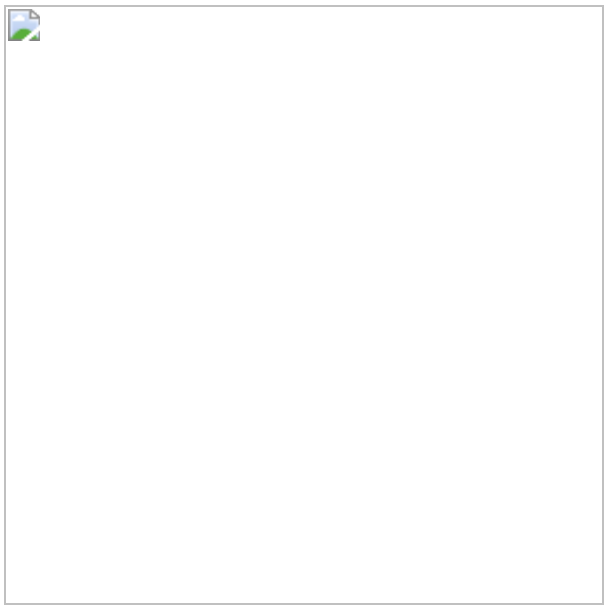
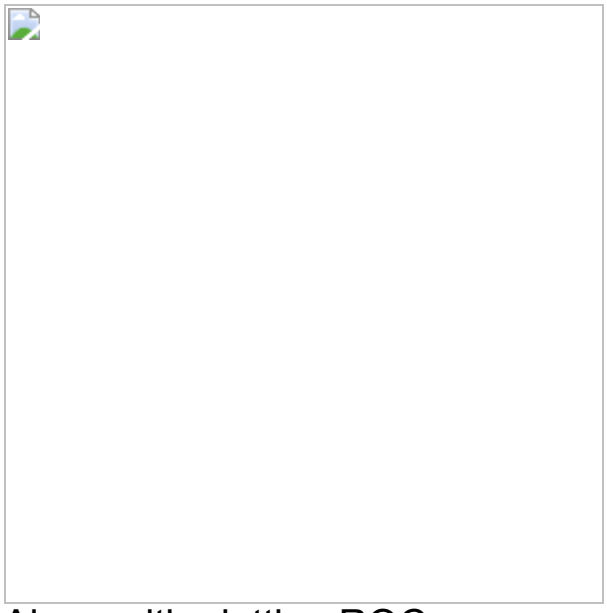
Assignment 6: Apply NB

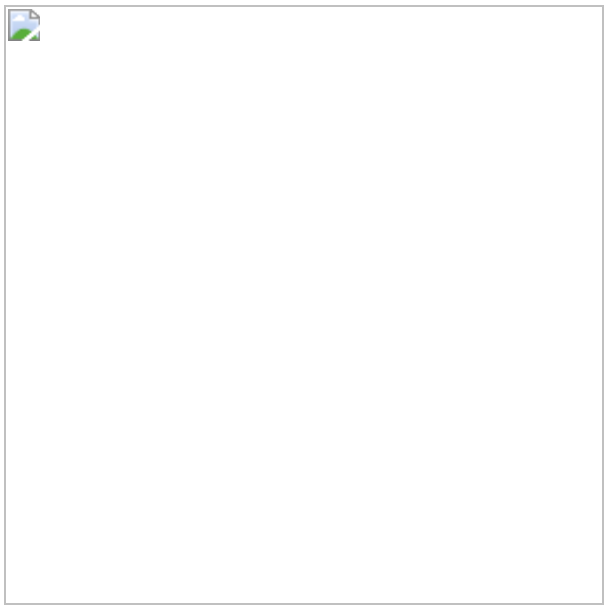
1. **Apply Multinomial NB on these feature sets**
    - Set 1: categorical, numerical features + preprocessed_eassay (BOW)
    - Set 2: categorical, numerical features + preprocessed_eassay (TFIDF)
2. **The hyper paramter tuning(find best alpha:smoothing parameter)**
    - Find the best hyper parameter which will give the maximum AUC value
    - find the best hyper paramter using k-fold cross validation(use GridsearchCV or RandomsearchCV)/simple cross validation data (write for loop to iterate over hyper parameter values)
    -
3. **Representation of results**
    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



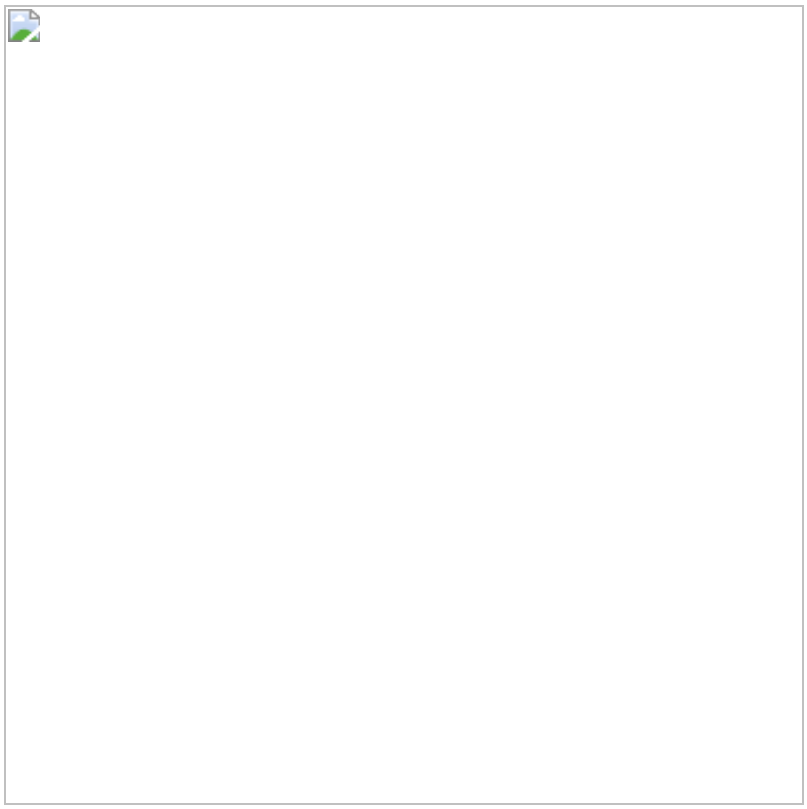
- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.



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



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 $Mt \in omialNB$ (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



2. Naive Bayes

1.1 Loading Data

In [1]:
```
%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 Tfi
```

```
dfVectorizer
from sklearn.feature_extraction.text import Cou
ntVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc

import re
# Tutorial about Python regular expressions: ht
tps://pymotw.com/2/re/

import pickle
from tqdm import tqdm
import os



from collections import Counter
```

In [2]:
```
import pandas
data = pandas.read_csv('preprocessed_data.csv')
```

In [3]:
```
data.columns
data.shape
```

Out[3]:
```
(109248, 9)
```

In [4]:
```
data.head()
```

Out[4]:

	school_state	teacher_prefix	project_gra
0	ca	mrs	grades_pre

	school_state	teacher_prefix	project_gra
1	ut	ms	grades_3_5
2	ca	mrs	grades_pre
3	ga	mrs	grades_pre
4	wa	mrs	grades_3_5

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

In [5]:
```
# please write all the code with proper documen
```

```
tation, and proper titles for each subsection
# go through documentations and blogs before yo
u start coding
# first figure out what to do, and then think a
bout how to do.
# reading and understanding error messages will
be very much helpfull in debugging your code
# 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

from sklearn.model_selection import train_test_
split

from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_s
core
from collections import Counter
from sklearn.metrics import accuracy_score
import numpy as np
```

In [6]:
```
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
```

In [7]:
```
X_train, X_test, y_train, y_test = train_test_s
plit(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_t
rain)
```

In [8]:
```
feature_names_bow=[]
feature_names_tfidf=[]
```

bow of essay

In [9]:
```
bow_essay_vectorizer = CountVectorizer(min_df=10)
bow_essay_vectorizer.fit(X_train['essay'])

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow=bow_essay_vectorizer.transform(X_train['essay'].values)
X_cv_essay_bow=bow_essay_vectorizer.transform(X_cv['essay'].values)
X_test_essay_bow=bow_essay_vectorizer.transform(X_test['essay'].values)


print("Shape of matrix after one hot encodig ",X_train_essay_bow.shape)
print("Shape of matrix after one hot encodig ",X_test_essay_bow.shape)
feature_names_bow.extend(bow_essay_vectorizer.get_feature_names())
```

```
Shape of matrix after one hot encodig  (49041, 12176)

Shape of matrix after one hot encodig  (36052, 12176)
```

In [10]:
```
print("Shape of matrix after one hot encodig ",X_cv_essay_bow.shape)
```

```
Shape of matrix after one hot encodig  (24155, 12176)
```

tfidf of essay

In [11]:
```
tfidf_essay_vectorizer = TfidfVectorizer(min_df
=10)
tfidf_essay_vectorizer.fit(X_train['essay'])

# we use the fitted CountVectorizer to convert
 the text to vector
X_train_essay_tfidf=tfidf_essay_vectorizer.tran
sform(X_train['essay'].values)
X_cv_essay_tfidf=tfidf_essay_vectorizer.transfo
rm(X_cv['essay'].values)
X_test_essay_tfidf=tfidf_essay_vectorizer.trans
form(X_test['essay'].values)


print("Shape of matrix after one hot encodig ",
X_train_essay_tfidf.shape)
print("Shape of matrix after one hot encodig ",
X_cv_essay_tfidf.shape)
print("Shape of matrix after one hot encodig ",
X_test_essay_tfidf.shape)

feature_names_tfidf.extend(tfidf_essay_vectoriz
er.get_feature_names())
```

```
Shape of matrix after one hot encodig  (4
9041, 12176)
Shape of matrix after one hot encodig  (2
4155, 12176)
Shape of matrix after one hot encodig  (3
6052, 12176)
```

one hot encoding and normalizing of categorical and numerical data

In [13]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer_clean_categories = CountVectorizer(lowercase=False, binary=True)
vectorizer_clean_categories.fit(X_train['clean_categories'].values)
feature_names_bow=[]
feature_names_tfidf=[]
# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_categories=vectorizer_clean_categories.transform(X_train['clean_categories'].values)
X_cv_clean_categories=vectorizer_clean_categories.transform(X_cv['clean_categories'].values)
X_test_clean_categories=vectorizer_clean_categories.transform(X_test['clean_categories'].values)


print(vectorizer_clean_categories.get_feature_names())
print("Shape of matrix after one hot encodig ",X_train_clean_categories.shape)
print("Shape of matrix after one hot encodig ",X_cv_clean_categories.shape)
print("Shape of matrix after one hot encodig ",X_test_clean_categories.shape)
feature_names_bow.extend(vectorizer_clean_categories.get_feature_names())
```

```
feature_names_tfidf.extend(vectorizer_clean_categories.get_feature_names())
```

```
['appliedlearning', 'care_hunger', 'healt
h_sports', 'history_civics', 'literacy_la
nguage', 'math_science', 'music_arts', 's
pecialneeds', 'warmth']
Shape of matrix after one hot encodig  (4
9041, 9)
Shape of matrix after one hot encodig  (2
4155, 9)
Shape of matrix after one hot encodig  (3
6052, 9)
```

In [15]:

```
vectorizer_clean_subcategories = CountVectorize
r(lowercase=False, binary=True)
vectorizer_clean_subcategories.fit(X_train['cle
an_subcategories'].values)

# we use the fitted CountVectorizer to convert
 the text to vector
X_train_clean_sub_categories=vectorizer_clean_s
ubcategories.transform(X_train['clean_subcatego
ries'].values)
X_cv_clean_sub_categories=vectorizer_clean_subc
ategories.transform(X_cv['clean_subcategories']
.values)
X_test_clean_sub_categories=vectorizer_clean_su
bcategories.transform(X_test['clean_subcategori
es'].values)


print(vectorizer_clean_subcategories.get_featur
e_names())
print("Shape of matrix after one hot encodig ",
X_train_clean_sub_categories.shape)
```

```
print("Shape of matrix after one hot encodig ",
X_cv_clean_sub_categories.shape)
print("Shape of matrix after one hot encodig ",
X_test_clean_sub_categories.shape)
feature_names_bow.extend(vectorizer_clean_subca
tegories.get_feature_names())
feature_names_tfidf.extend(vectorizer_clean_sub
categories.get_feature_names())
```

```
['appliedsciences', 'care_hunger', 'chara
ctereducation', 'civics_government', 'col
lege_careerprep', 'communityservice', 'ea
rlydevelopment', 'economics', 'environmen
talscience', 'esl', 'extracurricular', 'f
inancialliteracy', 'foreignlanguages', 'g
ym_fitness', 'health_lifescience', 'healt
h_wellness', 'history_geography', 'litera
cy', 'literature_writing', 'mathematics',
'music', 'nutritioneducation', 'other',
'parentinvolvement', 'performingarts', 's
ocialsciences', 'specialneeds', 'teamspor
ts', 'visualarts', 'warmth']
Shape of matrix after one hot encodig  (4
9041, 30)
Shape of matrix after one hot encodig  (2
4155, 30)
Shape of matrix after one hot encodig  (3
6052, 30)
```

In [16]:

```
vectorizer_school_state = CountVectorizer(lower
case=False, binary=True)
vectorizer_school_state.fit(X_train['school_sta
te'].values)

# we use the fitted CountVectorizer to convert
```

```
 the text to vector
X_train_skl_state=vectorizer_school_state.trans
form(X_train['school_state'].values)
X_cv_skl_state=vectorizer_school_state.transfor
m(X_cv['school_state'].values)
X_test_skl_state=vectorizer_school_state.transf
orm(X_test['school_state'].values)


print(vectorizer_school_state.get_feature_names
())
print("Shape of matrix after one hot encodig ",
X_train_skl_state.shape)
print("Shape of matrix after one hot encodig ",
X_cv_skl_state.shape)
print("Shape of matrix after one hot encodig ",
X_test_skl_state.shape)
feature_names_bow.extend(vectorizer_school_stat
e.get_feature_names())
feature_names_tfidf.extend(vectorizer_school_st
ate.get_feature_names())
```

```
['ak', 'al', 'ar', 'az', 'ca', 'co', 'c
t', 'dc', 'de', 'fl', 'ga', 'hi', 'ia',
'id', 'il', 'in', 'ks', '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']
Shape of matrix after one hot encodig  (4
9041, 51)
Shape of matrix after one hot encodig  (2
4155, 51)
Shape of matrix after one hot encodig  (3
6052, 51)
```

In [17]:

```
vectorizer_teacher_prefix = CountVectorizer(lowercase=False, binary=True)
vectorizer_teacher_prefix.fit(X_train['teacher_prefix'].values.astype("U"))

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_prefix=vectorizer_teacher_prefix.transform(X_train['teacher_prefix'].values.astype("U"))
X_cv_teacher_prefix=vectorizer_teacher_prefix.transform(X_cv['teacher_prefix'].values.astype("U"))
X_test_teacher_prefix=vectorizer_teacher_prefix.transform(X_test['teacher_prefix'].values.astype("U"))


print(vectorizer_teacher_prefix.get_feature_names())
print("Shape of matrix after one hot encodig ",X_train_teacher_prefix.shape)
print("Shape of matrix after one hot encodig ",X_cv_teacher_prefix.shape)
print("Shape of matrix after one hot encodig ",X_test_teacher_prefix.shape)
feature_names_bow.extend(vectorizer_teacher_prefix.get_feature_names())
feature_names_tfidf.extend(vectorizer_teacher_prefix.get_feature_names())
```

```
['dr', 'mr', 'mrs', 'ms', 'teacher']
Shape of matrix after one hot encodig  (49041, 5)
Shape of matrix after one hot encodig  (24155, 5)
```

```
4155, 5)
Shape of matrix after one hot encodig  (3
6052, 5)
```

In [18]:

```
vectorizer_project_grade_category = CountVectorizer(lowercase=False, binary=True)
vectorizer_project_grade_category.fit(X_train['project_grade_category'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_level=vectorizer_project_grade_category.transform(X_train['project_grade_category'].values)
X_cv_grade_level=vectorizer_project_grade_category.transform(X_cv['project_grade_category'].values)
X_test_grade_level=vectorizer_project_grade_category.transform(X_test['project_grade_category'].values)


print(vectorizer_project_grade_category.get_feature_names())
print("Shape of matrix after one hot encodig ", X_train_grade_level.shape)
print("Shape of matrix after one hot encodig ", X_cv_grade_level.shape)
print("Shape of matrix after one hot encodig ", X_test_grade_level.shape)
feature_names_bow.extend(vectorizer_project_grade_category.get_feature_names())
feature_names_tfidf.extend(vectorizer_project_grade_category.get_feature_names())
```

```
['grades_3_5', 'grades_6_8', 'grades_9_1
2', 'grades_prek_2']
```

```
2 ,  grades_prek_2 ]
Shape of matrix after one hot encodig  (4
9041, 4)
Shape of matrix after one hot encodig  (2
4155, 4)
Shape of matrix after one hot encodig  (3
6052, 4)
```

In [19]:

```
from sklearn.preprocessing import Normalizer

normalizer = Normalizer()

# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, go
t 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53
709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a singl
e feature
# array.reshape(1, -1)  if it contains a single
sample.

normalizer.fit(X_train['price'].values.reshape(
1,-1))

X_train_price_standardized = normalizer.transfo
rm(X_train['price'].values.reshape(1,-1))
X_cv_price_standardized = normalizer.transform(
X_cv['price'].values.reshape(1,-1))
X_test_price_standardized = normalizer.transfor
m(X_test['price'].values.reshape(1,-1))

print("After vectorizations")
print(X_train_price_standardized.shape, y_train
.shape)
```

```
print(X_cv_price_standardized.shape, y_cv.shape
)
print(X_test_price_standardized.shape, y_test.s
hape)
```

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

In [20]:

```
X_train_price_standardized = X_train_price_stan
dardized.reshape(-1,1)
X_cv_price_standardized = X_cv['price'].values.
reshape(-1,1)
X_test_price_standardized = X_test_price_standa
rdized.reshape(-1,1)
```

In [21]:

```
normalizer = Normalizer()

# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, go
t 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53
709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a singl
e feature
# array.reshape(1, -1)  if it contains a single
sample.

normalizer.fit(X_train['teacher_number_of_previ
ously_posted_projects'].values.reshape(1,-1))

X_train_prev_proj = normalizer.transform(X_trai
n['teacher_number_of_previously_posted_project
```

```
s'].values.reshape(1,-1))
X_cv_prev_proj = normalizer.transform(X_cv['tea
cher_number_of_previously_posted_projects'].val
ues.reshape(1,-1))
X_test_proj = normalizer.transform(X_test['teac
her_number_of_previously_posted_projects'].valu
es.reshape(1,-1))

print("After vectorizations")
print(X_train_prev_proj.shape, y_train.shape)
print(X_cv_prev_proj.shape, y_cv.shape)
print(X_test_proj.shape, y_test.shape)
```

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

In [22]:

```
X_train_prev_proj = X_train_prev_proj.reshape(-
1,1)
X_cv_prev_proj = X_cv['teacher_number_of_previo
usly_posted_projects'].values.reshape(-1,1)
X_test_proj = X_test_proj.reshape(-1,1)
```

merge bow

In [23]:

```
from scipy.sparse import hstack

# with the same hstack function we are concatin
ating a sparse matrix and a dense matirx :)
X_train_bow = hstack((X_train_clean_categories,
X_train_clean_sub_categories,X_train_skl_state,
X_train_teacher_prefix,
            X_train_grade_level,X_train_prev_pr
```

```
oj,X_train_price_standardized,
            X_train_essay_bow
            )).tocsr()

X_cv_bow = hstack((X_cv_clean_categories, X_cv_
clean_sub_categories,X_cv_skl_state,X_cv_teache
r_prefix,
            X_cv_grade_level,X_cv_prev_proj,X_c
v_price_standardized,
            X_cv_essay_bow
            )).tocsr()

X_test_bow = hstack((X_test_clean_categories, X
_test_clean_sub_categories,X_test_skl_state,X_t
est_teacher_prefix,
            X_test_grade_level,X_test_proj,X_te
st_price_standardized,
            X_test_essay_bow
            )).tocsr()




print(X_train_bow.shape)
print(X_cv_bow.shape)
print(X_test_bow.shape)
```

```
(49041, 12277)
(24155, 12277)
(36052, 12277)
```

merge tfidf

```

```

In [24]:

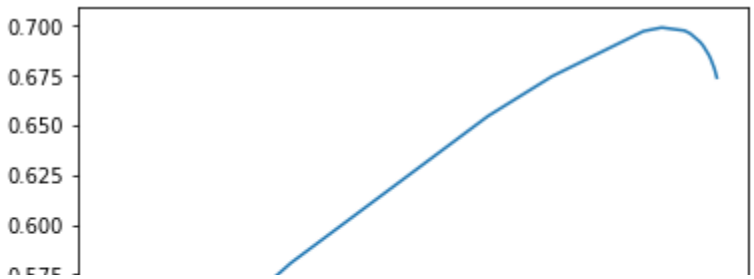
```
# with the same hstack function we are concatin
ating a sparse matrix and a dense matirx :)
X_train_tfidf = hstack((X_train_clean_categorie
s, X_train_clean_sub_categories,X_train_skl_sta
te,X_train_teacher_prefix,
            X_train_grade_level,X_train_prev_pr
oj,X_train_price_standardized,
            X_train_essay_tfidf
            )).tocsr()

X_cv_tfidf = hstack((X_cv_clean_categories, X_c
v_clean_sub_categories,X_cv_skl_state,X_cv_teac
her_prefix,
            X_cv_grade_level,X_cv_prev_proj,X_c
v_price_standardized,
            X_cv_essay_tfidf
            )).tocsr()

X_test_tfidf = hstack((X_test_clean_categories,
X_test_clean_sub_categories,X_test_skl_state,X_
test_teacher_prefix,
            X_test_grade_level,X_test_proj,X_te
st_price_standardized,
            X_test_essay_tfidf
            )).tocsr()




print(X_train_bow.shape)
print(X_cv_bow.shape)
print(X_test_bow.shape)
```

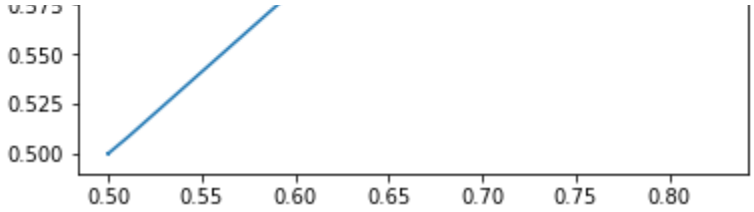
```
(49041, 12277)
(24155, 12277)
```

(36052, 12277)

apply naive bayes on bow

In [25]:
```
from sklearn.model_selection import GridSearchC
V
from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB(class_prior=[0.5,0.5])

parameters = {'alpha':[0.00001, 0.00005, 0.0001
, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1
, 5, 10, 50, 100, 500, 1000, 2500, 5000, 10000
]}

clf = GridSearchCV(nb, parameters, cv= 10, scor
ing='roc_auc',return_train_score=True)

clf.fit(X_train_bow, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std = clf.cv_results_['std_train_scor
e']
test_auc = clf.cv_results_['mean_test_score']
test_auc_std = clf.cv_results_['std_test_score'
]
plt.plot(train_auc,test_auc)
plt.show()
```





Gridsearch hyperparameter tuning

In [26]:
```
#Output of GridSearchCV
print('Best score: ',clf.best_score_)
print('parameter value with best score: ',clf.b
est_params_)
print('Train AUC scores')
print(clf.cv_results_['mean_train_score'])
print('CV AUC scores')
print(clf.cv_results_['mean_test_score'])
```

```
Best score:  0.699126174818557
parameter value with best score:  {'alph
a': 0.5}
Train AUC scores
[0.82551071 0.82453779 0.82398095 0.82225
726 0.82128641 0.81832476
 0.81665678 0.81141549 0.80825202 0.79608
6   0.78630743 0.7374272
 0.70313568 0.59775327 0.51100459 0.50005
532 0.5        0.5
 0.50005532 0.50004501]
CV AUC scores

[0.67398328 0.67746919 0.67914759 0.68334
092 0.68528564 0.68983848
 0.69180399 0.69606457 0.69759251 0.69912
617 0.6973239  0.67461645
```

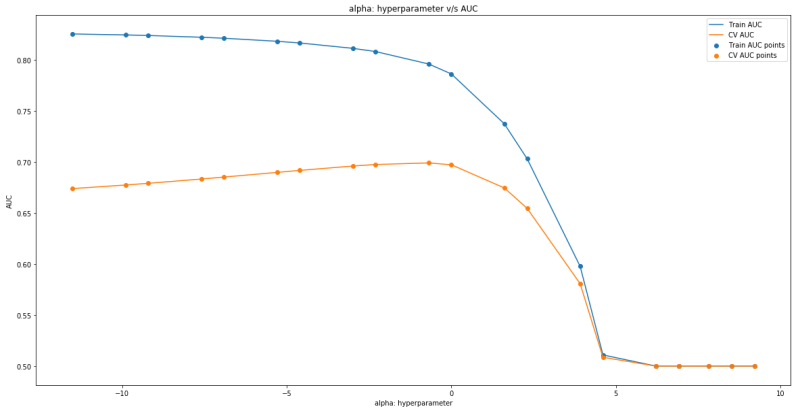
```
 0.65460986 0.58079892 0.50867716 0.50005
535 0.5        0.5
 0.50005535 0.50004369]
```

In [27]:

```
import math

params = [0.00001, 0.00005, 0.0001, 0.0005, 0.0
01, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50,
100, 500, 1000, 2500, 5000, 10000]

log_alphas =[]
for a in tqdm(params):
    b = math.log(a)
    log_alphas.append(b)

plt.figure(figsize=(20,10))

plt.plot(log_alphas, train_auc, label='Train AU
C')



plt.plot(log_alphas, test_auc, label='CV AUC')



plt.scatter(log_alphas,train_auc, label='Train
 AUC points')
plt.scatter(log_alphas, test_auc, label='CV AUC
points')


plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
```

```
plt.title("alpha: hyperparameter v/s AUC")

plt.show()
```

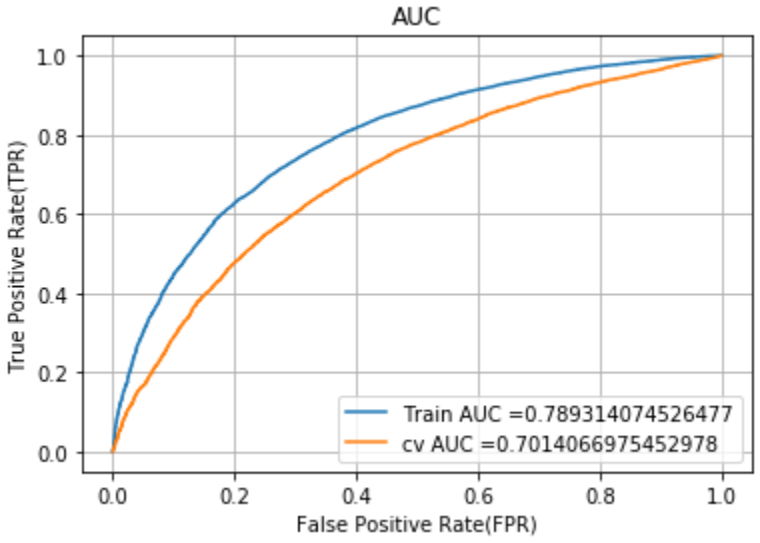
```
100%|██████████| 20/20 [00:00<00:00, 2015
5.23it/s]
```



In [28]:

```
def pred_prob(clf, data):
    y_pred = []
    y_pred = clf.predict_proba(data)[:,1]
    return y_pred
from sklearn.metrics import roc_curve, auc

nb_bow = MultinomialNB(alpha = clf.best_params_
['alpha'],class_prior = [0.5,0.5])

nb_bow.fit(X_train_bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parame
ter should be probability estimates of the posi
tive class
# not the predicted outputs

y_train_pred = pred_prob(nb_bow,X_train_bow)
y_test_pred = pred_prob(nb_bow,X_test_bow)

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="cv AUC ="+s
tr(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



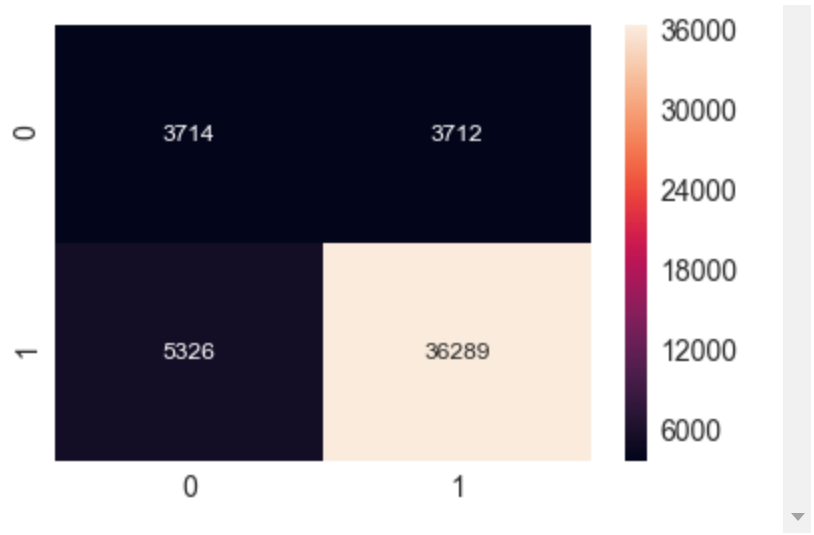
In [29]:
```
def prediction(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(fpr*(1-tpr))]

    # (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)", m
ax(tpr*(1-fpr)), "for threshold", np.round(t,3
))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [30]:
```
conf_matr_df_train_1 = pd.DataFrame(confusion_m
atrix(y_train, prediction(y_train_pred, tr_thre
sholds, train_fpr, train_fpr)), range(2),range(
2))
```

```
the maximum value of tpr*(1-fpr) 0.249999
9818661462 for threshold 0.037
```

In [31]:
```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_1, annot=True,an
not_kws={"size": 12}, fmt='g')
```

Out[31]:
```
<matplotlib.axes._subplots.AxesSubplot at
0x26b3b8e5358>
```



to get top 20 features

In [32]:

```
bow_features_names = []

for feature in vectorizer_clean_categories.get_
feature_names() :
    bow_features_names.append(feature)

for feature in vectorizer_clean_subcategories.g
et_feature_names() :
    bow_features_names.append(feature)

for feature in vectorizer_school_state.get_feat
ure_names() :
    bow_features_names.append(feature)

for feature in vectorizer_project_grade_categor
y.get_feature_names() :
    bow_features_names.append(feature)

for feature in vectorizer_teacher_prefix.get_fe
ature_names() :
```

```
    bow_features_names.append(feature)
for feature in bow_essay_vectorizer.get_feature
_names() :
    bow_features_names.append(feature)

bow_features_names.append("price")
```

In [37]:

```
neg_class_prob_sorted = nb_bow.feature_log_prob
_[0, :].argsort()   #class 0
pos_class_prob_sorted = nb_bow.feature_log_prob
_[1, :].argsort() #class1

print('Top 20 features from negative class:')
print(np.take(bow_features_names, neg_class_pro
b_sorted[len(neg_class_prob_sorted)-20:]))

print('Top 20 features from positive class:')
print(np.take(bow_features_names, pos_class_pro
b_sorted[len(pos_class_prob_sorted )-20:]))
```

```
Top 20 features from negative class:
['skip' 'abound' 'reads' 'loved' 'comes'
'workbook' 'neediest' 'weaker'
 'mapping' 'napkins' 'theaters' 'helper'
'thief' 'learner' 'notate'
 'classwork' 'myers' 'learniture' 'school
ers' 'studies']
Top 20 features from positive class:
['days' 'abound' 'loved' 'used' 'reads'
'workbook' 'neediest' 'weaker'
 'napkins' 'mapping' 'helper' 'learner'
'notate' 'thief' 'theaters'
 'classwork' 'learniture' 'myers' 'school
ers' 'studies']
```

In [34]: