In [209]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

In [210]:

```
%cd drive/My Drive/Assignments_DonorsChoose_2018
```

```
[Errno 2] No such file or directory: 'drive/My Drive/Assignments_DonorsChoose_2018'
/content/drive/My Drive/Assignments_DonorsChoose_2018
```

In [211]:

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

In [0]:

```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

In [0]:

```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_data.columns)]

#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
```

```
#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
project_data.drop('project_submitted_datetime', axis=1, inplace=True)
project_data.sort_values(by=['Date'], inplace=True)

# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084039
project_data = project_data[cols]
#project_data.head(2)
```

In [0]:

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
#price_data.head(2)
```

In [0]:

```
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

1.2 preprocessing of `project_subject_categories`

In [0]:

```
catogories = list(project_data['project_subject_categories'].values)
# 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
cat_list = []
for i in catogories:
    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 & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "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 &
Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

project_data['clean_categories'] = cat_list
project_data.drop(['project_subject_categories'], axis=1, inplace=True)

from collections import Counter
my_counter = Counter()
for word in project_data['clean_categories'].values:
    my_counter.update(word.split())

cat_dict = dict(my_counter)
sorted_cat_dict = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))
```

1.3 preprocessing of `project_subject_subcategories`

In [0]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# 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

sub_cat_list = []
for i in sub_catogories:
```

```
    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 & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "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 &
Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

project_data['clean_subcategories'] = sub_cat_list
project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
for word in project_data['clean_subcategories'].values:
    my_counter.update(word.split())

sub_cat_dict = dict(my_counter)
sorted_sub_cat_dict = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))
```

1.3 Text preprocessing

In [0]:

```
# 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 [0]:

```
# 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 [0]:

```
# 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', "d
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 [0]:

```
def find_num(text):
    if re.findall(r'\d+', text):
        return 1
    return 0

project_data['numerical_digits'] = project_data['project_resource_summary'].apply(lambda x: find_nu
m(x))
```

In [222]:

```
project_data['project_grade_category']=project_data['project_grade_category'].str.replace(' ','_')
project_data['project_grade_category']=project_data['project_grade_category'].str.replace('-','to'
)
set(project_data['project_grade_category'])
```

Out[222]:

```
{'Grades_3to5', 'Grades_6to8', 'Grades_9to12', 'Grades_PreKto2'}
```

In [0]:

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

In [0]:

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score
from collections import Counter
import matplotlib.pyplot as plt
import numpy as np
from scipy.sparse import csr_matrix
import time

project_data_features = project_data.copy()
project_data_features.drop('project_is_approved', axis=1, inplace=True)
y=list(project_data['project_is_approved'])
X_train, X_test, y_train, y_test = train_test_split(project_data_features, y, stratify=y, test_size
=0.20)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, stratify=y_train, test_size=0.2
0)
```

In [0]:

```
from sklearn.preprocessing import StandardScaler
def standardize_data(df_tr,df_cv,df_te,column_name):
  standardized_vec = StandardScaler(with_mean=False)
  # here it will learn mu and sigma
  standardized_vec.fit(df_tr[column_name].values.reshape(-1,1))

  # with the learned mu and sigma it will do std on train data
  standardized_data_train = standardized_vec.transform(df_tr[column_name].values.reshape(-1,1))
  print(standardized_data_train.shape)
```

```
  # with the same learned mu and sigma it will do std on cv data
  standardized_data_traincv = standardized_vec.transform(df_cv[column_name].values.reshape(-1,1))
  print(standardized_data_traincv.shape)

  # with the same learned mu and sigma it will do std on test data
  standardized_data_test =standardized_vec.transform(df_te[column_name].values.reshape(-1,1))
  print(standardized_data_test.shape)

  return standardized_data_train, standardized_data_traincv, standardized_data_test
```

In [0]:

```
from sklearn.feature_extraction.text import CountVectorizer
def vectorized_data(df_train,df_cv,df_test,column_name,vocab=False):
  if(vocab):
    vectorizer = CountVectorizer(vocabulary=list(vocab.keys()), lowercase=False, binary=True)
  else:
    vectorizer = CountVectorizer(lowercase=False, binary=True)

  categories_one_hot_tr = vectorizer.fit_transform(df_train[column_name].values)
  print(vectorizer.get_feature_names())
  print("Shape of matrix after one hot encoding ",categories_one_hot_tr.shape)
  vocab_list = vectorizer.get_feature_names()

  categories_one_hot_cv = vectorizer.transform(df_cv[column_name].values)
  print(vectorizer.get_feature_names())
  print("Shape of matrix after one hot encoding ",categories_one_hot_cv.shape)

  categories_one_hot_te = vectorizer.transform(df_test[column_name].values)
  print(vectorizer.get_feature_names())
  print("Shape of matrix after one hot encoding ",categories_one_hot_te.shape)
  return categories_one_hot_tr,categories_one_hot_cv, categories_one_hot_te,vocab_list
```

In [0]:

```
def create_dict(df,column_name):
  my_counter = Counter()
  for word in df[column_name].values:
    my_counter.update(word.split())

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

In [0]:

```
def num_hot_encode(df,column_name):
  one_hot_num_dig = pd.get_dummies(df[column_name].values)
  print("Shape of matrix after one hot encoding ",one_hot_num_dig.shape)
  return one_hot_num_dig
```

In [0]:

```
from tqdm import tqdm
def textpreprocessed(df,column_name):
  # Combining all the above stundents
  preprocessed_list = []
  # tqdm is for printing the status bar
  for sentance in tqdm(df[column_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_list.append(sent.lower().strip())
  return preprocessed_list
```

In [0]:

```
from sklearn.preprocessing import Normalizer
def normalize_data(df,column_data):
  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(df[column_data].values.reshape(-1,1))

  data_norm = normalizer.transform(df[column_data].values.reshape(-1,1))
  print("After vectorizations")
  print(data_norm.shape)
  return data_norm
```

In [231]:

```
price_standardized_tr, price_standardized_val, price_standardized_te =
standardize_data(X_train,X_val, X_test,'price')
print()
project_standardized_tr, project_standardized_val, project_standardized_te =
standardize_data(X_train,X_val, X_test,'teacher_number_of_previously_posted_projects')
```

```
(69918, 1)
(17480, 1)
(21850, 1)

(69918, 1)
(17480, 1)
(21850, 1)
```

In [232]:

```
cat_one_hot_tr,cat_one_hot_val,cat_one_hot_te,cat_one_hot_list =
vectorized_data(X_train,X_val,X_test,'clean_categories',vocab=sorted_cat_dict)
cat_sub_one_hot_tr,cat_sub_one_hot_val,cat_sub_one_hot_te,cat_sub_one_hot_list = vectorized_data(X
_train,X_val,X_test,'clean_subcategories',vocab=sorted_cat_dict)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding  (69918, 9)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding  (17480, 9)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding  (21850, 9)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding  (69918, 9)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding  (17480, 9)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding  (21850, 9)
```

In [233]:

```
school_state_dict = create_dict(X_train,'school_state')
teacher_prefix_dict = create_dict(X_train,'teacher_prefix')

state_one_hot_tr,state_one_hot_val,state_one_hot_te,state_one_hot_list = vectorized_data(X_train,X
_val,X_test,'school_state',vocab=school_state_dict)
teacher_one_hot_tr,teacher_one_hot_val,teacher_one_hot_te,teacher_one_hot_list = vectorized_data(X
_train,X_val,X_test,'teacher_prefix',vocab=teacher_prefix_dict)
```

```
['VT', 'WY', 'ND', 'MT', 'RI', 'SD', 'NE', 'NH', 'DE', 'AK', 'ME', 'WV', 'HI', 'DC', 'NM', 'KS', 'I
A', 'ID', 'AR', 'CO', 'MN', 'KY', 'OR', 'MS', 'NV', 'MD', 'CT', 'TN', 'UT', 'AL', 'WI', 'AZ', 'VA',
'NJ', 'WA', 'OK', 'LA', 'MA', 'OH', 'MO', 'IN', 'PA', 'MI', 'GA', 'SC', 'IL', 'NC', 'FL', 'NY', 'TX
', 'CA']
```

```
Shape of matrix after one hot encoding  (69918, 51)
['VT', 'WY', 'ND', 'MT', 'RI', 'SD', 'NE', 'NH', 'DE', 'AK', 'ME', 'WV', 'HI', 'DC', 'NM', 'KS', 'I
A', 'ID', 'AR', 'CO', 'MN', 'KY', 'OR', 'MS', 'NV', 'MD', 'CT', 'TN', 'UT', 'AL', 'WI', 'AZ', 'VA',
'NJ', 'WA', 'OK', 'LA', 'MA', 'OH', 'MO', 'IN', 'PA', 'MI', 'GA', 'SC', 'IL', 'NC', 'FL', 'NY', 'TX
', 'CA']
Shape of matrix after one hot encoding  (17480, 51)
['VT', 'WY', 'ND', 'MT', 'RI', 'SD', 'NE', 'NH', 'DE', 'AK', 'ME', 'WV', 'HI', 'DC', 'NM', 'KS', 'I
A', 'ID', 'AR', 'CO', 'MN', 'KY', 'OR', 'MS', 'NV', 'MD', 'CT', 'TN', 'UT', 'AL', 'WI', 'AZ', 'VA',
'NJ', 'WA', 'OK', 'LA', 'MA', 'OH', 'MO', 'IN', 'PA', 'MI', 'GA', 'SC', 'IL', 'NC', 'FL', 'NY', 'TX
', 'CA']
Shape of matrix after one hot encoding  (21850, 51)
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
Shape of matrix after one hot encoding  (69918, 5)
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
Shape of matrix after one hot encoding  (17480, 5)
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
Shape of matrix after one hot encoding  (21850, 5)
```

In [234]:

```
grade_dict = create_dict(X_train,'project_grade_category')
grade_one_hot_tr,grade_one_hot_val,grade_one_hot_te,grade_one_hot_list = vectorized_data(X_train,X
_val,X_test,'project_grade_category',vocab=grade_dict)
```

```
['Grades_9to12', 'Grades_6to8', 'Grades_3to5', 'Grades_PreKto2']
Shape of matrix after one hot encoding  (69918, 4)
['Grades_9to12', 'Grades_6to8', 'Grades_3to5', 'Grades_PreKto2']
Shape of matrix after one hot encoding  (17480, 4)
['Grades_9to12', 'Grades_6to8', 'Grades_3to5', 'Grades_PreKto2']
Shape of matrix after one hot encoding  (21850, 4)
```

In [235]:

```
one_hot_num_dig_tr = num_hot_encode(X_train,'numerical_digits')
one_hot_num_dig_te = num_hot_encode(X_test,'numerical_digits')
one_hot_num_dig_val = num_hot_encode(X_val,'numerical_digits')
```

```
Shape of matrix after one hot encoding  (69918, 2)
Shape of matrix after one hot encoding  (21850, 2)
Shape of matrix after one hot encoding  (17480, 2)
```

[Task-1] Apply KNN(brute force version) on these feature sets

In [0]:

```
from scipy.sparse import hstack

f_tr =
hstack((one_hot_num_dig_tr,grade_one_hot_tr,state_one_hot_tr,cat_one_hot_tr,cat_sub_one_hot_tr,teac
her_one_hot_tr,price_standardized_tr,project_standardized_tr))
f_cr =
hstack((one_hot_num_dig_val,grade_one_hot_val,state_one_hot_val,cat_one_hot_val,cat_sub_one_hot_val
,teacher_one_hot_val,price_standardized_val,project_standardized_val))
f_te =
hstack((one_hot_num_dig_te,grade_one_hot_te,state_one_hot_te,cat_one_hot_te,cat_sub_one_hot_te,teac
her_one_hot_te,price_standardized_te,project_standardized_te))

def hstack_data(f1_tr, f1_cr, f1_te, f2_tr, f2_cr, f2_te,f3_tr,f3_cr,f3_te):
  X_tr = hstack((f_tr, f1_tr, f2_tr,f3_tr)).tocsr()
  X_cr = hstack((f_cr, f1_cr, f2_cr,f3_cr)).tocsr()
  X_te = hstack((f_te, f1_te, f2_te,f3_te)).tocsr()
  return X_tr,X_cr,X_te
```

In [237]:

```
feature_list_x =
['dig_0','dig_1']+grade_one_hot_list+state_one_hot_list+cat_one_hot_list+cat_sub_one_hot_list+teach
er_one_hot_list+['price','teacher_number_of_previously_posted_projects']
len(feature_list_x)
```

Out[237]:

82

In [0]:

```
# https://stackoverflow.com/a/26980472/6660373
def most_informative_feature_for_binary_classification(feature_names, classifier, n=10):
    class_labels = classifier.classes_
    feature_names = feature_names
    topn_class1 = sorted(zip(classifier.coef_[0], feature_names))[:n]
    topn_class2 = sorted(zip(classifier.coef_[0], feature_names))[-n:]

    for coef, feat in topn_class1:
        print(class_labels[0], coef, feat)

    print("==========================================================================")

    for coef, feat in reversed(topn_class2):
        print(class_labels[1], coef, feat)
```

In [0]:

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%5000
    # consider you X_tr shape is 49041, then your cr_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, 5000):
        y_data_pred.extend(clf.predict_proba(data[i:i+5000])[:,1])
    # we will be predicting for the last data points
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [0]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
import time
#f = open("myfile.txt", "a")
def optimised_alpha(X_tr,y_train,X_cr,y_val):
  """
  y_true : array, shape = [n_samples] or [n_samples, n_classes]
  True binary labels or binary label indicators.

  y_score : array, shape = [n_samples] or [n_samples, n_classes]
  Target scores, can either be probability estimates of the positive class, confidence values, or
non-thresholded measure of
  decisions (as returned by "decision_function" on some classifiers).
  For binary y_true, y_score is supposed to be the score of the class with greater label.

  """
  start = time.time()

  train_auc = []
  cv_auc = []
  alpha_list = np.arange(0.00001, 100, 0.25)
  for alpha in tqdm(alpha_list):
    mnb = MultinomialNB(alpha = alpha)
    mnb.fit(X_tr, y_train)

    y_train_pred = batch_predict(mnb, X_tr)
    y_cv_pred = batch_predict(mnb, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_val, y_cv_pred))
```

```
  plt.plot(list(np.log(alpha_list)), train_auc, label='Train AUC')
  plt.plot(list(np.log(alpha_list)), cv_auc, label='CV AUC')

  plt.scatter(list(np.log(alpha_list)), train_auc, label='Train AUC points',s=10)
  plt.scatter(list(np.log(alpha_list)), cv_auc, label='CV AUC points',s=10)

  plt.legend()
  plt.xlabel("alpha: hyperparameter")
  plt.ylabel("AUC")
  plt.title("ERROR PLOTS")
  plt.grid()
  plt.show()
  end = time.time()
  minutes = int((end - start)/60)
  print("execution time in minutes:",minutes)
  print("execution time in hours:",int(minutes/60))
  print(max(cv_auc))
  optimalalpha = alpha_list[np.argmax(cv_auc)]
  print("optimal alpha is: ",optimalalpha)
  return optimalalpha
```

In [0]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

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

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [0]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix
import seaborn as sns
def error_plot(X_train,X_te,y_train,y_test):
  global mnb
  global clf
  mnb = MultinomialNB(alpha = best_alpha)
  clf = mnb.fit(X_train, y_train)
  # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positi
ve class
  # not the predicted outputs

  y_train_pred = batch_predict(mnb, X_train)
  y_test_pred = batch_predict(mnb, X_te)

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

  plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
  plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
  plt.legend()
  plt.xlabel("Alpha: hyperparameter")
  plt.ylabel("AUC")
  plt.title("ERROR PLOTS")
  plt.grid()
  plt.show()
  print("="*100)

  print("Train confusion matrix")
  print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
  print("Test confusion matrix")
```

```
print("Test Confusion Matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))

class_label = ["0:rejected", "1:approved"]
cm = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

class_label = ["0:rejected", "1:approved"]
cm = confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr))
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

In [0]:

```
BLUE = '34m'
def display_colored_text(color, text):
    colored_text = f"\033[{color}{text}\033[00m"
    return colored_text
```

In [0]:

```
#Code Reference:https://stackoverflow.com/questions/11116697/how-to-get-most-informative-features-for-scikit-learn-classifiers
def show_most_informative_features(feature_names, clf, n=10):
    #feature_names = vectorizer.get_feature_names()
    coefs_with_fns = sorted(zip(clf.coef_[0], feature_names))
    #print(list(coefs_with_fns))
    print(display_colored_text(BLUE, "\t\tPositive\t\t\t   Negative"))
    print('      ============================================================')
    #print("\t\tPositive\t\tNegative")
    top = zip(coefs_with_fns[:n], coefs_with_fns[:-(n + 1):-1])
    for (coef_1, fn_1), (coef_2, fn_2) in top:
        print("\t%.4f\t%-15s\t\t%.4f\t%-15s" % (coef_1, fn_1, coef_2, fn_2))
```

In [0]:

```
def important_features(feature_names,classifier,n=10):
  class_labels = classifier.classes_
  feature_names =feature_names
  topn_class1 = sorted(zip(classifier.feature_log_prob_[0], feature_names),reverse=True)[:n]
  topn_class2 = sorted(zip(classifier.feature_log_prob_[1], feature_names),reverse=True)[:n]
  print(display_colored_text(BLUE,"Top 10 features for negative class"))
  for coef, feat in topn_class1:
    print(class_labels[0], coef, feat)
  print("-----------------------------------------")
  print(display_colored_text(BLUE,"Top 10 features for positive class"))
  for coef, feat in topn_class2:
    print(class_labels[1], coef, feat)
```

Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF)

1.5.1 Vectorizing Categorical data

In [0]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
def tfidf_vec(preprocessed_data_tr,preprocessed_data_val,preprocessed_data_te):
  global vectorizer_tfidf
  vectorizer_tfidf = TfidfVectorizer(min_df=10)
  text_tfidf_tr = vectorizer_tfidf.fit_transform(preprocessed_data_tr)
  vectorizer_tf = vectorizer_tfidf.get_feature_names()
  print("Shape of matrix after one hot encodig ",text_tfidf_tr.shape)
```

```
  text_tfidf_val = vectorizer_tfidf.transform(preprocessed_data_val)
  print("Shape of matrix after one hot encodig ",text_tfidf_val.shape)

  text_tfidf_te = vectorizer_tfidf.transform(preprocessed_data_te)
  print("Shape of matrix after one hot encodig ",text_tfidf_te.shape)
  return text_tfidf_tr,text_tfidf_val, text_tfidf_te, vectorizer_tf
```

In [282]:

```
tfidf_vec_essay_tr,tfidf_vec_essay_val,tfidf_vec_essay_te,tfidf_vec_essay_list =
tfidf_vec(textpreprocessed(X_train,'essay'),textpreprocessed(X_val,'essay'), textpreprocessed(X_te
st,'essay'))
tfidf_vec_titles_tr,tfidf_vec_titles_val,tfidf_vec_titles_te,tfidf_vec_titles_list =
tfidf_vec(textpreprocessed(X_train,'project_title'),textpreprocessed(X_val,'project_title'), textp
reprocessed(X_test,'project_title'))
tfidf_vec_resource_tr,tfidf_vec_resource_val,tfidf_vec_resource_te,tfidf_vec_resource_list =
tfidf_vec(textpreprocessed(X_train,'project_resource_summary'),textpreprocessed(X_val,'project_resc
urce_summary'), textpreprocessed(X_test,'project_resource_summary'))
```

```
100%|██████████| 69918/69918 [00:43<00:00, 1599.49it/s]
100%|██████████| 17480/17480 [00:10<00:00, 1592.22it/s]
100%|██████████| 21850/21850 [00:13<00:00, 1622.55it/s]

Shape of matrix after one hot encodig  (69918, 13888)
Shape of matrix after one hot encodig  (17480, 13888)

  5%|▌          | 3408/69918 [00:00<00:01, 34079.71it/s]

Shape of matrix after one hot encodig  (21850, 13888)

100%|██████████| 69918/69918 [00:02<00:00, 34218.27it/s]
100%|██████████| 17480/17480 [00:00<00:00, 33694.48it/s]
100%|██████████| 21850/21850 [00:00<00:00, 33987.97it/s]

Shape of matrix after one hot encodig  (69918, 2463)
Shape of matrix after one hot encodig  (17480, 2463)

  2%|▏          | 1522/69918 [00:00<00:04, 15211.31it/s]

Shape of matrix after one hot encodig  (21850, 2463)

100%|██████████| 69918/69918 [00:04<00:00, 15337.43it/s]
100%|██████████| 17480/17480 [00:01<00:00, 15203.29it/s]
100%|██████████| 21850/21850 [00:01<00:00, 15253.75it/s]

Shape of matrix after one hot encodig  (69918, 4639)
Shape of matrix after one hot encodig  (17480, 4639)
Shape of matrix after one hot encodig  (21850, 4639)
```

In [283]:

```
print(one_hot_num_dig_tr.shape)
print(grade_one_hot_tr.shape)
print(state_one_hot_tr.shape)
print(cat_one_hot_tr.shape)
print(cat_sub_one_hot_tr.shape)
print(teacher_one_hot_tr.shape)
print(price_standardized_tr.shape)
print(project_standardized_tr.shape)
print(tfidf_vec_titles_tr.shape)
print(tfidf_vec_essay_tr.shape)
print(tfidf_vec_resource_tr.shape)
```

```
(69918, 2)
(69918, 4)
(69918, 51)
(69918, 9)
```

```
(69918, 9)
(69918, 5)
(69918, 1)
(69918, 1)
(69918, 2463)
(69918, 13888)
(69918, 4639)
```

In [284]:

```
feature_list = feature_list_x.copy()
feature_names = [*feature_list , *tfidf_vec_titles_list, *tfidf_vec_essay_list,
*tfidf_vec_resource_list]
print(len(feature_names))
```

```
21072
```

In [0]:

```
X_tr, X_cr, X_te = hstack_data(tfidf_vec_titles_tr, tfidf_vec_titles_val, tfidf_vec_titles_te, \
                  tfidf_vec_essay_tr, tfidf_vec_essay_val, tfidf_vec_essay_te,\
                  tfidf_vec_resource_tr, tfidf_vec_resource_val, tfidf_vec_resource_te)
```

In [286]:

```
X_tr.shape,X_cr.shape,X_te.shape
```

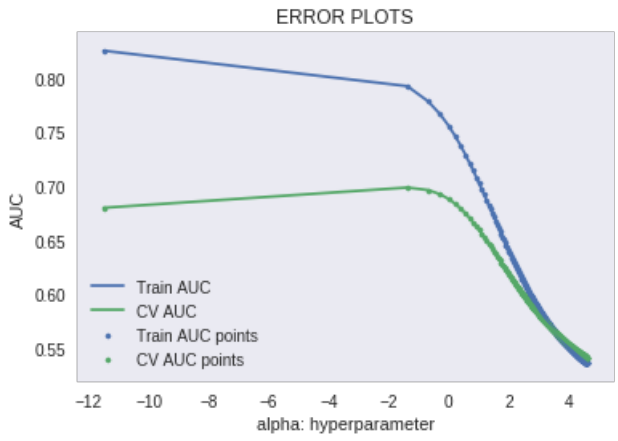
Out[286]:

```
((69918, 21072), (17480, 21072), (21850, 21072))
```

In [287]:

```
best_alpha = optimised_alpha(X_tr,y_train,X_cr,y_val)
```

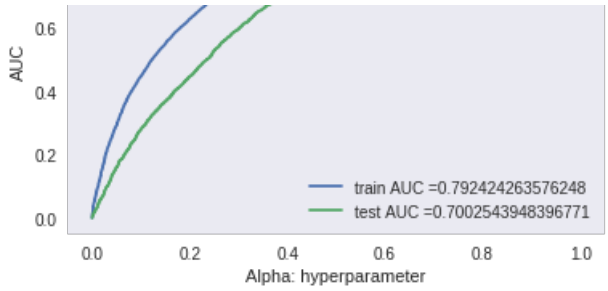
```
100%|██████████| 400/400 [02:35<00:00,  2.49it/s]
```




```
execution time in minutes: 2
execution time in hours: 0
0.6987986715517129
optimal alpha is:  0.25001
```

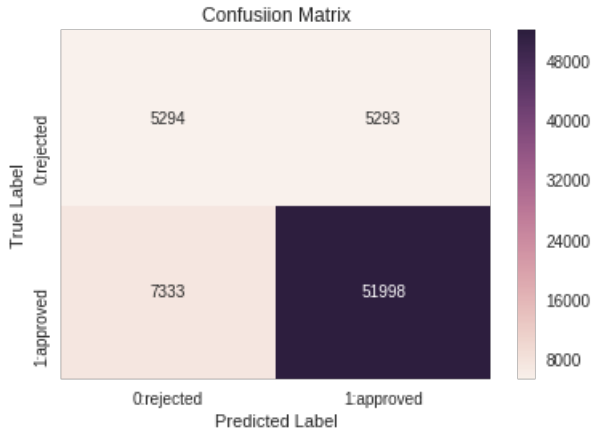
In [288]:

```
error_plot(X_tr,X_te,y_train,y_test)
```

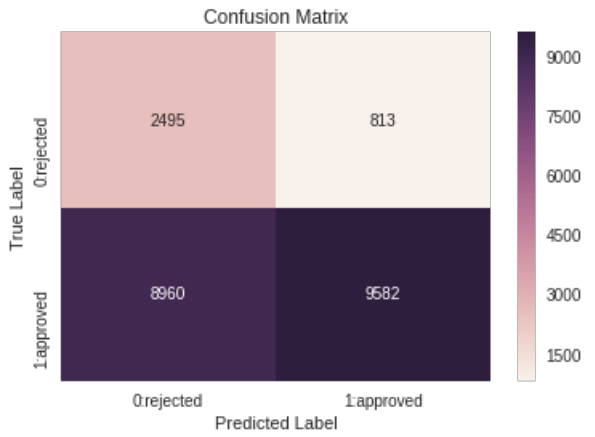


==========

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999776954132 for threshold 0.698
[[ 5294  5293]
 [ 7333 51998]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.914
[[2495  813]
 [8960 9582]]
the maximum value of tpr*(1-fpr) 0.24999999776954132 for threshold 0.698
```




```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.914
```



In [0]:

```
'''n=10
topn_class2 = sorted(zip(mnb.coef_[0], feature_names))[-n:]
print(top_class)
topn_class1,topn_class2'''
```

Out[0]:

```
'n=10\ntopn_class1 = sorted(zip(mnb.coef_[0], feature_names))[:n]\ntopn_class2 =
sorted(zip(mnb.coef_[0], feature_names))[-n:]\nprint(top_class)\ntopn_class1,topn_class2'
```

In [289]:

```
important_features(feature_names,mnb,n=10)
```

```
Top 10 features for negative class
0 -2.985510623418964 price
0 -3.0727144650872535 dig_0
0 -3.8299724701285047 Literacy_Language
0 -3.869387561199721 Grades_PreKto2
0 -3.8912360839638414 Math_Science
0 -4.09466840855228 Grades_3to5
0 -4.390266676896 teacher_number_of_previously_posted_projects
0 -4.803290208277376 Grades_6to8
0 -4.937222905358174 SpecialNeeds
0 -5.0035996307479556 Health_Sports
----------------------------------------
Top 10 features for positive class
1 -3.119213694169858 dig_0
1 -3.2011669014504314 price
1 -3.6726704205508174 Literacy_Language
1 -3.7941425172811787 teacher_number_of_previously_posted_projects
1 -3.860069251447406 Grades_PreKto2
1 -3.935177244903409 Math_Science
1 -4.0227923117329905 Grades_3to5
1 -4.827960989507034 Grades_6to8
1 -4.840325566194412 dig_1
1 -4.896176338900611 CA
```

Using coef_

In [0]:

```
#show_most_informative_features(feature_names, clf, n=10)
```

In [0]:

```
#most_informative_feature_for_binary_classification(feature_names, mnb)
```

Using feature_log*prob*

In [291]:

```
"""features = mnb.feature_count_
print(features.shape)
log_prob = mnb.feature_log_prob_
features_pd = pd.DataFrame(log_prob, columns = feature_names)
features_pd = features_pd.T
print(features_pd.shape)
"""
```

Out[291]:

```
'features = mnb.feature_count_\nprint(features.shape)\nlog_prob =
mnb.feature_log_prob_\nfeatures_pd = pd.DataFrame(log_prob, columns = feature_names)\nfeatures_pd
= features_pd.T\nprint(features_pd.shape)\n'
```

In [292]:

```
"""# Feature Importance
print("Top 10 Negative Features:\n",features_pd[0].sort_values(ascending = False)[0:10])
print("\n\n Top 10 Positive Features:\n",features_pd[1].sort_values(ascending = False)[0:10])"""
```

Out[292]:

```
'# Feature Importance\nprint("Top 10 Negative Features:\n",features_pd[0].sort_values(ascending =
False)[0:10])\nprint("\n\n Top 10 Positive Features:\n",features_pd[1].sort_values(ascending = Fal
se)[0:10])'
```

Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)

In [0]:

```
def bow_vec(preprocessed_data_tr,preprocessed_data_val,preprocessed_data_te):
  global vectorizer_bow
  vectorizer_bow = CountVectorizer(min_df=10)
  text_bow_tr = vectorizer_bow.fit_transform(preprocessed_data_tr)
  print("Shape of matrix after one hot encodig ",text_bow_tr.shape)
  vectorizer_list = vectorizer_bow.get_feature_names()
  text_bow_val = vectorizer_bow.transform(preprocessed_data_val)
  print("Shape of matrix after one hot encodig ",text_bow_val.shape)

  text_bow_te = vectorizer_bow.transform(preprocessed_data_te)
  print("Shape of matrix after one hot encodig ",text_bow_te.shape)
  return text_bow_tr,text_bow_val, text_bow_te, vectorizer_list
```

In [294]:

```
bow_vec_essay_tr,bow_vec_essay_val,bow_vec_essay_te,bow_vec_essay_list = bow_vec(textpreprocessed(
X_train,'essay'),textpreprocessed(X_val,'essay'), textpreprocessed(X_test,'essay'))
bow_vec_titles_tr,bow_vec_titles_val,bow_vec_titles_te,bow_vec_titles_list =
bow_vec(textpreprocessed(X_train,'project_title'),textpreprocessed(X_val,'project_title'),
textpreprocessed(X_test,'project_title'))
bow_vec_resource_tr,bow_vec_resource_val,bow_vec_resource_te,bow_vec_resource_list =
bow_vec(textpreprocessed(X_train,'project_resource_summary'),textpreprocessed(X_val,'project_resour
ce_summary'), textpreprocessed(X_test,'project_resource_summary'))
```

```
100%|██████████| 69918/69918 [00:43<00:00, 1606.37it/s]
100%|██████████| 17480/17480 [00:10<00:00, 1603.54it/s]
100%|██████████| 21850/21850 [00:13<00:00, 1614.11it/s]

Shape of matrix after one hot encodig  (69918, 13888)
Shape of matrix after one hot encodig  (17480, 13888)

  5%|▌         | 3396/69918 [00:00<00:01, 33959.63it/s]

Shape of matrix after one hot encodig  (21850, 13888)

100%|██████████| 69918/69918 [00:02<00:00, 34114.87it/s]
100%|██████████| 17480/17480 [00:00<00:00, 33415.02it/s]
100%|██████████| 21850/21850 [00:00<00:00, 33934.89it/s]

Shape of matrix after one hot encodig  (69918, 2463)
Shape of matrix after one hot encodig  (17480, 2463)

  2%|▏         | 1527/69918 [00:00<00:04, 15257.06it/s]

Shape of matrix after one hot encodig  (21850, 2463)

100%|██████████| 69918/69918 [00:04<00:00, 15120.85it/s]
100%|██████████| 17480/17480 [00:01<00:00, 14938.46it/s]
100%|██████████| 21850/21850 [00:01<00:00, 15016.57it/s]

Shape of matrix after one hot encodig  (69918, 4639)
Shape of matrix after one hot encodig  (17480, 4639)
Shape of matrix after one hot encodig  (21850, 4639)
```

In [295]:

```
feature_list = feature_list_x.copy()
feature_names = [*feature_list , *bow_vec_titles_list, *bow_vec_essay_list, *bow_vec_resource_list]
print(len(feature_names))
```

```
21072
```

In [0]:

```
X_tr, X_cr, X_te = hstack_data(bow_vec_titles_tr, bow_vec_titles_val, bow_vec_titles_te, \
```

```
                    bow_vec_essay_tr, bow_vec_essay_val, bow_vec_essay_te,\
                    bow_vec_resource_tr, bow_vec_resource_val, bow_vec_resource_te)
```

In [297]:

```
X_tr.shape,X_cr.shape,X_te.shape
```

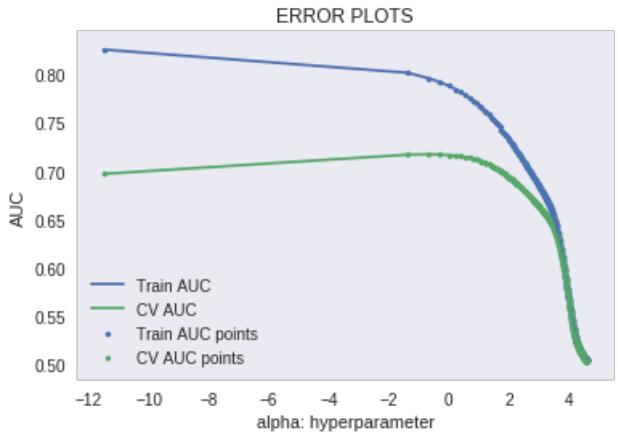
Out[297]:

```
((69918, 21072), (17480, 21072), (21850, 21072))
```

In [298]:

```
best_alpha = optimised_alpha(X_tr,y_train,X_cr,y_val)
```

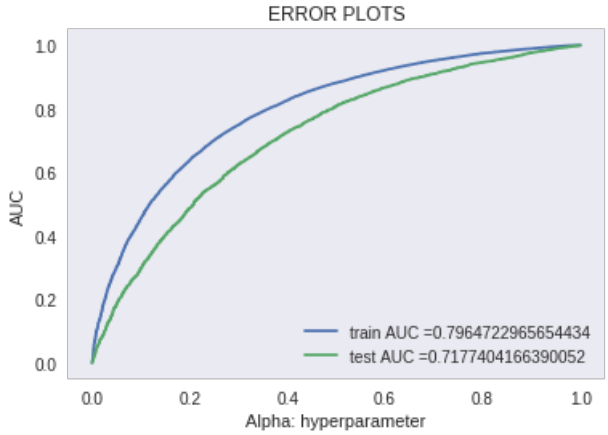
```
100%|██████████| 400/400 [02:32<00:00,  2.68it/s]
```




```
execution time in minutes: 2
execution time in hours: 0
0.7175202648420391
optimal alpha is:  0.50001
```

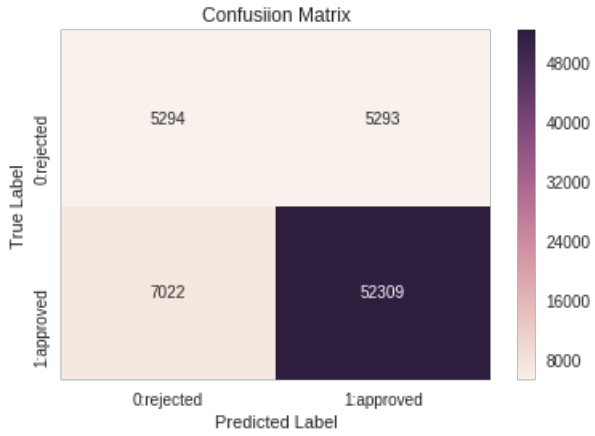
In [299]:

```
error_plot(X_tr,X_te,y_train,y_test)
```

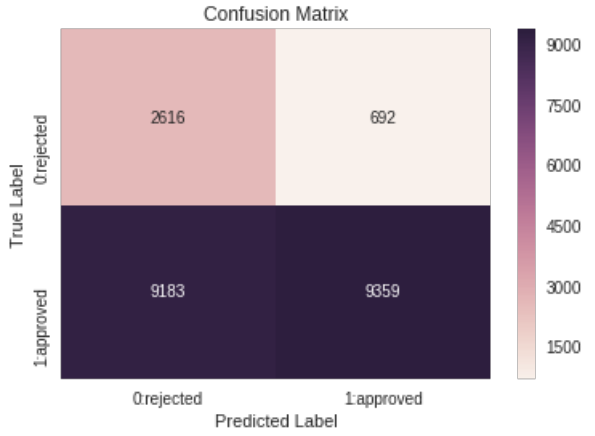



```
====================================================================================================

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999776954132 for threshold 0.067
[[ 5294  5293]
 [ 7022 52309]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.996
[[2616  692]
 [9183 9359]]
the maximum value of tpr*(1-fpr) 0.24999999776954132 for threshold 0.067
```



the maximum value of tpr*(1-fpr) 0.25 for threshold 0.996



In [306]:

```
important_features(feature_names,mnb,n=10)
```

```
Top 10 features for negative class
0 -3.0991711224968803 students
0 -4.187444488377221 school
0 -4.516875651613711 learning
0 -4.663180442844908 classroom
0 -4.867445250334713 not
0 -4.869959240789818 learn
0 -4.906340418103799 help
0 -4.915731350054621 students
0 -4.956147898813974 need
----------------------------------------
Top 10 features for positive class
1 -3.077994176219926 students
1 -4.223236641493973 school
1 -4.587715002132292 learning
1 -4.610239189693134 classroom
1 -4.875833576937367 not
1 -4.9222883245451445 learn
1 -4.9545673486226125 help
1 -4.9805410521511995 students
1 -5.016582264444141 need
1 -5.095443632330669 many
```

Using coef_

In [0]:

```
#show_most_informative_features(feature_names, mnb, n=10)
```

Using feature_log*prob*

In [303]:

```
"""features = mnb.feature_count_
print(features.shape)
log_prob = mnb.feature_log_prob_
features_pd = pd.DataFrame(log_prob, columns = feature_names)
features_pd = features_pd.T
print(features_pd.shape)"""
```

Out[303]:

```
'features = mnb.feature_count_\nprint(features.shape)\nlog_prob =
mnb.feature_log_prob_\nfeatures_pd = pd.DataFrame(log_prob, columns = feature_names)\nfeatures_pd
= features_pd.T\nprint(features_pd.shape)'
```

In [304]:

```
"""# Feature Importance
print("Top 10 Negative Features:\n",features_pd[0].sort_values(ascending = False)[0:10])
print("\n\n Top 10 Positive Features:\n",features_pd[1].sort_values(ascending = False)[0:10])"""
```

Out[304]:

```
'# Feature Importance\nprint("Top 10 Negative Features:\n",features_pd[0].sort_values(ascending =
False)[0:10])\nprint("\n\n Top 10 Positive Features:\n",features_pd[1].sort_values(ascending = Fal
se)[0:10])'
```

Summary

In [307]:

```
from prettytable import PrettyTable
import sys
sys.stdout.write("\033[1;30m")

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyper parameter", "AUC"]

x.add_row(["BOW", 'Brute', 0.50001, 0.7177])
x.add_row(["TFIDF", 'Brute',0.25001, 0.7002])

print(x)
```

```
+------------+-------+-----------------+--------+
| Vectorizer | Model | Hyper parameter |  AUC   |
+------------+-------+-----------------+--------+
|    BOW     | Brute |     0.50001     | 0.7177 |
|   TFIDF    | Brute |     0.25001     | 0.7002 |
+------------+-------+-----------------+--------+
```