Apply Multinomial NB

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 nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

from chart_studio.plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

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

In [2]:

```
project_data = pd.read_csv("train_data.csv", nrows = 75000)
```

In [3]:

```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
```

```
Number of data points in train data (75000, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [4]:

```
# Let's check for "missing" or "NaN" values in our dataset
project_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 75000 entries, 0 to 74999
Data columns (total 17 columns):
Unnamed: 0                                      75000 non-null int64
id                                              75000 non-null object
teacher_id                                      75000 non-null object
teacher_prefix                                  74997 non-null object
school_state                                    75000 non-null object
project_submitted_datetime                      75000 non-null object
project_grade_category                          75000 non-null object
project_subject_categories                      75000 non-null object
project_subject_subcategories                   75000 non-null object
project_title                                   75000 non-null object
project_essay_1                                 75000 non-null object
project_essay_2                                 75000 non-null object
project_essay_3                                 2558 non-null object
project_essay_4                                 2558 non-null object
project_resource_summary                        75000 non-null object
teacher_number_of_previously_posted_projects    75000 non-null int64
project_is_approved                             75000 non-null int64
dtypes: int64(3), object(14)
memory usage: 9.7+ MB
```

It seems to be a missing values present for "teacher_prefix" feature. We'll replace the missing values with the mode of "project_prefix" columns itself.

In [5]:

```
project_data['teacher_prefix'].isna().sum()
```

Out[5]:

```
3
```

In [6]:

```
# Let's replace the "missing" values by the most repeated value from teacher_prefix i.e., mode of
the column "teacher_prefix"
project_data['teacher_prefix'].mode()
```

Out[6]:

```
0    Mrs.
dtype: object
```

In [7]:

```
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna('Mrs.')
```

In [8]:

```
# Let's select only the selected features or columns
#
project_data.drop(['id','teacher_id','project_submitted_datetime','project_resource_summary'],axis
=1, inplace=True)
project_data.columns
```

Out[8]:

```
Index(['Unnamed: 0', 'teacher_prefix', 'school_state',
       'project_grade_category', 'project_subject_categories',
       'project_subject_subcategories', 'project_title', 'project_essay_1',
       'project_essay_2', 'project_essay_3', 'project_essay_4',
       'teacher_number_of_previously_posted_projects', 'project_is_approved'],
      dtype='object')
```

In [9]:

```
# Data seems to be highly imbalanced since the ratio of "class 1" to "class 0" is nearly 5.5
project_data['project_is_approved'].value_counts()
```

Out[9]:

```
1    63632
0    11368
Name: project_is_approved, dtype: int64
```

In [10]:

```
number_of_approved = project_data['project_is_approved'][project_data['project_is_approved'] == 1].
count()
number_of_not_approved = project_data['project_is_approved'][project_data['project_is_approved'] =
= 0].count()

print("Ratio of Project approved to Not approved is:", number_of_approved/number_of_not_approved)
```

```
Ratio of Project approved to Not approved is: 5.597466572836031
```

Let's first merge all the project_essays into single columns

In [11]:

```
# merge two column text dataframe:
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(str) + \
                        project_data["project_essay_4"].map(str)
```

In [12]:

```
project_data.head(5)
```

Out[12]:

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcatego
0	160221	Mrs.	IN	Grades PreK-2	Literacy & Language	ESL, Literacy
1	140945	Mr.	FL	Grades 6-8	History & Civics, Health & Sports	Civics & Government, Team Sports
2	21895	Ms.	AZ	Grades 6-8	Health & Sports	Health & Wellness, Team Sp
3	45	Mrs.	KY	Grades PreK-2	Literacy & Language, Math & Science	Literacy, Mathematics

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcatego
4	172407	Mrs.	TX	Grades PreK-2	Math & Science	Mathematics

In [13]:

```
# Let's drop the project essay columns from the dadaset now, as we have captured the essay text da
ta into single "essay" column
project_data.drop(['project_essay_1','project_essay_2','project_essay_3','project_essay_4'],axis=1
, inplace=True)
```

In [14]:

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

Out[14]:

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcatego
0	160221	Mrs.	IN	Grades PreK-2	Literacy & Language	ESL, Literacy

In [15]:

```
# 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)
```

2) Make Data Model Ready: encoding numerical, categorical features

In [16]:

```
def cleaning_text_data(list_text_feature,df,old_col_name,new_col_name):

    # remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039
    # https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
    # https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
    # https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
    feature_list = []
    for i in list_text_feature:
        temp = ""
        # consider we have text like this "Math & Science, Warmth, Care & Hunger"
        for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care
& Hunger"]
            if 'The' in j.split(): # this will split each of the catogory based on space "Math & Sc
ience"=> "Math","&", "Science"
                j=j.replace('The','') # if we have the words "The" we are going to replace it with
''(i.e removing 'The')
            j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Sc
ience"=>"Math&Science"
            temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
            temp = temp.replace('&','_') # we are replacing the & value into
        temp = temp.replace('&','_') # we are replacing the & value into
        feature_list.append(temp.strip())

    df[new_col_name] = feature_list
    df.drop([old_col_name], axis=1, inplace=True)

    from collections import Counter
    my_counter = Counter()
    for word in df[new_col_name].values:
        my_counter.update(word.split())

    feature_dict = dict(my_counter)
    sorted_feature_dict = dict(sorted(feature_dict.items(), key=lambda kv: kv[1]))
    return sorted_feature_dict
```

In [17]:

```
def clean_project_grade(list_text_feature,df,old_col_name,new_col_name):

    # remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039
    # https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
    # https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
    # https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
    feature_list = []
    for i in list_text_feature:
        temp = i.split(' ')
        last_dig = temp[-1].split('-')
        fin = [temp[0]]
        fin.extend(last_dig)
        feature = '_'.join(fin)
        feature_list.append(feature.strip())

    df[new_col_name] = feature_list
    df.drop([old_col_name], axis=1, inplace=True)

    from collections import Counter
    my_counter = Counter()
    for word in df[new_col_name].values:
        my_counter.update(word.split())

    feature_dict = dict(my_counter)
    sorted_feature_dict = dict(sorted(feature_dict.items(), key=lambda kv: kv[1]))
    return sorted_feature_dict
```

2.1) Text Preprocessing: project_subject_categories

In [18]:

```
x_train_sorted_category_dict = cleaning_text_data(X_train['project_subject_categories'],X_train,'p
roject_subject_categories','clean_categories')
x_test_sorted_category_dict =
cleaning_text_data(X_test['project_subject_categories'],X_test,'project_subject_categories','clean_
categories')
x_cv_sorted_category_dict =
cleaning_text_data(X_cv['project_subject_categories'],X_cv,'project_subject_categories','clean_cate
gories')
```

2.2) Text Preprocessing : project_subject_subcategories

In [19]:

```
x_train_sorted_subcategories = cleaning_text_data(X_train['project_subject_subcategories'],X_train
,'project_subject_subcategories','clean_subcategories')
x_test_sorted_subcategories = cleaning_text_data(X_test['project_subject_subcategories'],X_test,'p
roject_subject_subcategories','clean_subcategories')
x_cv_sorted_subcategories =
cleaning_text_data(X_cv['project_subject_subcategories'],X_cv,'project_subject_subcategories','cle
an_subcategories')
```

2.4) Text Preprocessing: project_grade_category

In [20]:

```
x_train_sorted_grade =
clean_project_grade(X_train['project_grade_category'],X_train,'project_grade_category','clean_grade
')
x_test_sorted_grade =
clean_project_grade(X_test['project_grade_category'],X_test,'project_grade_category','clean_grade'
)
x_cv_sorted_grade =
clean_project_grade(X_cv['project_grade_category'],X_cv,'project_grade_category','clean_grade')
```

2.5) Text Preprocessing (stowords): project_essay, project_title

In [21]:

```
# 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
```

In [22]:

```
# 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're", "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', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', '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', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [23]:

```
# Combining all the above stundents
from tqdm import tqdm
def process_text(df,col_name):
    preprocessed_feature = []
    # tqdm is for printing the status bar
    for sentance in tqdm(df[col_name].values):
        sent = decontracted(sentance)
        sent = sent.replace('\\r', ' ')
        sent = sent.replace('\\"', ' ')
        sent = sent.replace('\\n', ' ')
        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_feature.append(sent.lower().strip())
    return preprocessed_feature
```

In [24]:

```
x_train_essay_preprocessed = process_text(X_train,'essay')
x_test_essay_preprocessed = process_text(X_test,'essay')
x_cv_essay_preprocessed = process_text(X_cv,'essay')
```

```
100%|██████████████████████████████████████████████████████████████████████████| 33667/33667
[00:32<00:00, 1035.58it/s]
100%|██████████████████████████████████████████████████████████████████████████| 24750/24750
[00:23<00:00, 1048.01it/s]
100%|██████████████████████████████████████████████████████████████████████████| 16583/16583
[00:15<00:00, 1050.76it/s]
```

In [25]:

```
x_train_title_preprocessed = process_text(X_train,'project_title')
x_test_title_preprocessed = process_text(X_test,'project_title')
x_cv_title_preprocessed = process_text(X_cv,'project_title')
```

```
100%|█████████████████████████████████████████████████████████████████████████| 33667/33667
[00:01<00:00, 19224.49it/s]
100%|█████████████████████████████████████████████████████████████████████████| 24750/24750
[00:01<00:00, 21328.48it/s]
100%|█████████████████████████████████████████████████████████████████████████| 16583/16583
[00:00<00:00, 20601.77it/s]
```

2.6) Vectorizing Categorical Data

project_subject_categories (clean_categories)

In [26]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
def cat_vectorizer(X_train,df,col_name):
    vectorizer = CountVectorizer()
    vectorizer.fit(X_train[col_name].values)
    feature_one_hot = vectorizer.transform(df[col_name].values)
    print(vectorizer.get_feature_names())
    return feature_one_hot, vectorizer.get_feature_names()
```

In [27]:

```
x_train_cat_one_hot, x_train_cat_feat_list = cat_vectorizer(X_train,X_train,'clean_categories')
x_test_cat_one_hot, x_test_cat_feat_list = cat_vectorizer(X_train,X_test,'clean_categories')
x_cv_cat_one_hot, x_cat_cat_feat_list = cat_vectorizer(X_train,X_cv,'clean_categories')
```

```
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
```

```
'math_science', 'music_arts', 'specialneeds', 'warmth']
```

In [28]:

```
# shape after categorical one hot encoding
print(x_train_cat_one_hot.shape)
print(x_test_cat_one_hot.shape)
print(x_cv_cat_one_hot.shape)
```

```
(33667, 9)
(24750, 9)
(16583, 9)
```

project_subject_subcategory (clean_subcategory)

In [29]:

```
x_train_subcat_one_hot, x_train_subcat_feat_list =
cat_vectorizer(X_train,X_train,'clean_subcategories')
x_test_subcat_one_hot, x_test_subcat_feat_list =
cat_vectorizer(X_train,X_test,'clean_subcategories')
x_cv_subcat_one_hot, x_cv_subcat_feat_list = cat_vectorizer(X_train,X_cv,'clean_subcategories')
```

```
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience',
'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness',
'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'm
athematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socia
lsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience',
'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness',
'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'm
athematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socia
lsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience',
'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness',
'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'm
athematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socia
lsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
```

In [30]:

```
# shape after categorical one hot encoding
print(x_train_subcat_one_hot.shape)
print(x_test_subcat_one_hot.shape)
print(x_cv_subcat_one_hot.shape)
```

```
(33667, 30)
(24750, 30)
(16583, 30)
```

school_state

In [31]:

```
# we use count vectorizer to convert the values into one hot encoding
# CountVectorizer for "school_state"
x_train_state_one_hot, x_train_state_feat_list = cat_vectorizer(X_train,X_train,'school_state')
x_test_state_one_hot, x_test_state_feat_list = cat_vectorizer(X_train,X_test,'school_state')
x_cv_state_one_hot, x_cv_state_feat_list = cat_vectorizer(X_train,X_cv,'school_state')
```

```
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'k
s', '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']
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'k
s', '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']
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'k
s', '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']
```

In [32]:

```
# shape after categorical one hot encoding
print(x_train_state_one_hot.shape)
print(x_test_state_one_hot.shape)
print(x_cv_state_one_hot.shape)
```

```
(33667, 51)
(24750, 51)
(16583, 51)
```

teacher_prefix

In [33]:

```
# we use count vectorizer to convert the values into one hot encoding
# CountVectorizer for teacher_prefix
x_train_teacher_prefix_one_hot,x_train_teacher_prefix_feat_list = cat_vectorizer(X_train,X_train,'
teacher_prefix')
x_test_teacher_prefix_one_hot,x_test_teacher_prefix_feat_list =
cat_vectorizer(X_train,X_test,'teacher_prefix')
x_cv_teacher_prefix_one_hot,x_cv_teacher_prefix_feat_list =
cat_vectorizer(X_train,X_cv,'teacher_prefix')
```

```
['dr', 'mr', 'mrs', 'ms', 'teacher']
['dr', 'mr', 'mrs', 'ms', 'teacher']
['dr', 'mr', 'mrs', 'ms', 'teacher']
```

In [34]:

```
# shape after categorical one hot encoding
print(x_train_teacher_prefix_one_hot.shape)
print(x_test_teacher_prefix_one_hot.shape)
print(x_cv_teacher_prefix_one_hot.shape)
```

```
(33667, 5)
(24750, 5)
(16583, 5)
```

project_grade_category

In [35]:

```
# using count vectorizer for one-hot encoding of project_grade_category
x_train_grade_one_hot, x_train_grade_feat_list = cat_vectorizer(X_train,X_train,'clean_grade')
x_test_grade_one_hot, x_test_grade_feat_list = cat_vectorizer(X_train,X_test,'clean_grade')
x_cv_grade_one_hot, x_cv_grade_feat_list = cat_vectorizer(X_train,X_cv,'clean_grade')
```

```
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
```

In [36]:

```
# shape after categorical one hot encoding
print(x_train_grade_one_hot.shape)
print(x_test_grade_one_hot.shape)
print(x_cv_grade_one_hot.shape)
```

```
(33667, 4)
(24750, 4)
(16583, 4)
```

2.7) Vectorizing Text Data

2.7.1) Bag of Words (essay)

In [37]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
def bow_vectorizer(X_train,col_name,df):
    vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
    vectorizer.fit(X_train[col_name].values)
    df_bow = vectorizer.transform(df[col_name].values)
    return df_bow, vectorizer.get_feature_names()
```

In [38]:

```
x_train_essay_bow, x_train_essay_feat = bow_vectorizer(X_train,'essay',X_train)
x_test_essay_bow,  x_test_essay_feat  = bow_vectorizer(X_train,'essay',X_test)
x_cv_essay_bow,  x_cv_essay_feat  = bow_vectorizer(X_train,'essay',X_cv)
```

In [39]:

```
print(x_train_essay_bow.shape)
print(x_test_essay_bow.shape)
print(x_cv_essay_bow.shape)
```

```
(33667, 5000)
(24750, 5000)
(16583, 5000)
```

2.7.1) TFIDF (essay)

In [40]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
# We are considering only the words which appeared in at least 10 documents(rows or projects).
def tfidf_vectorizer(X_train,col_name,df):
    vectorizer = TfidfVectorizer(min_df=10)
    vectorizer.fit(X_train[col_name].values)
    df_tfidf = vectorizer.transform(df[col_name].values)
    return df_tfidf, vectorizer.get_feature_names()
```

In [41]:

```
# Lets vectorize essay
x_train_essay_tfidf, x_train_essay_tfidf_feat = tfidf_vectorizer(X_train,'essay',X_train)
x_test_essay_tfidf, x_test_essay_tfidf_feat = tfidf_vectorizer(X_train,'essay',X_test)
x_cv_essay_tfidf, x_cv_essay_tfidf_feat = tfidf_vectorizer(X_train,'essay',X_cv)
```

In [42]:

```
print(x_train_essay_tfidf.shape)
print(x_test_essay_tfidf.shape)
print(x_cv_essay_tfidf.shape)
```

```
(33667, 10755)
(24750, 10755)
(16583, 10755)
```

2.8) Vectorizing Numerical Features

2.8) Vectorizing Numerical Features

In [43]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html
from sklearn.preprocessing import Normalizer
def scaler_function(df,col_name):

    normalizer = Normalizer()
    normalizer.fit(df[col_name].values.reshape(1,-1)) # finding the mean and standard deviation of
this data

    # Now standardize the data with above maen and variance.
    normalized = normalizer.transform(df[col_name].values.reshape(1, -1))
    return normalized
```

teacher_number_of_previously_posted_projects

In [44]:

```
x_train_teacher_number = scaler_function(X_train,'teacher_number_of_previously_posted_projects')
x_test_teacher_number = scaler_function(X_test,'teacher_number_of_previously_posted_projects')
x_cv_teacher_number = scaler_function(X_cv,'teacher_number_of_previously_posted_projects')
```

In [45]:

```
x_train_teacher_number
```

Out[45]:

```
array([[0.00140748, 0.        , 0.0005278 , ..., 0.        , 0.00017593,
        0.00105561]])
```

2.9) Merging all the above features

In [46]:

```
# train dataset
print("After Vectorization and One hot encoding train dataset shape becomes:")
print(x_train_cat_one_hot.shape)
print(x_train_subcat_one_hot.shape)
print(x_train_state_one_hot.shape)
print(x_train_teacher_prefix_one_hot.shape)
print(x_train_grade_one_hot.shape)
print(x_train_essay_bow.shape)
print(x_train_essay_tfidf.shape)
print(x_train_teacher_number.shape)
print("="*50)

# test dataset
print("After Vectorization and One hot encoding test dataset shape becomes:")
print(x_test_cat_one_hot.shape)
print(x_test_subcat_one_hot.shape)
print(x_test_state_one_hot.shape)
print(x_test_teacher_prefix_one_hot.shape)
print(x_test_grade_one_hot.shape)
print(x_test_essay_bow.shape)
print(x_test_essay_tfidf.shape)
print(x_test_teacher_number.shape)
print("="*50)

# cv dataset
print("After Vectorization and One hot encoding cv dataset shape becomes:")
print(x_cv_cat_one_hot.shape)
print(x_cv_subcat_one_hot.shape)
print(x_cv_state_one_hot.shape)
print(x_cv_teacher_prefix_one_hot.shape)
print(x_cv_grade_one_hot.shape)
print(x_cv_essay_bow.shape)
print(x_cv_essay_tfidf.shape)
print(x_cv_teacher_number.shape)
print("="*50)
```

```
After Vectorization and One hot encoding train dataset shape becomes:
(33667, 9)
(33667, 30)
(33667, 51)
(33667, 5)
(33667, 4)
(33667, 5000)
(33667, 10755)
(1, 33667)
==================================================
After Vectorization and One hot encoding test dataset shape becomes:
(24750, 9)
(24750, 30)
(24750, 51)
(24750, 5)
(24750, 4)
(24750, 5000)
(24750, 10755)
(1, 24750)
==================================================
After Vectorization and One hot encoding cv dataset shape becomes:
(16583, 9)
(16583, 30)
(16583, 51)
(16583, 5)
(16583, 4)
(16583, 5000)
(16583, 10755)
(1, 16583)
==================================================
```

3) Apply NB on Set 1

Set_1: categorical, numerical features + preprocessed_eassay (BOW)

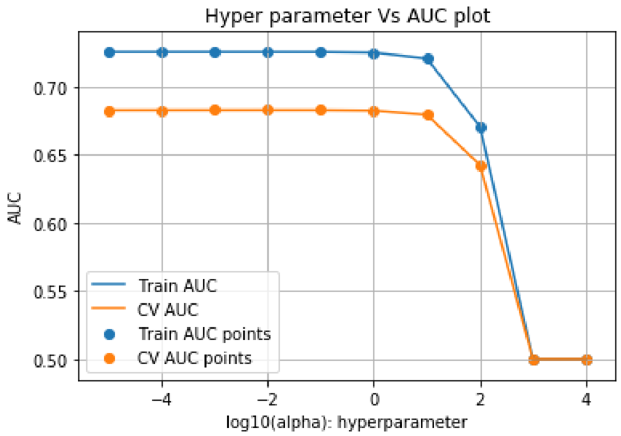
In [47]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)

X_train_set_1 =
hstack((x_train_cat_one_hot,x_train_subcat_one_hot,x_train_state_one_hot,x_train_teacher_prefix_one
_hot,\
                         x_train_grade_one_hot,x_train_essay_bow)).tocsr()
X_test_set_1 =
hstack((x_test_cat_one_hot,x_test_subcat_one_hot,x_test_state_one_hot,x_test_teacher_prefix_one_hot
,\
                         x_test_grade_one_hot,x_test_essay_bow)).tocsr()
X_cv_set_1 =
hstack((x_cv_cat_one_hot,x_cv_subcat_one_hot,x_cv_state_one_hot,x_cv_teacher_prefix_one_hot,\
                         x_cv_grade_one_hot,x_cv_essay_bow)).tocsr()
```

In [48]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
# for class prior i referred https://scikit-
learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html,\
# and https://stackoverflow.com/questions/42498208/setting-prior-probabilities-in-naive-bayes-mult
inomialnb
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.naive_bayes import MultinomialNB
import math
```

```
multiNB = MultinomialNB(class_prior=[0.5,0.5])
parameters = {'alpha':[10**x for x in range(-5,5)]}
clf = RandomizedSearchCV(multiNB, parameters, cv=20, scoring='roc_auc')
clf.fit(X_train_set_1, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_alpha'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
alpha_ =  results['param_alpha'].apply(lambda x: math.log10(x))

plt.plot(alpha_, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.2,color='darkblue')

plt.plot(alpha_, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

plt.scatter(alpha_, train_auc, label='Train AUC points')
plt.scatter(alpha_, cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("log10(alpha): hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```



Out[48]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_scor
0	0.286377	0.007276	0.009451	0.005410	1e-05	{'alpha': 1e-05}	0.685373	0.677569
1	0.294686	0.014868	0.011606	0.005097	0.0001	{'alpha': 0.0001}	0.685373	0.677569
2	0.292897	0.011080	0.009799	0.004384	0.001	{'alpha': 0.001}	0.685371	0.677569
3	0.285426	0.010836	0.010562	0.004365	0.01	{'alpha': 0.01}	0.685371	0.677564
4	0.293545	0.012020	0.009438	0.003439	0.1	{'alpha': 0.1}	0.685335	0.677528

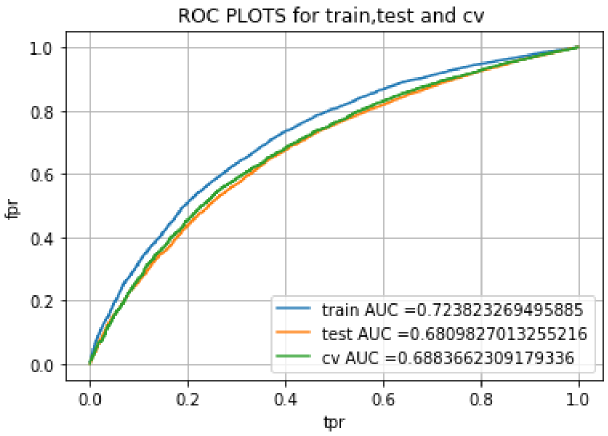
5 rows × 51 columns

In [49]:

```
# From the AUC plot, we find that the best value for "aplha" - "smoothing factor" for the MultinomialNB is 0.1
best_alpha = 0.1
```

In [50]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


multiNB = MultinomialNB(alpha=best_alpha, class_prior = [0.5,0.5])
multiNB.fit(X_train_set_1, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = multiNB.predict_proba(X_train_set_1)
y_test_pred =  multiNB.predict_proba(X_test_set_1)
y_cv_pred = multiNB.predict_proba(X_cv_set_1)

y_train_pred_prob = []
y_test_pred_prob = []
y_cv_pred_prob = []

for index in range(len(y_train_pred)):
    y_train_pred_prob.append(y_train_pred[index][1])

for index in range(len(y_test_pred)):
    y_test_pred_prob.append(y_test_pred[index][1])

for index in range(len(y_cv_pred)):
    y_cv_pred_prob.append(y_cv_pred[index][1])


train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred_prob)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred_prob)
cv_fpr, cv_tpr, cv_thresholds = roc_curve(y_cv,y_cv_pred_prob)



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.plot(cv_fpr, cv_tpr, label="cv AUC ="+str(auc(cv_fpr, cv_tpr)))

plt.legend()
plt.xlabel("tpr")
plt.ylabel("fpr")
plt.title("ROC PLOTS for train,test and cv ")
plt.grid()
plt.show()
```



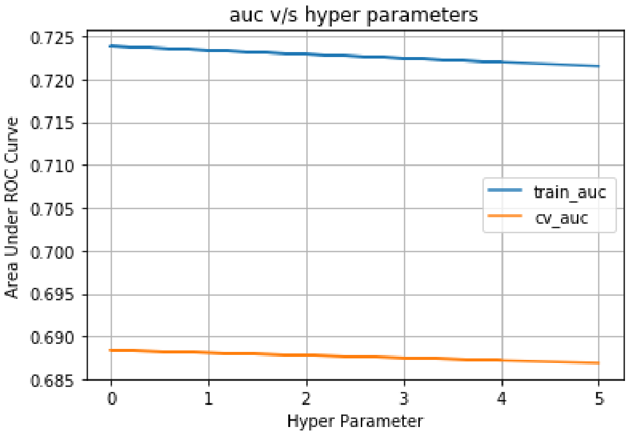
In [51]:

```
def compute_auc_with_hyper_para(x_tr,y_tr,x_cv,y_cv,para_list):
    auc_tr = []
    auc_cv = []
    y_train_pred_prob = []
    y_cv_pred_prob = []


    for para in para_list:
        multiNB = MultinomialNB(alpha = para)
        multiNB.fit(x_tr,y_tr)

        y_train_pred = multiNB.predict_proba(x_tr)
        y_cv_pred = multiNB.predict_proba(x_cv)

        for index in range(len(y_train_pred)):
            y_train_pred_prob.append(y_train_pred[index][1])

        for index in range(len(y_cv_pred)):
            y_cv_pred_prob.append(y_cv_pred[index][1])

        train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_train_pred_prob)
        cv_fpr, cv_tpr, tc_thresholds = roc_curve(y_cv, y_cv_pred_prob)

        y_train_pred_prob = []
        y_cv_pred_prob = []



        auc_tr.append(auc(train_fpr,train_tpr))
        auc_cv.append(auc(cv_fpr,cv_tpr))
    plt.plot(para_list,auc_tr,label="train_auc")
    plt.plot(para_list,auc_cv,label="cv_auc")
    plt.legend()
    plt.xlabel("Hyper Parameter")
    plt.ylabel("Area Under ROC Curve")
    plt.title("auc v/s hyper parameters")
    plt.grid()
    plt.show()
```

In [52]:

```
import math
para_list = [10**x for x in range(-5,5)]
log_list = [abs(math.log10(i)) for i in para_list]
compute_auc_with_hyper_para(X_train_set_1,y_train,X_cv_set_1,y_cv,log_list)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\naive_bayes.py:472: UserWarning:

alpha too small will result in numeric errors, setting alpha = 1.0e-10
```



In [53]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

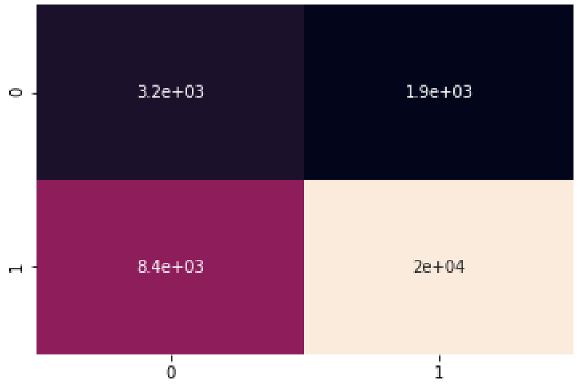
In [54]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
sns.heatmap(confusion_matrix(y_train, predict_with_best_t(y_train_pred_prob, best_t)),annot = True,
cbar=False)
```

```
====================================================================================================

the maximum value of tpr*(1-fpr) 0.4455162457465279 for threshold 0.215
Train confusion matrix
```

Out[54]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x186be02ff98>
```



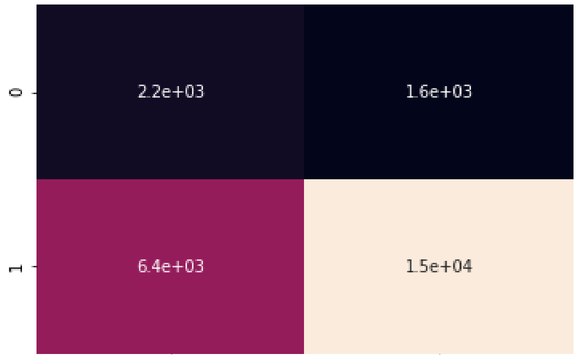
In [55]:

```
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, predict_with_best_t(y_test_pred_prob, best_t)), annot = True,
cbar=False)
```

```
Test confusion matrix
```

Out[55]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x186be04f6a0>
```



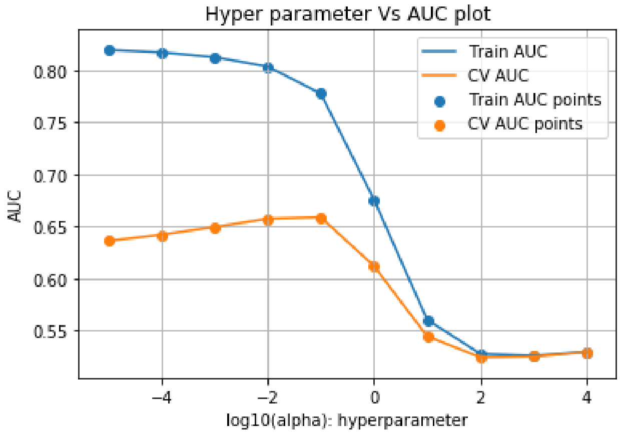
Set_2: categorical, numerical features + preprocessed_eassay (TFIDF)

In [56]:

```
X_train_set_2 =
hstack((x_train_cat_one_hot,x_train_subcat_one_hot,x_train_state_one_hot,x_train_teacher_prefix_one
_hot,\
                         x_train_grade_one_hot,x_train_essay_tfidf)).tocsr()
X_test_set_2 =
hstack((x_test_cat_one_hot,x_test_subcat_one_hot,x_test_state_one_hot,x_test_teacher_prefix_one_hot
,\
                         x_test_grade_one_hot,x_test_essay_tfidf)).tocsr()
X_cv_set_2 =
hstack((x_cv_cat_one_hot,x_cv_subcat_one_hot,x_cv_state_one_hot,x_cv_teacher_prefix_one_hot,\
                         x_cv_grade_one_hot,x_cv_essay_tfidf)).tocsr()
```

In [57]:

```
multiNB = MultinomialNB()
parameters = {'alpha':[10**x for x in range(-5,5)]}
clf = RandomizedSearchCV(multiNB, parameters, cv=20, scoring='roc_auc')
clf.fit(X_train_set_2, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_alpha'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
alpha_=  results['param_alpha'].apply(lambda x: math.log10(x))

plt.plot(alpha_, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.2,color='darkblue')

plt.plot(alpha_, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

plt.scatter(alpha_, train_auc, label='Train AUC points')
plt.scatter(alpha_, cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("log10(alpha): hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```



Out[57]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_scor
0	0.147249	0.010381	0.004084	0.003489	1e-05	{'alpha': 1e-05}	0.644791	0.655356
1	0.144199	0.007538	0.006046	0.004886	0.0001	{'alpha': 0.0001}	0.651226	0.657592
2	0.144627	0.007819	0.005249	0.004802	0.001	{'alpha': 0.001}	0.658295	0.661154
3	0.144403	0.007727	0.003866	0.004386	0.01	{'alpha': 0.01}	0.664437	0.664533
4	0.144953	0.010063	0.005220	0.004862	0.1	{'alpha': 0.1}	0.664516	0.655993

5 rows × 51 columns

In [58]:

```
# From the AUC plot, we find that the best value for "alpha" - "smoothing factor" for the Multinom
ialNB is 0.01
best_alpha_2 = 0.01
```

In [59]:

```
multiNB = MultinomialNB(alpha=best_alpha_2)
multiNB.fit(X_train_set_2, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = multiNB.predict_proba(X_train_set_2)
y_test_pred = multiNB.predict_proba(X_test_set_2)
y_cv_pred = multiNB.predict_proba(X_cv_set_2)

y_train_pred_prob = []
y_test_pred_prob = []
y_cv_pred_prob = []

for index in range(len(y_train_pred)):
    y_train_pred_prob.append(y_train_pred[index][1])

for index in range(len(y_test_pred)):
    y_test_pred_prob.append(y_test_pred[index][1])

for index in range(len(y_cv_pred)):
    y_cv_pred_prob.append(y_cv_pred[index][1])

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred_prob)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred_prob)
cv_fpr, cv_tpr, cv_thresholds = roc_curve(y_cv, y_test_pred_prob)



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.plot(cv_fpr, cv_tpr, label="cv AUC ="+str(auc(cv_fpr, cv_tpr)))


plt.legend()
plt.xlabel("tpr")
plt.ylabel("fpr")
plt.title("ROC PLOTS for train, test and cv")
plt.grid()
plt.show()
```

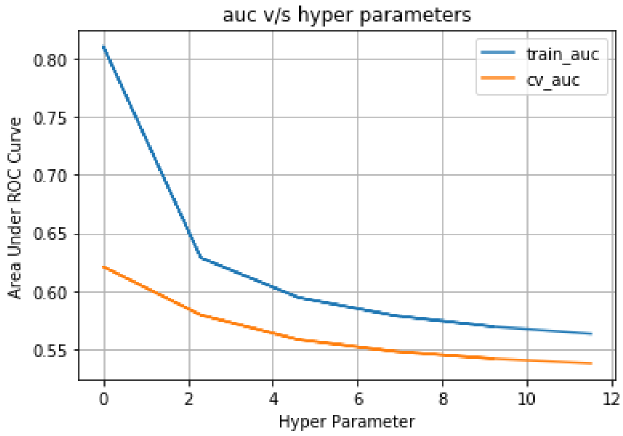
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-59-99efae93b8e1> in <module>()
```

```
<ipython-input-59-90efce92b8e1> in <module>()
     23 train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred_prob)
     24 test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred_prob)
---> 25 cv_fpr, cv_tpr, cv_thresholds = roc_curve(y_cv, y_test_pred_prob)
     26 
     27 

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\ranking.py in roc_curve(y_true,
y_score, pos_label, sample_weight, drop_intermediate)
    532     """
    533     fps, tps, thresholds = _binary_clf_curve(
--> 534         y_true, y_score, pos_label=pos_label, sample_weight=sample_weight)
    535 
    536     # Attempt to drop thresholds corresponding to points in between and

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\ranking.py in _binary_clf_curve(y_true,
y_score, pos_label, sample_weight)
    318         raise ValueError("{0} format is not supported".format(y_type))
    319 
--> 320     check_consistent_length(y_true, y_score, sample_weight)
    321     y_true = column_or_1d(y_true)
    322     y_score = column_or_1d(y_score)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in check_consistent_length(
*arrays)
    202     if len(uniques) > 1:
    203         raise ValueError("Found input variables with inconsistent numbers of"
--> 204                          " samples: %r" % [int(l) for l in lengths])
    205 
    206 

ValueError: Found input variables with inconsistent numbers of samples: [16583, 24750]
```

In [102]:

```
para_list = [10**x for x in range(-5,5)]
log_list = [abs(math.log(i)) for i in para_list]
compute_auc_with_hyper_para(X_train_set_2,y_train,X_cv_set_2,y_cv,log_list)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\naive_bayes.py:472: UserWarning:

alpha too small will result in numeric errors, setting alpha = 1.0e-10
```



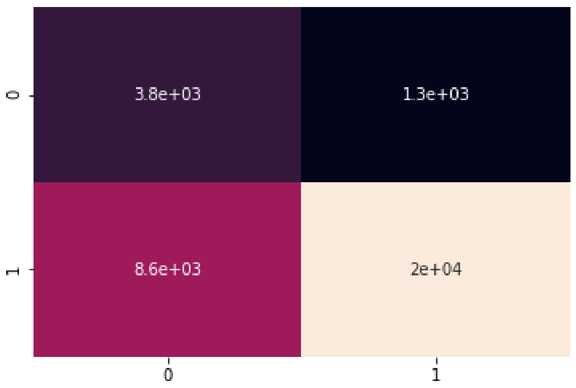
In [104]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
sns.heatmap(confusion_matrix(y_train, predict_with_best_t(y_train_pred_prob, best_t)),annot = True,
cbar = False)
```

```
====================================================================================================

the maximum value of tpr*(1-fpr) 0.5193181571515865 for threshold 0.856
Train confusion matrix
Train confusion matrix
```

Out[104]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21f74971160>
```



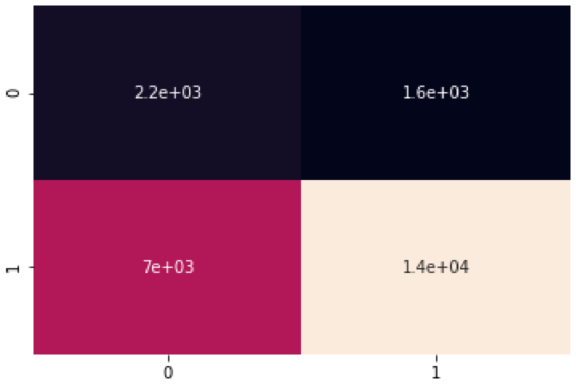
In [105]:

```
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, predict_with_best_t(y_test_pred_prob, best_t)),annot = True, cbar = False)
```

```
Test confusion matrix
```

Out[105]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21f4db41828>
```



In [106]:

```
# Let's append the features list we obtained as per the order the matrices are merged

global_features = []
global_features.extend(x_train_cat_feat_list)
global_features.extend(x_train_subcat_feat_list)
global_features.extend(x_train_state_feat_list)
global_features.extend(x_train_teacher_prefix_feat_list)
global_features.extend(x_train_grade_feat_list)
global_features.extend(x_train_essay_feat)
```

In [107]:

```
# the code from the following link has referred
# https://stats.stackexchange.com/questions/266031/what-is-log-probability-of-feature-in-sklearn-multinomialnb
#https://stackoverflow.com/questions/50526898/how-to-get-feature-importance-in-naive-bayes
multiNB = MultinomialNB(alpha = best_alpha_2)
naive_bayes = multiNB.fit(X_train_set_1,y_train)
# print(naive_bayes.predict(d[0].reshape(1,-1)))
# print(naive_bayes.predict_proba(d[0].reshape(1,-1)))
features = naive_bayes.feature_log_prob_[0,:].argsort()[:20][::-1]
```

```
scores = naive_bayes.feature_log_prob_[0,:]
```

In [108]:

```
# let's print the top-20 features and their feature_log_proba scores
top_20_features = np.take(global_features, features)
top_20_scores = [scores[i] for i in features]
```

In [109]:

```
for feature, score in zip(top_20_features, top_20_scores):
    print("Feature name:", feature,"-"*20,"feature_log_proba_score:",score)
```

```
Feature name: hi -------------------- feature_log_proba_score: -11.225817982373284
Feature name: the computers -------------------- feature_log_proba_score: -11.268359488559172
Feature name: chromebooks to -------------------- feature_log_proba_score: -11.312791497068076
Feature name: wv -------------------- feature_log_proba_score: -11.312791497068076
Feature name: the chromebooks -------------------- feature_log_proba_score: -11.312791497068076
Feature name: calculators -------------------- feature_log_proba_score: -11.359289877749537
Feature name: me -------------------- feature_log_proba_score: -11.408056244010805
Feature name: mt -------------------- feature_log_proba_score: -11.570486986198675
Feature name: these stools -------------------- feature_log_proba_score: -11.631074865597823
Feature name: sd -------------------- feature_log_proba_score: -11.631074865597823
Feature name: ak -------------------- feature_log_proba_score: -11.695571746961075
Feature name: economics -------------------- feature_log_proba_score: -11.695571746961075
Feature name: ne -------------------- feature_log_proba_score: -11.695571746961075
Feature name: ri -------------------- feature_log_proba_score: -11.76451703225753
Feature name: nh -------------------- feature_log_proba_score: -12.100703799279318
Feature name: de -------------------- feature_log_proba_score: -12.2059533209862
Feature name: wy -------------------- feature_log_proba_score: -12.456950691559948
Feature name: nd -------------------- feature_log_proba_score: -12.61086364405933
Feature name: vt -------------------- feature_log_proba_score: -13.015497151287969
Feature name: dr -------------------- feature_log_proba_score: -14.394338061753277
```

In [111]:

```
# referred the code from http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
summarizer = PrettyTable()
summarizer.field_names = ["vectorizer","Model","Hyper Parameter","AUC test"]
summarizer.add_row(["BOW","Multinomial NB",best_alpha,"0.68"])
summarizer.add_row(["TFIDF","Multinomial NB",best_alpha_2,"0.65"])
print(summarizer)
```

```
+------------+----------------+-----------------+----------+
| vectorizer |     Model      | Hyper Parameter | AUC test |
+------------+----------------+-----------------+----------+
|    BOW     | Multinomial NB |       0.1       |   0.68   |
|   TFIDF    | Multinomial NB |       0.01      |   0.65   |
+------------+----------------+-----------------+----------+
```