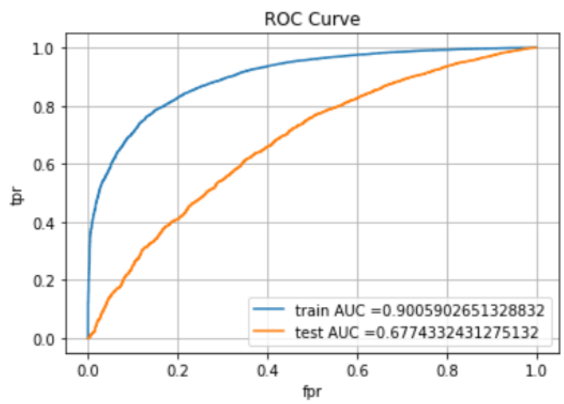
      numerical features
        - price
        - teacher_number_of_previously_posted_projects

        while encoding the numerical features check this and this
    - Set 1: categorical, numerical features + preprocessed_eassay (BOW)
    - Set 2: categorical, numerical features + preprocessed_eassay (TFIDF)
6. **The hyper paramter tuning(find best alpha:smoothing parameter)**
    - Consider alpha values in range: 10^-5 to 10^2 like [0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,100]
    - Explore class_prior = [0.5, 0.5] parameter which can be present in MultinomialNB function(go through this) then check how results might change.
    - Find the best hyper parameter which will give the maximum AUC value
    - For hyper parameter tuning using k-fold cross validation(use GridsearchCV or RandomsearchCV)/simple cross validation data (write for loop to iterate over hyper parameter values)
    - 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



-while plotting take log(alpha) on your X-axis so that it will be more readable

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



- Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points

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

-plot the confusion matrix in heatmaps, while plotting the confusion matrix go through the link

7. find the top 20 features from either from feature Set 1 or feature Set 2 using values of $feature_{\log}\_prob$ parameter of $M\underline{t} \in omialNB$ (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html) and print **BOTH** positive as well as negative corresponding feature names.
   - go through the link

8. 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

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.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')    #Taking all data
data.shape                                         #printing shape of pre processed data
```

Out[3]: (109248, 9)

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

In [4]:
```
y = data['project_is_approved'].values      #storing grounf truth in y
X = data.drop(['project_is_approved'], axis=1)      #dropping y from data set
```

In [5]:
```
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)
```

1.3 Make Data Model Ready: encoding eassay, and project_title

In [6]:
```
feature_names = []  #to store all feature names

print(X_train.shape, y_train.shape)      #checking shape of train
print(X_test.shape, y_test.shape)        #checking shape of test

print("="*100)

print("  BOW representation of feature Essay   ")
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)  #using co
vectorizer.fit(X_train['essay'].values)                                       # fit has

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['essay'].values)   #train set of BOW e
X_test_essay_bow = vectorizer.transform(X_test['essay'].values)     #test set of BOW es

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)   #checking to ensure both test and train
print(X_test_essay_bow.shape, y_test.shape)
print("="*100)


print("  TFIDF representation of feature Essay   ")
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)   #using T
vectorizer.fit(X_train['essay'].values)

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

print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)  #checking to ensure both test and trai
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)

feature_names.extend(vectorizer.get_feature_names())
```

```
(73196, 8) (73196,)
(36052, 8) (36052,)
========================================================================================
============
  BOW representation of feature Essay
After vectorizations
(73196, 5000) (73196,)
(36052, 5000) (36052,)
========================================================================================
============
  TFIDF representation of feature Essay
After vectorizations
(73196, 5000) (73196,)
(36052, 5000) (36052,)
========================================================================================
============
```

1.4 Make Data Model Ready: encoding numerical, categorical features

In [7]:

```
print ("   One Hot Encoding of feature Teacher Prefix")
vectorizer = CountVectorizer()      #count vectorizer for one hot encoding
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_prefix_ohe = vectorizer.transform(X_train['teacher_prefix'].values)   #
X_test_teacher_prefix_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_prefix_ohe.shape, y_train.shape)  #checking to ensure both test a
print(X_test_teacher_prefix_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())       #checking feature names
print("="*100)

feature_names.extend(vectorizer.get_feature_names())  #extending feature names list to

print ("   One Hot Encoding of Project Grade Category  ")
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only on tr

# we use the fitted CountVectorizer to convert the text to vector
X_train_project_grade_category_ohe = vectorizer.transform(X_train['project_grade_catego
X_test_project_grade_category_ohe = vectorizer.transform(X_test['project_grade_category

feature_names.extend(vectorizer.get_feature_names())  #extending feature names list for

print ("   One Hot Encoding of School State  ")

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

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

feature_names.extend(vectorizer.get_feature_names())  #extending feature names list for

print ("   One Hot Encoding of cleaned categories  ")

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_categories_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_test_clean_categories_ohe = vectorizer.transform(X_test['clean_categories'].values)

feature_names.extend(vectorizer.get_feature_names())  #extending feature names list for

print ("   One Hot Encoding of cleaned sub categories  ")

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_subcategories_ohe = vectorizer.transform(X_train['clean_subcategories'].v
X_test_clean_subcategories_ohe = vectorizer.transform(X_test['clean_subcategories'].val
```

```
feature_names.extend(vectorizer.get_feature_names())  #extending feature names list for
```

```
   One Hot Encoding of feature Teacher Prefix
After vectorizations
(73196, 5) (73196,)
(36052, 5) (36052,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
==========================================================================================
============
   One Hot Encoding of Project Grade Category
   One Hot Encoding of School State
   One Hot Encoding of cleaned categories
   One Hot Encoding of cleaned sub categories
```

In [8]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

print(" Normalizing Price feature")

normalizer.fit(X_train['price'].values.reshape(1,-1)) #reshaping to one row from one co

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(1,-1))  #resh
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1))

X_train_price_norm = X_train_price_norm.reshape(-1,1) #reshaping back to
X_test_price_norm = X_test_price_norm.reshape(-1,1)

print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)  #checking to ensure ouput dimensions ar
print(X_test_price_norm.shape, y_test.shape)
print("="*100)

print(" Normalizing Teacher number of previously posted projects feature")

normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1

X_train_teacher_number_of_previously_posted_projects_norm = normalizer.transform(X_trai
X_test_teacher_number_of_previously_posted_projects_norm = normalizer.transform(X_test[

X_train_teacher_number_of_previously_posted_projects_norm =X_train_teacher_number_of_pr
X_test_teacher_number_of_previously_posted_projects_norm = X_test_teacher_number_of_pre

feature_names.append('price')
feature_names.append('teacher_number_of_previously_posted_projects')
```

```
 Normalizing Price feature
After vectorizations
(73196, 1) (73196,)
(36052, 1) (36052,)
==========================================================================================
============
 Normalizing Teacher number of previously posted projects feature
```

In [20]:

```
from scipy.sparse import hstack

X_tr_bow = hstack((X_train_essay_bow,X_train_teacher_prefix_ohe,X_train_project_grade_c
X_te_bow = hstack((X_test_essay_bow,X_test_teacher_prefix_ohe,X_test_project_grade_cate

X_tr_tfidf = hstack((X_train_essay_tfidf,X_train_teacher_prefix_ohe,X_train_project_gra
X_te_tfidf = hstack((X_test_essay_tfidf,X_test_teacher_prefix_ohe,X_test_project_grade_
```

```
print("Final Data matrix BOW")
print(X_tr_bow.shape, y_train.shape)   #final train matrix after horizontally stacking
print(X_te_bow.shape, y_test.shape)    #final test matrix after horizontally stacking a
print("="*100)

print("Final Data matrix TFIDF")
print(X_tr_tfidf.shape, y_train.shape)   #final train matrix after horizontally stackin
print(X_te_tfidf.shape, y_test.shape)    #final test matrix after horizontally stacking
print("="*100)
```

```
Final Data matrix BOW
(73196, 5101) (73196,)
(36052, 5101) (36052,)
====================================================================================================
Final Data matrix TFIDF
(73196, 5101) (73196,)
(36052, 5101) (36052,)
====================================================================================================
```

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

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

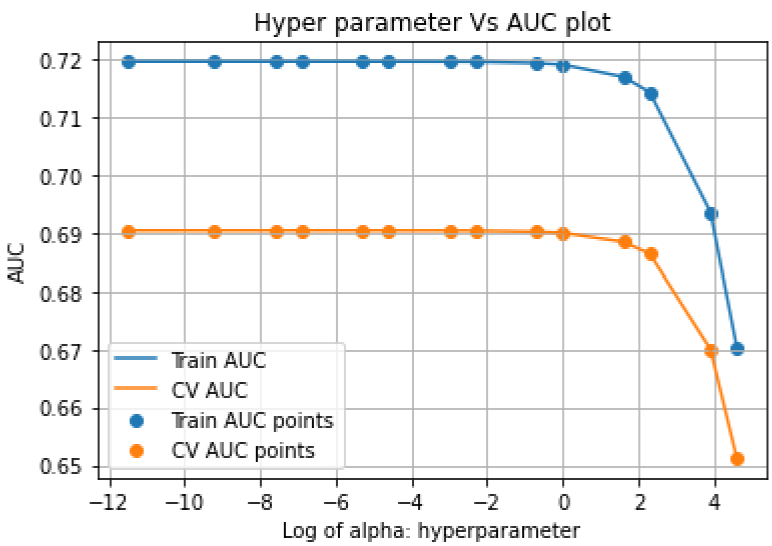
For Every model that you work on make sure you do the step 2 and step 3 of instrucations

Bag of Words

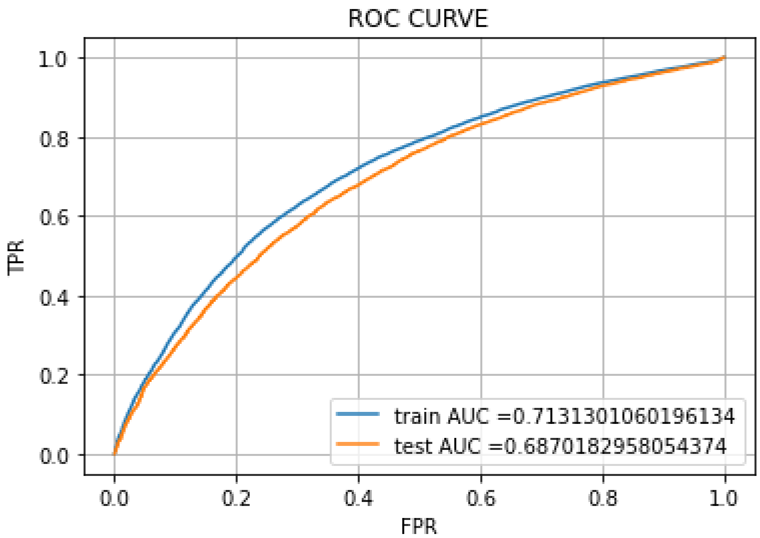
In [77]:
```
import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
import math

mnb = MultinomialNB(class_prior = [0.5,0.5])  #Multinomial NB classifier
mnb.fit(X_tr_bow, y_train)
param_grid = {'alpha':[0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,1
clf = GridSearchCV(mnb, param_grid, cv=4, scoring='roc_auc',return_train_score=True)
clf = clf.fit(X_tr_bow, y_train)

print("Best AUC score : ",clf.fit(X_tr_bow, y_train).best_score_)
print("Best alpha : ",clf.fit(X_tr_bow, y_train).best_params_)


results = pd.DataFrame.from_dict(clf.cv_results_)  #storing results of gridsearch in pa
results = results.sort_values(['rank_test_score'])
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']

lst = []
```

```
for i in K:
    lst.append(math.log(i))    #taking log values of alpha to make the output more read

plt.plot(lst, 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

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

```
Best AUC score :  0.6903754393071408
Best alpha :  {'alpha': 1e-05}
```



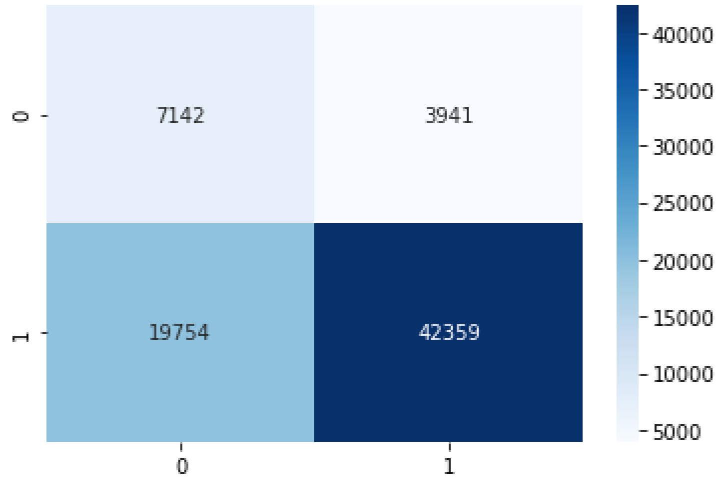
In [79]:
```
from sklearn.metrics import roc_curve, auc


nmb = MultinomialNB(alpha = 0.00001,class_prior = [0.5,0.5])   #using best alpha to tra

nmb.fit(X_tr_bow, y_train)

y_train_pred = nmb.predict_proba(X_tr_bow)[:,1]  # train predicted probabilities
y_test_pred = nmb.predict_proba(X_te_bow)[:,1]   # test predicted probabilities

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)    #train fpr,tr
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)        #test fpr,tes

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))   #p
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()
```



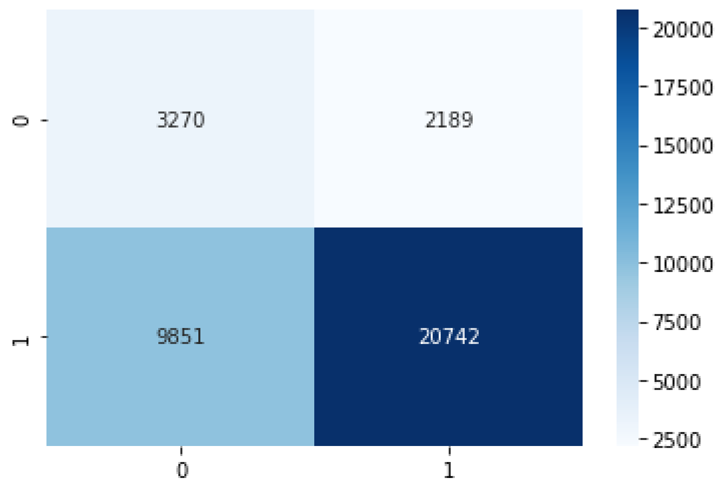
In [73]:

```
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.rou
    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_cm = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
sns.heatmap(train_cm, annot=True,fmt="d",cmap='Blues')
plt.show()

print("Test confusion matrix")
test_cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
sns.heatmap(test_cm, annot=True,fmt="d",cmap='Blues')
plt.show()
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.43946642994637275 for threshold 0.361
Train confusion matrix
```



Test confusion matrix



In [56]:
```
log_prob_nmb = nmb.feature_log_prob_

top_20_0 =  []
top_20_1 =  []

top_20_idx_0 = np.argsort(log_prob_nmb[0])[-20:]  #getting indices of top 20 features o
top_20_idx_1 = np.argsort(log_prob_nmb[1])[-20:]  #getting indices of top 10 features o

for i in top_20_idx_0:
    top_20_0.append(feature_names[i])     #getting feature names based on indices obtai
for j in top_20_idx_1:
    top_20_1.append(feature_names[j])
print("\t\tTop 20 features for SET 1 : BOW\n")
print("Top 20 features of Negative class :\n" ,top_20_0)
print("\n\nTop 20 features of Positive class :\n" ,top_20_1)
```

```
                Top 20 features for SET 1 : BOW

Top 20 features of Negative class :
 ['music_arts', 'mr', 'grades_9_12', 'appliedsciences', 'students', 'appliedlearning',
'ca', 'health_sports', 'specialneeds', 'specialneeds', 'grades_6_8', 'literature_writin
g', 'literacy', 'mathematics', 'grades_3_5', 'ms', 'math_science', 'grades_prek_2', 'lit
eracy_language', 'mrs']


Top 20 features of Positive class :
 ['health_wellness', 'appliedsciences', 'mr', 'grades_9_12', 'appliedlearning', 'special
needs', 'specialneeds', 'students', 'health_sports', 'ca', 'grades_6_8', 'literature_wri
```

```
ting', 'mathematics', 'literacy', 'grades_3_5', 'ms', 'math_science', 'grades_prek_2',
'literacy_language', 'mrs']
```

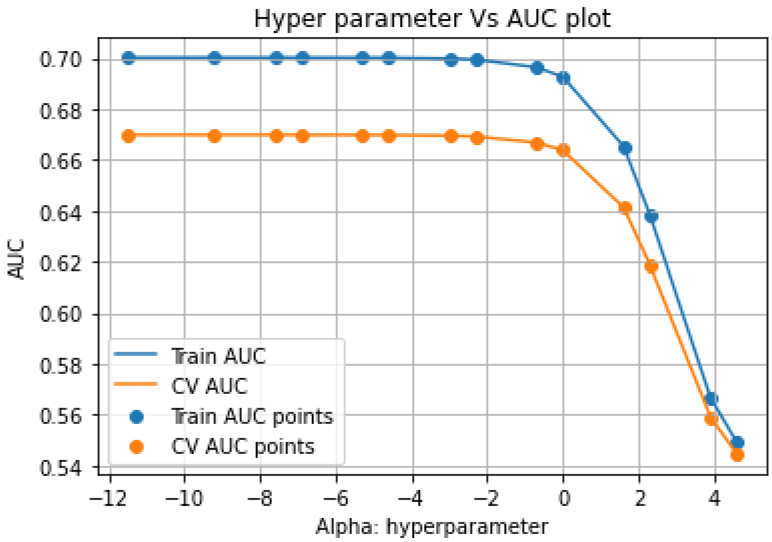
TFIDF

In [83]:

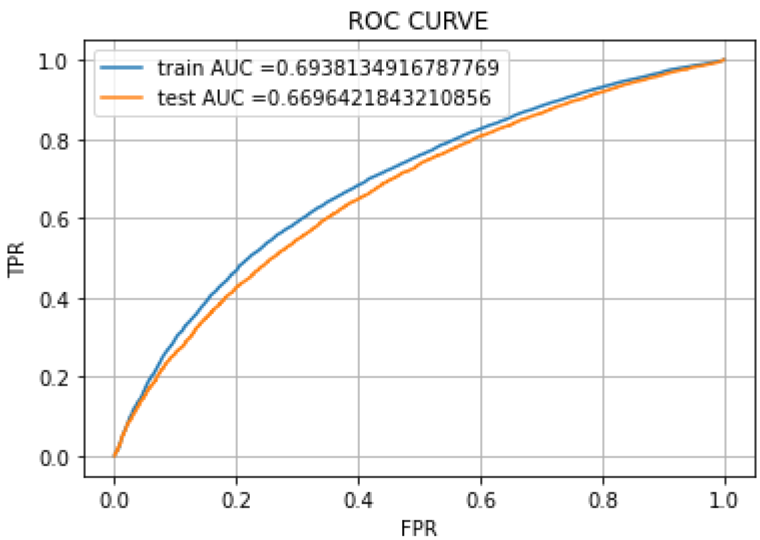
```
### Following similar steps as done for BOW representation of ESSAY
print ("        TFIDF of ESSAY     ")

mnb = MultinomialNB(class_prior = [0.5,0.5])
mnb.fit(X_tr_tfidf, y_train)
param_grid = {'alpha':[0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,1
clf = GridSearchCV(mnb, param_grid, cv=4, scoring='roc_auc',return_train_score=True)
clf.fit(X_tr_tfidf, y_train)

print("Best AUC score : ",clf.fit(X_tr_tfidf, y_train).best_score_)
print("Best alpha : ",clf.fit(X_tr_tfidf, y_train).best_params_)

results = pd.DataFrame.from_dict(clf.cv_results_)

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

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']

lst = []
for i in K:
    lst.append(math.log(i))

plt.plot(lst, 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

plt.plot(lst, 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='da

plt.scatter(lst, train_auc, label='Train AUC points')
plt.scatter(lst, 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()


nmb = MultinomialNB(alpha = 0.00001,class_prior = [0.5,0.5])  #training on best alpha

nmb.fit(X_tr_tfidf, y_train)
```

```
print(X_tr_tfidf.shape)
y_train_pred = nmb.predict_proba(X_tr_tfidf)[:,1]
y_test_pred = nmb.predict_proba(X_te_tfidf)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC CURVE")
plt.grid()
plt.show()

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.rou
    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)

### Printing Confusion Matrix
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
train_cm = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
sns.heatmap(train_cm, annot=True,fmt="d",cmap='Blues')
plt.show()

print("Test confusion matrix")
test_cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
sns.heatmap(test_cm, annot=True,fmt="d",cmap='Blues')
plt.show()
```

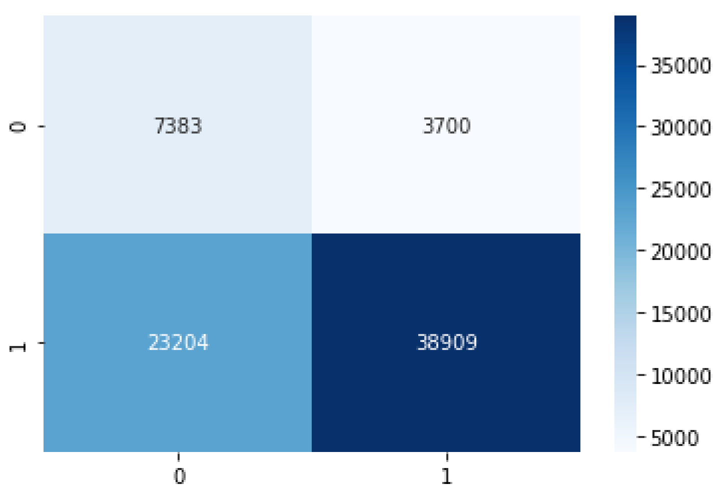
```
        TFIDF of ESSAY
Best AUC score :  0.6698646897625224
Best alpha :  {'alpha': 1e-05}
```



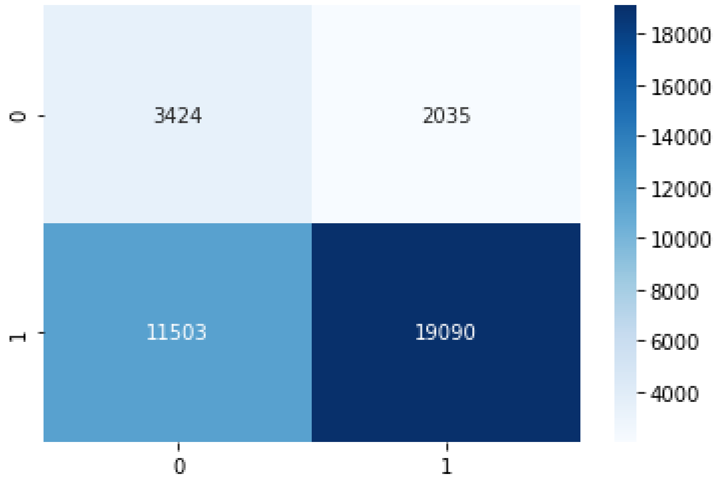

```
(73196, 5101)
```




```
========================================================================================
============
the maximum value of tpr*(1-fpr) 0.41729492073658697 for threshold 0.505
Train confusion matrix
```




```
Test confusion matrix
```



3. Summary

as mentioned in the step 5 of instructions

In [81]:
```
from tabulate import tabulate
print(tabulate([['BOW', 'Multinomial NB' , 0.00001 ,0.69], ['TFIDF', 'Multinomial NB' ,
```

```
Vectorizer    Model             Hyper parameter    AUC
------------  --------------  -----------------  -----
BOW           Multinomial NB              1e-05   0.69
TFIDF         Multinomial NB              1e-05   0.67
```