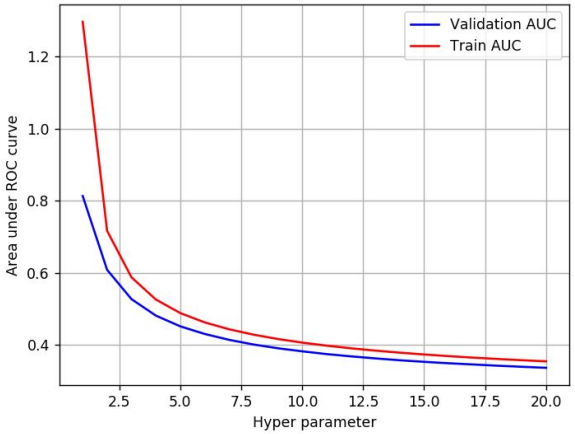
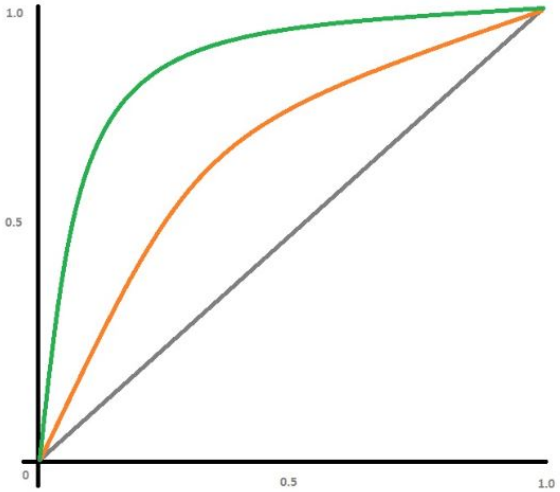
Assignment 6: Apply NB

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/)](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
TFIDF	Brute	12	0.79
W2V	Brute	10	0.78
TFIDFW2V	Brute	6	0.78

2. Naive Bayes

1.1 Loading Data

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
# Tutorial about Python regular expressions: https://pymotw.com/2/re/

import pickle
from tqdm import tqdm
import os

from chart_studio import plotly # use chart_studio instead of plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

In [3]:
```
import pandas
data = pandas.read_csv('preprocessed_data.csv',nrows=50000)
```

In [4]:
```
data.head(1)
```

Out[4]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_categ
0	ca	mrs	grades_prek_2	53	1	math_sc

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

In [5]:
```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about 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
```

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

Out[6]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcateg
0	ca	mrs	grades_prek_2	53	math_science	appliedscie health_lifesc

In [ ]:

In [7]:
```
# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

1.3 Make Data Model Ready: encoding eassay, and project_title

In [8]:
```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# make sure you featurize train and test data separatly

# 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
```

Bow representation of 'essay' features

In [9]:
```
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)


# I have choosen max_features = 10000 because of having low configuration system
# you can increase the no.of max_features or simply remove max_features and use all the features
essay_vectorizer_bow = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=10000)
essay_vectorizer_bow.fit(X_train['essay'].values) # fit has to happen only on train data

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

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

```
(22445, 8) (22445,)
(11055, 8) (11055,)
(16500, 8) (16500,)
====================================================================================================
After vectorizations
(22445, 10000) (22445,)
(11055, 10000) (11055,)
(16500, 10000) (16500,)
====================================================================================================
```

tfidf representation of essay feature

In [10]:
```
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)

# I have choosen max_features = 10000 because of having low configuration system
# you can increase the no.of max_features or simply remove max_features and use all the features
essay_vectorizer_tfidf = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=10000)
essay_vectorizer_tfidf.fit(X_train['essay'].values) # fit has to happen only on train data

# we use the fitted TfidfVectorizer to convert the text to vector
X_train_essay_tfidf = essay_vectorizer_tfidf.transform(X_train['essay'].values)
X_cv_essay_tfidf = essay_vectorizer_tfidf.transform(X_cv['essay'].values)
X_test_essay_tfidf = essay_vectorizer_tfidf.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
```

```
(22445, 8) (22445,)
(11055, 8) (11055,)
(16500, 8) (16500,)
====================================================================================================
After vectorizations
(22445, 10000) (22445,)
(11055, 10000) (11055,)
(16500, 10000) (16500,)
====================================================================================================
```

In [ ]:

1.4 Make Data Model Ready: encoding numerical, categorical features

In [11]:
```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# make sure you featurize train and test data separatly

# 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
```

In [12]:
```
data.info() # basic information of dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 9 columns):
school_state                                    50000 non-null object
teacher_prefix                                  50000 non-null object
project_grade_category                          50000 non-null object
teacher_number_of_previously_posted_projects    50000 non-null int64
project_is_approved                             50000 non-null int64
clean_categories                                50000 non-null object
clean_subcategories                             50000 non-null object
essay                                           50000 non-null object
price                                           50000 non-null float64
dtypes: float64(1), int64(2), object(6)
memory usage: 3.4+ MB
```

1.4.1 encoding categorical features: school_state

In [13]:
```
school_vectorizer = CountVectorizer()
school_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 = school_vectorizer.transform(X_train['school_state'].values)
X_cv_state_ohe = school_vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = school_vectorizer.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(school_vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 51) (22445,)
(11055, 51) (11055,)
(16500, 51) (16500,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', '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']
====================================================================================================
```

1.4.2 encoding categorical features: teacher_prefix

In [14]:
```
teacher_vectorizer = CountVectorizer()
teacher_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 = teacher_vectorizer.transform(X_train['teacher_prefix'].values)
X_cv_teacher_ohe = teacher_vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = teacher_vectorizer.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_cv_teacher_ohe.shape, y_cv.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(teacher_vectorizer.get_feature_names())
print("="*100)
```

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

1.4.3 encoding categorical features: project_grade_category

In [15]:
```
pgrade_vectorizer = CountVectorizer()
pgrade_vectorizer.fit(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_train_grade_ohe = pgrade_vectorizer.transform(X_train['project_grade_category'].values)
X_cv_grade_ohe = pgrade_vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = pgrade_vectorizer.transform(X_test['project_grade_category'].values)

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(pgrade_vectorizer.get_feature_names())
print("="*100)
```

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

1.4.4 encoding categorical features: clean_categories

In [16]:
```
category_vectorizer = CountVectorizer()
category_vectorizer.fit(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_train_category_ohe = category_vectorizer.transform(X_train['clean_categories'].values)
X_cv_category_ohe = category_vectorizer.transform(X_cv['clean_categories'].values)
X_test_category_ohe = category_vectorizer.transform(X_test['clean_categories'].values)

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

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

1.4.5 encoding categorical features: clean_subcategories

In [17]:
```
subcat_vectorizer = CountVectorizer()
subcat_vectorizer.fit(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_train_subcategory_ohe = subcat_vectorizer.transform(X_train['clean_subcategories'].values)
X_cv_subcategory_ohe = subcat_vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_subcategory_ohe = subcat_vectorizer.transform(X_test['clean_subcategories'].values)

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

```
After vectorizations
(22445, 30) (22445,)
(11055, 30) (11055,)
(16500, 30) (16500,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep', 'communit
yservice', 'earlydevelopment', 'economics', 'environmentalscience', 'esl', 'extracurricular', 'financialliter
acy', 'foreignlanguages', 'gym_fitness', 'health_lifescience', 'health_wellness', 'history_geography', 'liter
acy', 'literature_writing', 'mathematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'perf
ormingarts', 'socialsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
====================================================================================================
```

standardizing numerical value

1.5.1 encoding numerical features: Price

In [18]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 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 single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['price'].values.reshape(1,-1))

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



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

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

In [ ]:

1.5.2 encoding numerical features: teacher_number_of_previously_posted_projects

In [19]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values)
# this will rise an error Expected 2D array, got 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 single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))

X_train_nproject_norm = (normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.
reshape(1,-1))).reshape(-1,1)
X_cv_nproject_norm = (normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshap
e(1,-1))).reshape(-1,1)
X_test_nproject_norm = (normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.re
shape(1,-1))).reshape(-1,1)



print("After vectorizations")
print(X_train_nproject_norm.shape, y_train.shape)
print(X_cv_nproject_norm.shape, y_cv.shape)
print(X_test_nproject_norm.shape, y_test.shape)
print("="*100)
```

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

In [ ]:

Create set1 and set2

In [20]:
```
# Create set1
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_s1 = hstack((X_train_essay_bow, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_price
_norm,X_train_category_ohe,X_train_nproject_norm,X_train_subcategory_ohe)).tocsr()
X_cr_s1 = hstack((X_cv_essay_bow, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_price_norm,X_cv_cate
gory_ohe,X_cv_subcategory_ohe,X_cv_nproject_norm)).tocsr()
X_te_s1 = hstack((X_test_essay_bow, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_price_norm
,X_test_category_ohe,X_test_subcategory_ohe,X_test_nproject_norm)).tocsr()

print("Final Data matrix")
print(X_tr_s1.shape, y_train.shape)
print(X_cr_s1.shape, y_cv.shape)
print(X_te_s1.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 10101) (22445,)
(11055, 10101) (11055,)
(16500, 10101) (16500,)
====================================================================================================
```

create set2

In [21]:
```
# Create set2
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_s2 = hstack((X_train_essay_tfidf, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_pri
ce_norm,X_train_category_ohe,X_train_subcategory_ohe,X_train_nproject_norm)).tocsr()
X_cr_s2 = hstack((X_cv_essay_tfidf, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_price_norm,X_cv_ca
tegory_ohe,X_cv_subcategory_ohe,X_cv_nproject_norm)).tocsr()
X_te_s2 = hstack((X_test_essay_tfidf, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_price_no
rm,X_test_category_ohe,X_test_subcategory_ohe,X_test_nproject_norm)).tocsr()

print("Final Data matrix")
print(X_tr_s2.shape, y_train.shape)
print(X_cr_s2.shape, y_cv.shape)
print(X_te_s2.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 10101) (22445,)
(11055, 10101) (11055,)
(16500, 10101) (16500,)
====================================================================================================
```

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

In [22]:
```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about 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
```

implement MultinomialNB on set1

In [23]:
```
# import MultinomialNB from sklearn.naive_bayes
from sklearn.naive_bayes import MultinomialNB
```

In [24]:
```
model_bow = MultinomialNB() # creating the model of MultinomialNB with default parameter
```

In [25]:
```
model_bow.fit(X_tr_s1,y_train) # let's fit the model
```

Out[25]:
```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

In [26]:
```
predictions_bow = model_bow.predict(X_te_s1) # predicton on test data
```

In [27]:
```
# import metrics
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
```

In [28]:
```
# confusion_matrix
confusion_matrix(predictions_bow,y_test)
```

Out[28]:
```
array([[ 1351,  3303],
       [ 1291, 10555]], dtype=int64)
```

In [29]:
```
# accuracy_score
print('accuracy_score of set1:',accuracy_score(predictions_bow,y_test))
```

```
accuracy_score of set1: 0.7215757575757575
```

In [30]:
```
# classification report
print(classification_report(predictions_bow, y_test))
```

```
              precision    recall  f1-score   support

           0       0.51      0.29      0.37      4654
           1       0.76      0.89      0.82     11846

    accuracy                           0.72     16500
   macro avg       0.64      0.59      0.60     16500
weighted avg       0.69      0.72      0.69     16500
```

In [31]:
```
from sklearn.metrics import roc_auc_score # import roc_auc_score
```

In [32]:
```
# find roc_auc_score of test data
print('roc_auc_score:',roc_auc_score(y_test,model_bow.predict_proba(X_te_s1)[:,1]))
```

```
roc_auc_score: 0.6968027824995584
```

implement MultinomialNB on set2 data

In [33]:
```
model_tfidf = MultinomialNB() # creating model of MultinomialNB by default parameter
```

In [34]:
```
model_tfidf.fit(X_tr_s2,y_train) # fitting the model
```

Out[34]:
```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

In [35]:
```
predictions_tfidf = model_tfidf.predict(X_te_s2) # find predicted value by trained model
```

In [36]:
```
confusion_matrix(predictions_tfidf,y_test) # confusion matrix of set2(used tfidf)
```

Out[36]:
```
array([[   46,   139],
       [ 2596, 13719]], dtype=int64)
```

In [37]:
```
print('accuracy of model (using tfidf:)')
accuracy_score(predictions_tfidf,y_test) # accuracy of
```

```
accuracy of model (using tfidf:)
```

Out[37]:
```
0.8342424242424242
```

In [38]:
```
print(classification_report(predictions_tfidf,y_test))
```

```
              precision    recall  f1-score   support

           0       0.02      0.25      0.03       185
           1       0.99      0.84      0.91     16315

    accuracy                           0.83     16500
   macro avg       0.50      0.54      0.47     16500
weighted avg       0.98      0.83      0.90     16500
```

In [39]:
```
# roc_auc_score of test dataset of set2
roc_auc_score(y_test,model_tfidf.predict_proba(X_te_s2)[:,1])
```

Out[39]:
```
0.6609722338908682
```

2.2 Hyperparameter tuning(find best alpha value)

set1: BOW represenatation

In [40]:
```
from sklearn.metrics import roc_auc_score,roc_curve,auc
```

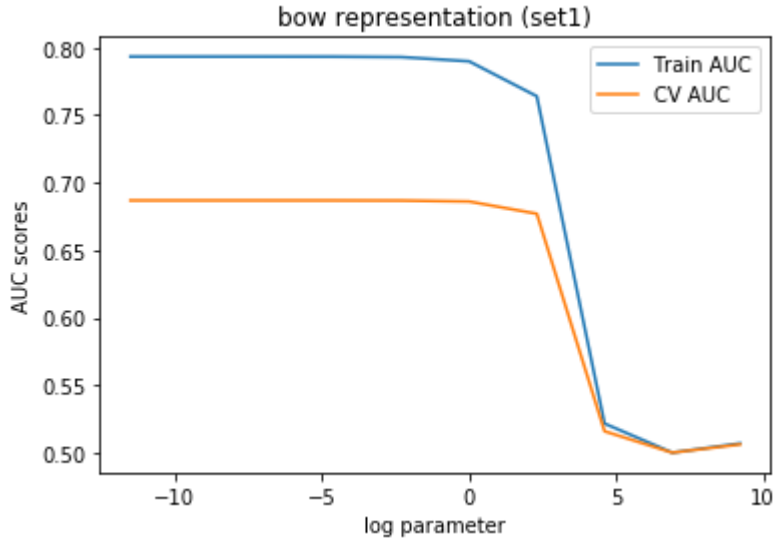
In [41]:
```
#https://datascience.stackexchange.com/questions/22762/understanding-predict-proba-from-multioutputclassifier
auc_scores_tr = [] # to store auc_score
auc_scores_cv = []

alphas = [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
for i in alphas:
    model = MultinomialNB(alpha=i,class_prior=[0.5,0.5])
    model.fit(X_tr_s1,y_train)
    y_train_pred = model.predict_proba(X_tr_s1)[:,1]
    y_cv_pred = model.predict_proba(X_cr_s1)[:,1]



    auc_tr = roc_auc_score(y_train, y_train_pred)
    auc_cv = roc_auc_score(y_cv, y_cv_pred)

    auc_scores_tr.append(auc_tr)
    auc_scores_cv.append(auc_cv)
```

In [42]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [43]:
```
import math
```

In [44]:
```
log_alpha = [math.log(i) for i in alphas]
```

In [45]:
```
plt.title('bow representation (set1)')
plt.plot(log_alpha, auc_scores_tr,label='Train AUC')
plt.plot(log_alpha, auc_scores_cv,label='CV AUC')
plt.xlabel('log parameter')
plt.ylabel('AUC scores')
plt.legend()
plt.show()
```

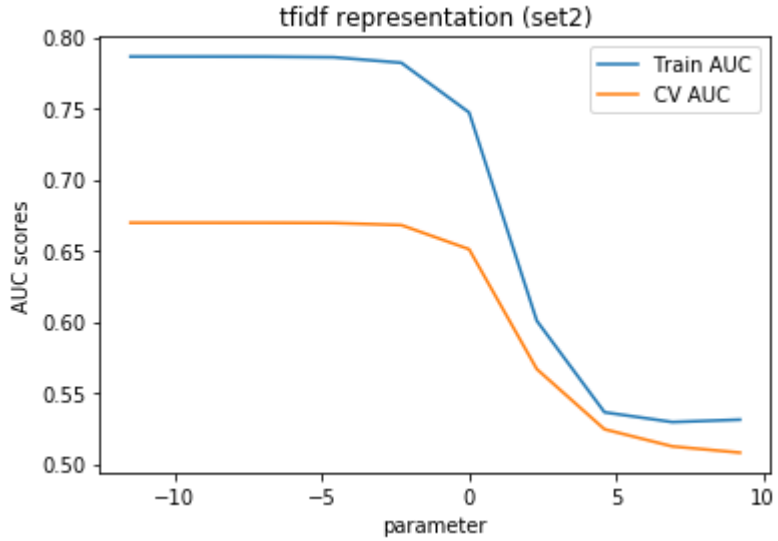


In [ ]:
```
```

set2: TFIDF representation

In [46]:
```
#https://datascience.stackexchange.com/questions/22762/understanding-predict-proba-from-multioutputclassifier
auc_scores_tr = [] # to store auc_score
auc_scores_cv = []
for i in alphas: # for each value in range of 0 to 20
    model = MultinomialNB(alpha=i,class_prior=[0.5,0.5])
    model.fit(X_tr_s2,y_train)
    y_train_pred = model.predict_proba(X_tr_s2)[:,1]
    y_cv_pred = model.predict_proba(X_cr_s2)[:,1]


   # finding auc score of both train and CV dataset
    auc_tr = roc_auc_score(y_train, y_train_pred)
    auc_cv = roc_auc_score(y_cv, y_cv_pred)

#     print(auc_tr)
    auc_scores_tr.append(auc_tr)
    auc_scores_cv.append(auc_cv)
```

In [47]:
```
plt.title('tfidf representation (set2)')
plt.plot(log_alpha, auc_scores_tr,label='Train AUC')
plt.plot(log_alpha, auc_scores_cv,label='CV AUC')
plt.xlabel('parameter')
plt.ylabel('AUC scores')
plt.legend()
plt.show()
```



Observation: Best alpha value is: 0.00001 using bow

In [48]:
```
model_bow = MultinomialNB(alpha=0.00001,class_prior=[0.5,0.5]) # creating model of MultinomialNB by alpha=0
model_tfidf = MultinomialNB(alpha=0.00001,class_prior=[0.5,0.5]) # creating model of MultinomialNB by alpha=0

model_bow.fit(X_tr_s1,y_train) # fitting the model
model_tfidf.fit(X_tr_s2,y_train)

# find prediction of every model
y_test_bow = model_bow.predict_proba(X_te_s1)
y_test_tfidf = model_tfidf.predict_proba(X_te_s2)


# find auc score
auc_test_bow = roc_auc_score(y_test, y_test_bow[:,1])
auc_test_tfidf = roc_auc_score(y_test, y_test_tfidf[:,1])
```

In [49]:
```
print('auc of test using bow',auc_test_bow)
print('auc of test using tfidf',auc_test_tfidf)
```

```
auc of test using bow 0.6974910657016572
auc of test using tfidf 0.6790381384277361
```

In [50]:
```
np.array(auc_scores_tr)
```

Out[50]:
```
array([0.78639403, 0.78639026, 0.78635008, 0.78595097, 0.78204515,
       0.74712518, 0.60062716, 0.53630713, 0.52944396, 0.53111407])
```

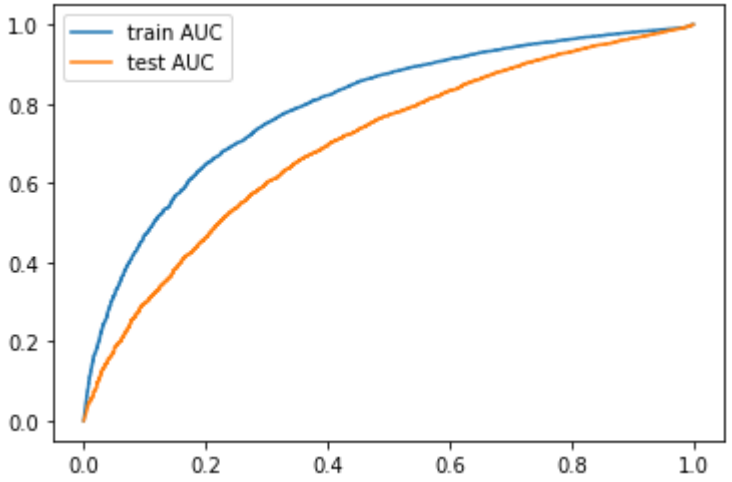
In [ ]:
```

```

plot AUC of train and test dataset of set1(used Bow)

In [51]:
```
y_train_pred = model_bow.predict_proba(X_tr_s1)[:,1]
y_test_pred = model_bow.predict_proba(X_te_s1)[:,1]
```

In [52]:
```
fpr_train,tpr_train, thresholds = metrics.roc_curve(y_train,y_train_pred)
fpr_test,tpr_test, thresholds = metrics.roc_curve(y_test,y_test_pred)

plt.plot(fpr_train,tpr_train,label='train AUC')
plt.plot(fpr_test,tpr_test,label='test AUC')
plt.legend()
```

Out[52]:
```
<matplotlib.legend.Legend at 0x19fce11efc8>
```



confusion matrix of test dataset of set1

In [53]:
```
confusion_matrix(y_test,model_bow.predict(X_te_s1))
```

Out[53]:
```
array([[1569, 1073],
       [4137, 9721]], dtype=int64)
```

accuracy of test dataset of set1

In [54]:
```
accuracy_score(y_test,model_bow.predict(X_te_s1))
```

Out[54]:
```
0.6842424242424242
```

auc score of test dataset of set1

In [55]:
```
print('auc of test using bow',auc_test_bow)
```

```
auc of test using bow 0.6974910657016572
```

In [ ]:
```

```

In [56]:
```
# Intentionally left blank
```

4. find top 20 features

In [57]:
```
#https://datascience.stackexchange.com/questions/65219/find-the-top-n-features-from-feature-set-using-absolute-values-of-feature-log-p/65232#65232?newreg=b328560c7a3b415a932c90d8e0f80182
# initialize alpha=0(found optimal)
modelNB = MultinomialNB(alpha=0.00001)
modelNB.fit(X_tr_s1, y_train)
```

Out[57]:
```
MultinomialNB(alpha=1e-05, class_prior=None, fit_prior=True)
```

In [58]:
```
#.argsort() returns the indices of features sorted accoding to feature_log_prob_

# for positive class(1)
class_1_index = modelNB.feature_log_prob_[1,:].argsort()
# for negative class(0)
class_0_index = modelNB.feature_log_prob_[0,:].argsort()
```

In [59]:
```
# calculating features_list
features_list_bow = list(essay_vectorizer_bow.get_feature_names()+ school_vectorizer.get_feature_names()+
                     teacher_vectorizer.get_feature_names()+ pgrade_vectorizer.get_feature_names()+
                     ['price']+category_vectorizer.get_feature_names()+
                     ['teacher_number_of_previously_posted_projects']+subcat_vectorizer.get_feature_names())
```

In [60]:
```
len(features_list_bow) # total no.of features
```

Out[60]:
```
10101
```

In [61]:
```
features_1 = [] # to store important features of class1
features_0 = [] # to store important features of class0

for index in class_1_index[-20:-1]:
    features_1.append(features_list_bow[index])

for index in class_0_index[-20:-1]:
    features_0.append(features_list_bow[index])
```

In [62]:
```
print('20 most important features of class 1(positive class):')
print(features_1)
print('='*50)
print('20 most important features of class 0(negative class):')
print(features_0)
```

```
20 most important features of class 1(positive class):
['able', 'day', 'use', 'need', 'we', 'work', 'reading', 'nannan', 'many', 'help', 'my students', 'they', 'lea
rn', 'not', 'the', 'classroom', 'learning', 'my', 'school']
==================================================
20 most important features of class 0(negative class):
['reading', 'able', 'year', 'come', 'work', 'need', 'we', 'nannan', 'many', 'my students', 'the', 'help', 'th
ey', 'learn', 'not', 'classroom', 'my', 'learning', 'school']
```

In [ ]:
```

```

3. Summary

Summary in Table Format

In [63]:
```
# http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

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


x.add_row(['BOW','MultinomialNB',0.00001,auc_test_bow])
x.add_row(['TFIDF','MultinomialNB',0.00001,auc_test_tfidf])
```

In [64]:
```
print(x)
```

```
+------------+---------------+-----------------------+--------------------+
| Vectorizer |     Model     | Hyperparameter(alpha) |        AUC         |
+------------+---------------+-----------------------+--------------------+
|    BOW     | MultinomialNB |         1e-05         | 0.6974910657016572 |
|   TFIDF    | MultinomialNB |         1e-05         | 0.6790381384277361 |
+------------+---------------+-----------------------+--------------------+
```