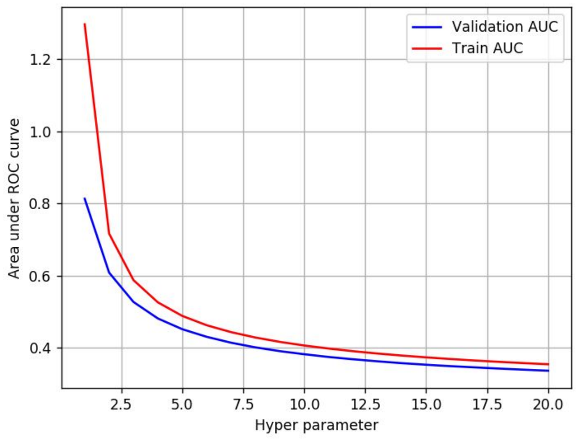
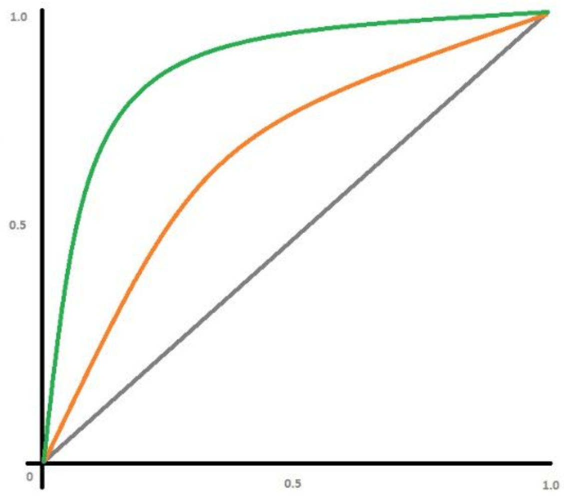
Assignment 6: Apply NB

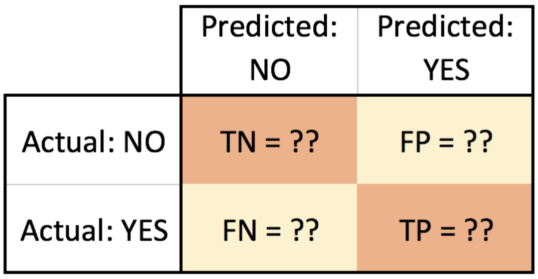
1. **Apply Multinomial NB on these feature sets**
    - Set 1: categorical, numerical features + preprocessed_eassay (BOW)
    - Set 2: categorical, numerical features + preprocessed_eassay (TFIDF)
2. **The hyper paramter tuning(find best alpha:smoothing parameter)**
    - Find the best hyper parameter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



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

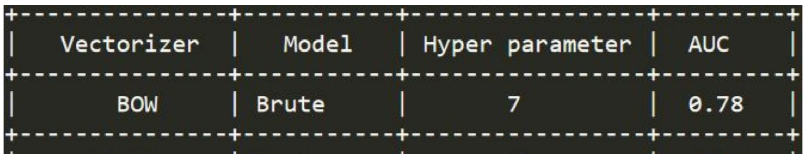


    - Along with plotting ROC curve, you need to print the confusion matrix (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points



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

4. fine the top 20 features from either from feature Set 1 or feature Set 2 using absolute values of \`feature_log_prob_ \` parameter of \`MultinomialNB\` (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html) and print their corresponding feature names
5. You need to summarize the results at the end of the notebook, summarize it in the table format



```
+----------------+-----------+------------------+---------+
|   Vectorizer   |   Model   | Hyper parameter  |   AUC   |
+----------------+-----------+------------------+---------+
|      BOW       | Brute     |        7         | 0.78    |
+----------------+-----------+------------------+---------+
```

2. Naive Bayes

1.1 Loading Data

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from gensim.models import KeyedVectors
from tqdm import tqdm
import os
from collections import Counter
```

In [2]:

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

data.columns.values
```

Out[2]:

```
array(['school_state', 'teacher_prefix', 'project_grade_category',
       'teacher_number_of_previously_posted_projects',
       'project_is_approved', 'clean_categories', 'clean_subcategories',
       'essay', 'price', 'project_title'], dtype=object)
```

In [3]:

```
data.shape
```

Out[3]:

```
(50000, 10)
```

Preprocess categorical features PROJECT TITLE

1) project title

In [4]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you'r
e", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 't
hey', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
at'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
d', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as'
, 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through'
, 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'ov
er', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
y', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too'
, 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
w', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'migh
tn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
asn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
from tqdm import tqdm
def preprocess_text(text_data):
    preprocessed_text = []
    # tqdm is for printing the status bar
    for sentance in tqdm(text_data):
        sent = decontracted(sentance)
        sent = sent.replace('\\r', ' ')
        sent = sent.replace('\\n', ' ')
        sent = sent.replace('\\"', ' ')
        sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
        # https://gist.github.com/sebleier/554280
        sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
        preprocessed_text.append(sent.lower().strip())
    return preprocessed_text
```

In [5]:

```
preprocessed_titles = preprocess_text(data['project_title'].values)
```

```
100%|████████████████████████████████████████████████████████████████████
████| 50000/50000 [00:01<00:00, 38395.08it/s]
```

creating new column with preprocessed project title values.

In [6]:

```
data['cleaned_project_title']=preprocessed_titles
```

In [7]:

```
data=data.drop('project_title',axis=1)
data.head(5)
```

Out[7]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previous
0	ca	mrs	grades_prek_2	53
1	ut	ms	grades_3_5	4
2	ca	mrs	grades_prek_2	10
3	ga	mrs	grades_prek_2	2
4	wa	mrs	grades_3_5	2

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

In [8]:

```
y = data['project_is_approved'].values
x = data.drop(['project_is_approved'], axis=1)
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.33, stratify=y)
```

In [9]:

```



print(xtrain.shape)
print(xtest.shape)
print('-'*100)
print(ytrain.shape)
print(ytest.shape)
```

```
(33500, 9)
(16500, 9)
---------------------------------------------------------------------------
-------------------------
(33500,)
(16500,)
```

1)we have 2 text features "essay" and "cleaned_project_title". we encode/vectorise it using BOW and TFIDF

1.3 Make Data Model Ready: encoding essay, and project_title

1)BOW vectorization

In [10]:

```
essay_bow=xtrain['essay'].values
```

In [11]:

```
vect=CountVectorizer(min_df=10,ngram_range=(1,4),max_features=5000)
vect.fit(essay_bow)
#now we have learnt the vocab of xtrain.lets try to transofrm test and cv data using xt
rain vocab
xtr_bow_essay=vect.transform(essay_bow)
xtest_bow_essay=vect.transform(xtest['essay'].values)
#xcv_bow_essay=vect.transform(xcv['essay'].values)
essay_bow_features=vect.get_feature_names()

print('train_essay bow shape',xtr_bow_essay.shape,ytrain.shape)
print('test_essay bow shape',xtest_bow_essay.shape,ytest.shape)
#print('cv_essay bow shape',xcv_bow_essay.shape,ycv.shape)
```

```
train_essay bow shape (33500, 5000) (33500,)
test_essay bow shape (16500, 5000) (16500,)
```

In [12]:

```
print('cv_essay bow feature names',essay_bow_features[:10])
```

```
cv_essay bow feature names ['000', '10', '100', '100 free', '100 percent',
'100 students', '100 students receive', '100 students receive free', '11',
'12']
```

BOW vectorization project_title

In [13]:

```
prjtitle_bow=xtrain['cleaned_project_title'].values
```

In [14]:

```
vect=CountVectorizer(min_df=10,ngram_range=(1,4),max_features=5000)
vect.fit(prjtitle_bow)
#now we have learnt the vocab of xtrain.lets try to transform test and cv data using xt
rain vocab
xtr_bow_prjtitle=vect.transform(prjtitle_bow)
xtest_bow_prjtitle=vect.transform(xtest['cleaned_project_title'].values)
#xcv_bow_prjtitle=vect.transform(xcv['cleaned_project_title'].values)
prjtitle_bow_features=vect.get_feature_names()

print('train_prj title bow shape',xtr_bow_prjtitle.shape,ytrain.shape)
print('test_prj title bow shape',xtest_bow_prjtitle.shape,ytest.shape)
#print('cv_prj title bow shape',xcv_bow_prjtitle.shape,ycv.shape)
print(prjtitle_bow_features[:10])
```

```
train_prj title bow shape (33500, 2427) (33500,)
test_prj title bow shape (16500, 2427) (16500,)
['05', '10', '100', '101', '16', '17', '1st', '1st grade', '1st graders',
'2016']
```

1.4 Make Data Model Ready: encoding numerical, categorical features

1)school_state (one hot encoding)

In [15]:

```
vect=CountVectorizer(binary=True)
vect.fit(xtrain['school_state'].values)
xtrain_ohe_school_state=vect.transform(xtrain['school_state'].values)
#xcv_ohe_school_state=vect.transform(xcv['school_state'].values)
xtest_ohe_school_state=vect.transform(xtest['school_state'].values)
school_state_feature=vect.get_feature_names()

print('school_state xtrain shape',xtrain_ohe_school_state.shape,ytrain.shape)
print('school_state xtest shape',xtest_ohe_school_state.shape,ytest.shape)
#print('school_state xcv shape',xcv_ohe_school_state.shape,ycv.shape)
```

```
school_state xtrain shape (33500, 51) (33500,)
school_state xtest shape (16500, 51) (16500,)
```

2)teacher_prefix one hot encoding

In [16]:

```
vect=CountVectorizer(binary=True)
vect.fit(xtrain['teacher_prefix'].values)
xtrain_ohe_teacher_prefix=vect.transform(xtrain['teacher_prefix'].values)
xtest_ohe_teacher_prefix=vect.transform(xtest['teacher_prefix'].values)
teacher_prefix_feature=vect.get_feature_names()
#xcv_ohe_teacher_prefix=vect.transform(xcv['teacher_prefix'].values)
print('xtrain_ohe_teacher shape',xtrain_ohe_teacher_prefix.shape,ytrain.shape)
print('xtest_ohe_teacher shape',xtest_ohe_teacher_prefix.shape,ytest.shape)
#print('xcv_ohe_teacher_prefix shape',xcv_ohe_teacher_prefix.shape,ycv.shape)
```

```
xtrain_ohe_teacher shape (33500, 5) (33500,)
xtest_ohe_teacher shape (16500, 5) (16500,)
```

3)project_grade_category ohe hot encoding

In [17]:

```
vect=CountVectorizer(binary=True)
vect.fit(xtrain['project_grade_category'].values)
xtrain_ohe_project_grade_category=vect.transform(xtrain['project_grade_category'].value
s)
xtest_ohe_project_grade_category=vect.transform(xtest['project_grade_category'].values)
project_grade_category_feature=vect.get_feature_names()
#xcv_ohe_project_grade_category=vect.transform(xcv['project_grade_category'].values)
print('xtrain_ohe_project_grade_category shape',xtrain_ohe_project_grade_category.shape
,ytrain.shape)
print('xtest_ohe_project_grade_category shape',xtest_ohe_project_grade_category.shape,y
test.shape)
#print('xcv_ohe_teacher_project_grade_category shape',xcv_ohe_project_grade_category.sh
ape,ycv.shape)
```

```
xtrain_ohe_project_grade_category shape (33500, 4) (33500,)
xtest_ohe_project_grade_category shape (16500, 4) (16500,)
```

4)clean_categories one hot encoding

In [18]:

```
vect=CountVectorizer(binary=True)
vect.fit(xtrain['clean_categories'].values)
xtrain_ohe_clean_categories=vect.transform(xtrain['clean_categories'].values)
xtest_ohe_clean_categories=vect.transform(xtest['clean_categories'].values)
#xcv_ohe_clean_categories=vect.transform(xcv['clean_categories'].values)
clean_categories_feature=vect.get_feature_names()
print('xtrain_ohe_clean_categories shape',xtrain_ohe_clean_categories.shape,ytrain.shap
e)
print('xtest_ohe_clean_categories shape',xtest_ohe_clean_categories.shape,ytest.shape)
#print('xcv_ohe_teacher_clean_categories shape',xcv_ohe_clean_categories.shape,ycv.shap
e)
print(vect.get_feature_names())
```

```
xtrain_ohe_clean_categories shape (33500, 9) (33500,)
xtest_ohe_clean_categories shape (16500, 9) (16500,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'lit
eracy_language', 'math_science', 'music_arts', 'specialneeds', 'warmth']
```

5)clean_subcategories one hot encoding

In [19]:

```
vect=CountVectorizer(binary=True)
vect.fit(xtrain['clean_subcategories'].values)
xtrain_ohe_clean_subcategories=vect.transform(xtrain['clean_subcategories'].values)
xtest_ohe_clean_subcategories=vect.transform(xtest['clean_subcategories'].values)
#xcv_ohe_clean_subcategories=vect.transform(xcv['clean_subcategories'].values)
clean_subcategories_feature=vect.get_feature_names()
print('xtrain_ohe_clean_subcategories shape',xtrain_ohe_clean_subcategories.shape,ytrai
n.shape)
print('xtest_ohe_clean_subcategories shape',xtest_ohe_clean_subcategories.shape,ytest.s
hape)
#print('xcv_ohe_teacher_clean_subcategories shape',xcv_ohe_clean_subcategories.shape,yc
v.shape)
print(vect.get_feature_names())
```

```
xtrain_ohe_clean_subcategories shape (33500, 30) (33500,)
xtest_ohe_clean_subcategories shape (16500, 30) (16500,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_governmen
t', 'college_careerprep', 'communityservice', 'earlydevelopment', 'economi
cs', 'environmentalscience', 'esl', 'extracurricular', 'financialliterac
y', 'foreignlanguages', 'gym_fitness', 'health_lifescience', 'health_welln
ess', 'history_geography', 'literacy', 'literature_writing', 'mathematic
s', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performi
ngarts', 'socialsciences', 'specialneeds', 'teamsports', 'visualarts', 'wa
rmth']
```

Normalize numerical features "price" and "previosuly submitted projects count"

1)price

In [25]:

```
from sklearn.preprocessing import Normalizer
norm=Normalizer()
norm.fit(xtrain['price'].values.reshape(1,-1))
xtrain_norm_price=norm.transform(xtrain['price'].values.reshape(1,-1))
xtest_norm_price=norm.transform(xtest['price'].values.reshape(1,-1))

#xcv_norm_price=norm.transform(xcv['price'].values.reshape(-1,1))
print('xtrain_norm_price shape',xtrain_norm_price.shape,ytrain.shape)
print('xtest_norm_price shape',xtest_norm_price.shape,ytest.shape)
#print('xcv_norm_price shape',xcv_norm_price.shape,ycv.shape)
```

```
xtrain_norm_price shape (1, 33500) (33500,)
xtest_norm_price shape (1, 16500) (16500,)
```

2)teacher_number_of_previously_posted_projects

In [29]:

```
from sklearn.preprocessing import Normalizer
norm=Normalizer()
norm.fit(xtrain['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
xtrain_norm_prjcount=norm.transform(xtrain['teacher_number_of_previously_posted_project
s'].values.reshape(1,-1))
xtest_norm_prjcount=norm.transform(xtest['teacher_number_of_previously_posted_projects'
].values.reshape(1,-1))
#xcv_norm_prjcount=norm.transform(xcv['teacher_number_of_previously_posted_projects'].v
alues.reshape(-1,1))
print('xtrain_norm_prjcount shape',xtrain_norm_prjcount.shape,ytrain.shape)
print('xtest_norm_prjcount shape',xtest_norm_prjcount.shape,ytest.shape)
#print('xcv_norm_prjcount shape',xcv_norm_prjcount.shape,ycv.shape)
```

```
xtrain_norm_prjcount shape (1, 33500) (33500,)
xtest_norm_prjcount shape (1, 16500) (16500,)
```

1) https://stackoverflow.com/a/19710648/4084039 (https://stackoverflow.com/a/19710648/4084039) : concatenate two matrices

lets combine all the features into one single stack.

SET1

In [30]:

```
xtrain_norm_price=xtrain_norm_price.reshape(-1,1)
xtest_norm_price=xtest_norm_price.reshape(-1,1)
xtrain_norm_prjcount=xtrain_norm_prjcount.reshape(-1,1)
xtest_norm_prjcount=xtest_norm_prjcount.reshape(-1,1)



from scipy.sparse import hstack
xtrain_hstack_set1 = hstack((xtr_bow_essay,xtr_bow_prjtitle,xtrain_ohe_school_state, xt
rain_ohe_teacher_prefix, xtrain_ohe_project_grade_category,xtrain_ohe_clean_categories,
xtrain_ohe_clean_subcategories,xtrain_norm_price,xtrain_norm_prjcount)).tocsr()

xtest_hstack_set1 = hstack((xtest_bow_essay,xtest_bow_prjtitle,xtest_ohe_school_state,x
test_ohe_teacher_prefix,xtest_ohe_project_grade_category,xtest_ohe_clean_categories,xte
st_ohe_clean_subcategories,xtest_norm_price,xtest_norm_prjcount)).tocsr()

#xcv_hstack_set1=hstack((xcv_bow_essay,xcv_bow_prjtitle,xcv_ohe_school_state,xcv_ohe_te
acher_prefix,xcv_ohe_project_grade_category,xcv_ohe_clean_categories,xcv_ohe_clean_subc
ategories,xcv_norm_price,xcv_norm_prjcount)).tocsr()
print("Data Final matrix")
print(xtrain_hstack_set1.shape, ytrain.shape)
#print(xtest_hstack_set1.shape, ytest.shape)
#print(xcv_hstack_set1.shape, ycv.shape)
```

```
Data Final matrix
(33500, 7528) (33500,)
```

TFIDF vectorization

1)essay

In [31]:

```
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [32]:

```
vect=TfidfVectorizer(min_df=10,ngram_range=(1,4),max_features=5000)
vect.fit(xtrain['essay'].values)
#now we have learnt the vocab of xtrain.lets try to transofrm test and cv data using xt
rain vocab
xtr_tfidf_essay=vect.transform(xtrain['essay'].values)
xtest_tfidf_essay=vect.transform(xtest['essay'].values)
#xcv_tfidf_essay=vect.transform(xcv['essay'].values)
essay_tfidf_features=vect.get_feature_names()

print('train_essay bow shape',xtr_tfidf_essay.shape,ytrain.shape)
print('test_essay bow shape',xtest_tfidf_essay.shape,ytest.shape)
#print('cv_essay bow shape',xcv_tfidf_essay.shape,ycv.shape)
```

```
train_essay bow shape (33500, 5000) (33500,)
test_essay bow shape (16500, 5000) (16500,)
```

2)project_title

In [33]:

```
vect=TfidfVectorizer(min_df=10,ngram_range=(1,4),max_features=5000)
vect.fit(xtrain['cleaned_project_title'].values)
#now we have learnt the vocab of xtrain.lets try to transofrm test and cv data using xt
rain vocab
xtr_tfidf_prjtitle=vect.transform(xtrain['cleaned_project_title'].values)
xtest_tfidf_prjtitle=vect.transform(xtest['cleaned_project_title'].values)
#xcv_tfidf_prjtitle=vect.transform(xcv['cleaned_project_title'].values)
prjtitle_tfidf_features=vect.get_feature_names()

print('train_prjtitle bow shape',xtr_tfidf_prjtitle.shape,ytrain.shape)
print('test_prjtitle bow shape',xtest_tfidf_prjtitle.shape,ytest.shape)
#print('cv_prjtitle bow shape',xcv_tfidf_prjtitle.shape,ycv.shape)
```

```
train_prjtitle bow shape (33500, 2427) (33500,)
test_prjtitle bow shape (16500, 2427) (16500,)
```

stack the tfidf features with the other one hot encoded features to get set2

SET2

In [34]:

```
from scipy.sparse import hstack
xtrain_hstack_set2 = hstack((xtr_tfidf_essay,xtr_tfidf_prjtitle,xtrain_ohe_school_state
, xtrain_ohe_teacher_prefix, xtrain_ohe_project_grade_category,xtrain_ohe_clean_categor
ies,xtrain_ohe_clean_subcategories, xtrain_norm_price,xtrain_norm_prjcount)).tocsr()

xtest_hstack_set2 = hstack((xtest_tfidf_essay,xtest_tfidf_prjtitle,xtest_ohe_school_sta
te,xtest_ohe_teacher_prefix,xtest_ohe_project_grade_category,xtest_ohe_clean_categories
,xtest_ohe_clean_subcategories,xtest_norm_price,xtest_norm_prjcount)).tocsr()

#xcv_hstack_set2=hstack((xcv_tfidf_essay,xcv_tfidf_prjtitle,xcv_ohe_school_state,xcv_oh
e_teacher_prefix,xcv_ohe_project_grade_category,xcv_ohe_clean_categories,xcv_ohe_clean_
subcategories,xcv_norm_price,xcv_norm_prjcount)).tocsr()

print("Data Final matrix")
print(xtrain_hstack_set2.shape, ytrain.shape)
print(xtest_hstack_set2.shape, ytest.shape)
#print(xcv_hstack_set2.shape, ycv.shape)
```

```
Data Final matrix
(33500, 7528) (33500,)
(16500, 7528) (16500,)
```

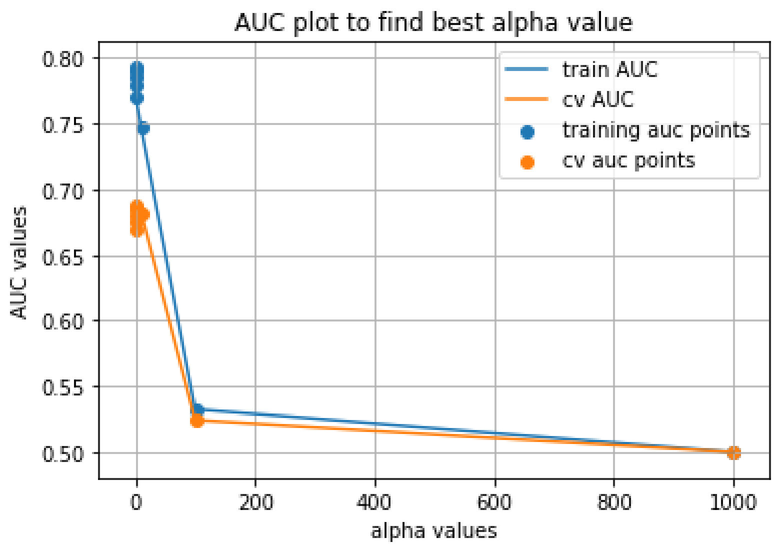
1.5 Appling NB on different kind of featurization as mentioned in the instructions

https://scikit-learn.org/stable/modules/generated/sklearn.naive_baye (https://scikit-learn.org/stable/modules/generated/sklearn.naive_baye

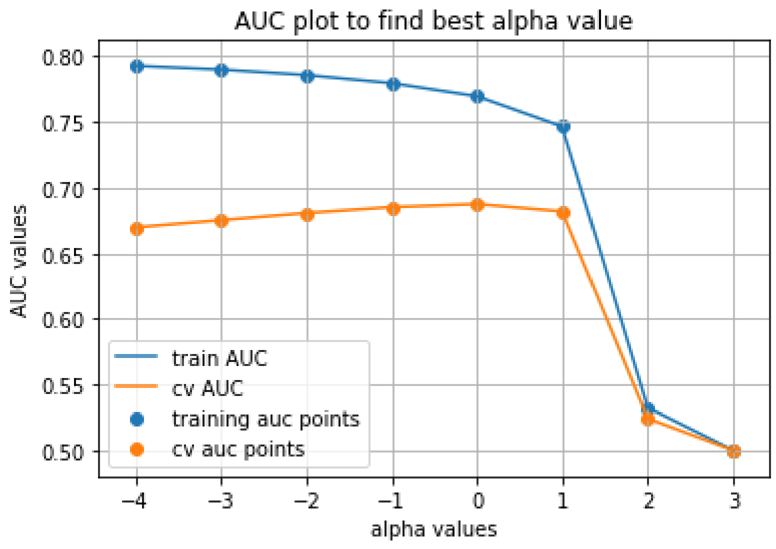
In [35]:

```
import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn import cross_validation
import math
import warnings
warnings.filterwarnings("ignore")


train_auc=[]
cv_auc=[]

#idea

#here write a for loop using logarithms to get powers of 10, also use randint starting
 from megative integers till postitive integers.

clf=MultinomialNB()
params={'alpha':[10**i for i in range(-4,4)]}
RSearch=GridSearchCV(clf,params, cv=3, scoring='roc_auc', return_train_score=True ,n_jo
bs=-1)
RSearch.fit(xtrain_hstack_set1,ytrain)

#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSe
archCV.html (cv_results_)

result=pd.DataFrame.from_dict(RSearch.cv_results_)
result=result.sort_values(['param_alpha'])
train_auc=result['mean_train_score']
train_auc_std=result['std_train_score']
cv_auc=result['mean_test_score']
cv_auc_std=result['std_test_score']
alpha=result['param_alpha']

plt.plot(alpha,train_auc,label='train AUC') #train auc scores
plt.plot(alpha,cv_auc,label='cv AUC') #cv auc scores

plt.scatter(alpha,train_auc,label='training auc points')
plt.scatter(alpha,cv_auc,label='cv auc points')

plt.legend()
plt.xlabel('alpha values')
plt.ylabel('AUC values')
plt.title('AUC plot to find best alpha value')
plt.grid()
plt.show()
print('this is not readable,hence convert alpha values to original values')
#https://stackoverflow.com/questions/49209338/converting-log-values-back-to-number/5098
0215

plt.plot([np.log10(alpha[i]) for i in range(len(alpha))],train_auc,label='train AUC') #
train auc scores
plt.plot([np.log10(alpha[i]) for i in range(len(alpha))],cv_auc,label='cv AUC') #cv auc
scores

plt.scatter([np.log10(alpha[i]) for i in range(len(alpha))],train_auc,label='training a
uc points')
plt.scatter([np.log10(alpha[i]) for i in range(len(alpha))],cv_auc,label='cv auc point
s')

plt.legend()
plt.xlabel('alpha values')
plt.ylabel('AUC values')
plt.title('AUC plot to find best alpha value')
plt.grid()
plt.show()
result.head()
```

```
C:\Users\Sudhi\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41:
DeprecationWarning: This module was deprecated in version 0.18 in favor of
the model_selection module into which all the refactored classes and funct
ions are moved. Also note that the interface of the new CV iterators are d
ifferent from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```



```
this is not readable,hence convert alpha values to original values
```



Out[35]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	parar
0	0.106272	0.009666	0.012326	0.000471	0.0001	{'alph 0.000
1	0.108271	0.010492	0.011659	0.000472	0.001	{'alph 0.001
2	0.107602	0.007931	0.012326	0.000471	0.01	{'alph 0.01}
3	0.096606	0.004644	0.011660	0.000471	0.1	{'alph 0.1}
4	0.098283	0.007534	0.011653	0.000480	1	{'alph 1}

In [36]:

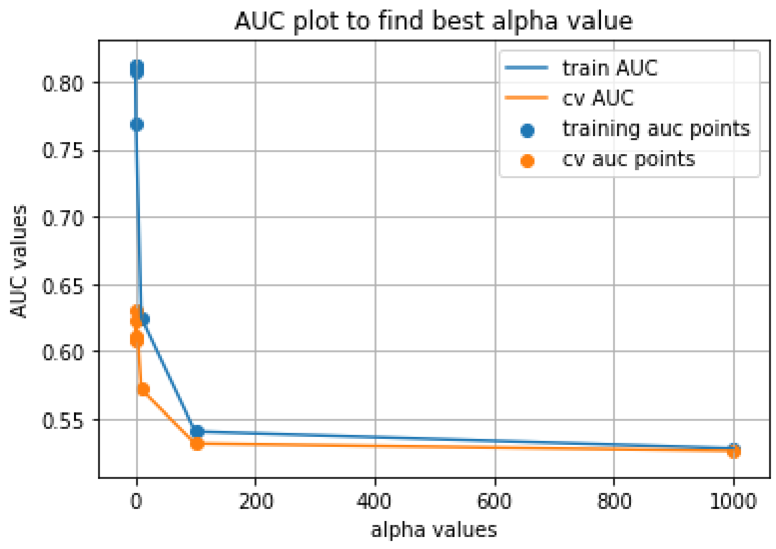
```
best_alpha=RSearch.best_estimator_
print("the best alpha value is",best_alpha.alpha)
```

```
the best alpha value is 1
```

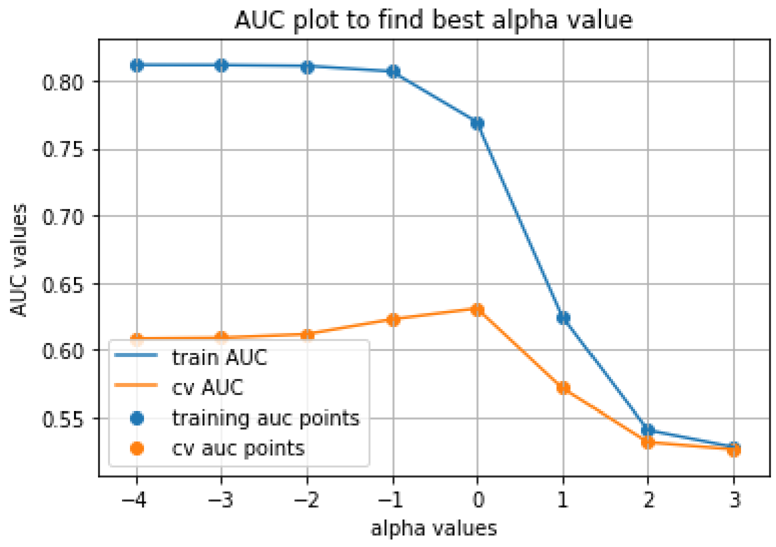
FOR SET2

In [37]:

```
import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn import cross_validation



train_auc=[]
cv_auc=[]

#idea

#here write a for loop using logarithms to get powers of 10, also use randint starting
 from megative integers till postitive integers.

clf=MultinomialNB()
params={'alpha':[10**i for i in range(-4,4)]}
RSearch1=GridSearchCV(clf,params, cv=3, scoring='roc_auc', return_train_score=True ,n_j
obs=-1)
RSearch1.fit(xtrain_hstack_set2,ytrain)

#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSe
archCV.html (cv_results_)

result=pd.DataFrame.from_dict(RSearch1.cv_results_)
result=result.sort_values(['param_alpha'])
train_auc=result['mean_train_score']
train_auc_std=result['std_train_score']
cv_auc=result['mean_test_score']
cv_auc_std=result['std_test_score']
alpha=result['param_alpha']

plt.plot(alpha,train_auc,label='train AUC') #train auc scores
plt.plot(alpha,cv_auc,label='cv AUC') #cv auc scores

plt.scatter(alpha,train_auc,label='training auc points')
plt.scatter(alpha,cv_auc,label='cv auc points')

plt.legend()
plt.xlabel('alpha values')
plt.ylabel('AUC values')
plt.title('AUC plot to find best alpha value')
plt.grid()
plt.show()
print('this is not readable,hence convert alpha values to original values')

plt.plot([np.log10(alpha[i]) for i in range(len(alpha))],train_auc,label='train AUC') #
train auc scores
plt.plot([np.log10(alpha[i]) for i in range(len(alpha))],cv_auc,label='cv AUC') #cv auc
scores

plt.scatter([np.log10(alpha[i]) for i in range(len(alpha))],train_auc,label='training a
uc points')
plt.scatter([np.log10(alpha[i]) for i in range(len(alpha))],cv_auc,label='cv auc point
s')

plt.legend()
plt.xlabel('alpha values')
plt.ylabel('AUC values')
plt.title('AUC plot to find best alpha value')
plt.grid()
plt.show()
result.head()
```



```
this is not readable,hence convert alpha values to original values
```



Out[37]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	para
0	0.103599	0.011716	0.011664	0.000474	0.0001	{'alph 0.000
1	0.102599	0.005312	0.011657	0.000469	0.001	{'alph 0.001
2	0.102264	0.005726	0.013325	0.001247	0.01	{'alph 0.01}
3	0.100948	0.004535	0.011653	0.000466	0.1	{'alph 0.1}
4	0.097277	0.005433	0.012327	0.000472	1	{'alph 1}

In [38]:

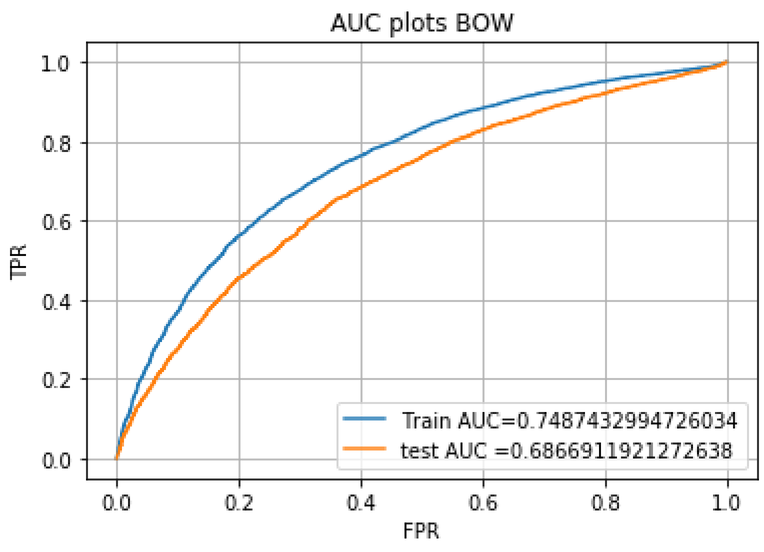
```
best_alpha1=RSearch1.best_estimator_
print("the best alpha value is",best_alpha1.alpha)
```

```
the best alpha value is 1
```

NOW PREDICT ON THE TEST DATA (SET1)(BOW)

In [39]:

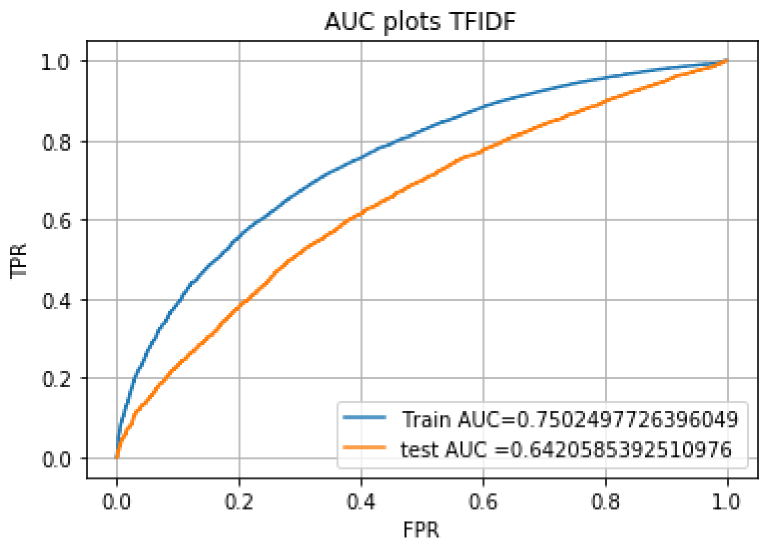
```
from sklearn.metrics import roc_curve, auc
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
 of the positive class
    # not the predicted outputs

    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


clf1=MultinomialNB(alpha=best_alpha.alpha)
clf1.fit(xtrain_hstack_set1,ytrain)

y_train_pred=batch_predict(clf1,xtrain_hstack_set1)
y_test_pred=batch_predict(clf1,xtest_hstack_set1)

train_fpr,train_tpr,tr_thresholds=roc_curve(ytrain,y_train_pred)
test_fpr,test_tpr,te_thresholds=roc_curve(ytest,y_test_pred)
auc1=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('AUC plots BOW')
plt.grid()
plt.show()
```



NOW PREDICT ON THE TEST DATA (SET2)(TFIDF)

In [41]:

```
from sklearn.metrics import roc_curve, auc
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
 of the positive 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


clf=MultinomialNB(alpha=best_alpha1.alpha)
clf.fit(xtrain_hstack_set2,ytrain)

y_train_pred=batch_predict(clf,xtrain_hstack_set2)
y_test_pred=batch_predict(clf,xtest_hstack_set2)

train_fpr,train_tpr,tr_thresholds=roc_curve(ytrain,y_train_pred)
test_fpr1,test_tpr1,te_thresholds=roc_curve(ytest,y_test_pred)
auc2=auc(test_fpr1,test_tpr1)
plt.plot(train_fpr,train_tpr,label='Train AUC='+str(auc(train_fpr,train_tpr)))
plt.plot(test_fpr1,test_tpr1,label='test AUC ='+str(auc(test_fpr1,test_tpr1)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel('TPR')
plt.title('AUC plots TFIDF')
plt.grid()
plt.show()
```



TOP 20 FEATURES

In [43]:

```
import numpy as np
feature_list=[]
#names=[]
#feature_list=list(feature_list)
feature_list.extend(essay_bow_features)
feature_list.extend(prjtitle_bow_features)
feature_list.extend(school_state_feature)
feature_list.extend(teacher_prefix_feature)
feature_list.extend(project_grade_category_feature)
feature_list.extend(clean_categories_feature)
feature_list.extend(clean_subcategories_feature)
feature_list.extend(['price'])
feature_list.extend(['teacher_number_of_previously_posted_projects'])

print(len(feature_list))

#print(feature_list)

top_20_index0=np.argsort(((clf1.feature_log_prob_)[0])[::-1])[:20]
top_20_index1=np.argsort(((clf1.feature_log_prob_)[1])[::-1])[:20]
names=np.take(feature_list,top_20_index)
names1=np.take(feature_list,top_20_index1)
print('top 20 feature indices for class 0')
print(top_20_index)
print('-'*100)
print(' top 20 features for class 0')
print('-'*100)
print(names)


print('-'*100)
print('top 20 feature indices for class 1')
print('-'*100)
print(top_20_index1)
print('-'*100)
print(' top 20 features for class 1')
print('-'*100)
print(names1)


#for i in top_20_index:
    #print(feature_list[i])

#feature_list[7]
```

```
7528
top 20 feature indices for class 0
[1913 2290 1603 1605  843 1468  200 1219 2163 1409  622 1915  489 1306
 1608  492 1691 1637 1692  863]
--------------------------------------------------------------------------
--------------------------
 top 20 features for class 0
--------------------------------------------------------------------------
--------------------------
['helped' 'learn the' 'forever' 'form' 'community we' 'extend' 'algebra'
 'early childhood' 'items' 'examples' 'chance' 'helping' 'book read'
 'engaged' 'forms' 'books classroom' 'gains' 'freedom' 'game'
 'comprehension skills']
--------------------------------------------------------------------------
--------------------------
top 20 feature indices for class 1
--------------------------------------------------------------------------
--------------------------
[   2   39   30   32   49 1960  147 2430 1291  744 1790 1545 1138 1020
  177  559 1861 1170  272  116]
--------------------------------------------------------------------------
--------------------------
 top 20 features for class 1
--------------------------------------------------------------------------
--------------------------
['100' '2nd graders' '22' '24' '4th 5th' 'hope' 'across'
 'literacy centers' 'encouragement' 'clean' 'great' 'fingertips'
 'district students faced several' 'day our' 'adult' 'called' 'health'
 'donors choose' 'anchor charts' 'absolutely love']
```

confusion matrix

In [44]:

```
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
nd(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

print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(ytrain, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(ytest, predict_with_best_t(y_test_pred, best_t)))
```

```
===========================================================================
=========================
the maximum value of tpr*(1-fpr) 0.4706915581925847 for threshold 0.85
Train confusion matrix
[[ 3657  1708]
 [ 8707 19428]]
Test confusion matrix
[[1418 1224]
 [4582 9276]]
```

3. Summary

as mentioned in the step 5 of instructions

In [45]:

```
# http://zetcode.com/python/prettytable/
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.auc.html
from prettytable import PrettyTable
pt=PrettyTable()
pt.field_names = ["Vectorizer", "Model", "Hyper-Parameter", "AUC"]
pt.add_row([" BOW ", "Naive Bayes", best_alpha.alpha,auc1])
pt.add_row([" TFIDF ", "Naive Bayes", best_alpha1.alpha, auc2])
print(pt)
```

```
+------------+-------------+-----------------+--------------------+
| Vectorizer |    Model    | Hyper-Parameter |        AUC         |
+------------+-------------+-----------------+--------------------+
|    BOW     | Naive Bayes |        1        | 0.6866911921272638 |
|   TFIDF    | Naive Bayes |        1        | 0.6420585392510976 |
+------------+-------------+-----------------+--------------------+
```