# Naive Bayes on DonorsChoose

DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need of funding. Right now, a large number of volunteers is needed to manually screen each submission before it's approved to be posted on the DonorsChoose.org website. Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result, there are three main problems they need to solve: How to scale current manual processes and resources to screen 500,000 projects so that they can be posted as quickly and as efficiently as possible How to increase the consistency of project vetting across different volunteers to improve the experience for teachers How to focus volunteer time on the applications that need the most assistance

The goal of the competition is to predict whether or not a DonorsChoose.org project proposal submitted by a teacher will be approved, using the text of project descriptions as well as additional metadata about the project, teacher, and school. DonorsChoose.org can then use this information to identify projects most likely to need further review before approval.

In [1]:

```
%matplotlib inline
import warnings
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.svm import SVC
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
from sklearn.model_selection import GridSearchCV

import warnings

warnings.filterwarnings(
    'ignore',
    'detected Windows; aliasing chunkize to chunkize_serial',
)
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

import warnings
warnings.filterwarnings(action='ignore', category=UserWarning, module='gensim')
import gensim

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
```

# 1.LOAD AND PROCESS DATA

In [2]:

```
data  = pd.read_csv('train_data.csv', nrows=50000)
resource_data=pd.read_csv('resources.csv')
data.head(5)
data.shape
```

Out[2]:

```
(50000, 17)
```

## 1.1 Merging resourse and project data

In [3]:

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

Out[3]:

| | id | price | quantity |
|---|---|---|---|
| **0** | p000001 | 459.56 | 7 |
| **1** | p000002 | 515.89 | 21 |

In [4]:

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

## 1.2 process Project Essay

In [5]:

```
data["essay"] = data["project_essay_1"].map(str) +\
                data["project_essay_2"].map(str) + \
                data["project_essay_3"].map(str) + \
                data["project_essay_4"].map(str)
```

In [6]:

```
# 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 [7]:

```
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', 't
heir', \
```

```
heir',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these',
'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'd
o', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'whil
e', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'bef
ore', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'a
gain', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each
', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', '
m', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn
't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', '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 [8]:

```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(data['essay'].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 not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
data['cleaned_essay']=preprocessed_essays
```

```
100%|██████████| 50000/50000 [00:23<00:00, 2090.56it/s]
```

## 1.3 Project Title

In [9]:

```
# https://stackoverflow.com/a/47091490/4084039
from tqdm import tqdm
preprocessed_title = []
# tqdm is for printing the status bar
for sentance in tqdm(data['project_title'].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 not in stopwords)
    preprocessed_title.append(sent.lower().strip())
data['cleaned_project_title']=preprocessed_title
```

```
100%|██████████| 50000/50000 [00:01<00:00, 40523.53it/s]
```

In [10]:

```
data.head(2)
```

Out[10]:

| | Unnamed: 0 | id | teacher_id | teacher_prefix | school_state | project_submitted_datetime | project_grade_c |
|---|---|---|---|---|---|---|---|
| **0** | 160221 | p253737 | c90749f5d961ff158d4b4d1e7dc665fc | Mrs. | IN | 2016-12-05 13:43:57 | Grades |
| **1** | 140945 | p258326 | 897464ce9ddc600bced1151f324dd63a | Mr. | FL | 2016-10-25 09:22:10 | Gra |

2 rows × 22 columns

## 1.4 teacher_prefix

In [11]:

```
#https://stackoverflow.com/questions/9452108/how-to-use-string-replace-in-python-3-x
temp1=data.teacher_prefix.apply(lambda x: str(x).replace('.', ''))
data['teacher_prefix']=temp1
data['teacher_prefix'].value_counts()
```

Out[11]:

```
Mrs        26140
Ms         17936
Mr          4859
Teacher     1061
Dr             2
nan            2
Name: teacher_prefix, dtype: int64
```

## 1.5 project grade

In [12]:

```
data['project_grade_category'].value_counts()
```

Out[12]:

```
Grades PreK-2    20316
Grades 3-5       16968
Grades 6-8        7750
Grades 9-12       4966
Name: project_grade_category, dtype: int64
```

In [13]:

```
#https://stackoverflow.com/questions/9452108/how-to-use-string-replace-in-python-3-x
grade_list = []
for i in data['project_grade_category'].values:
     i=i.replace(' ','_')
     i=i.replace('-','_')
     grade_list.append(i.strip())


data['project_grade_category']=grade_list
```

In [14]:

```
data['project_grade_category'].value_counts()
```

Out[14]:

```
Grades_PreK_2    20316
Grades_3_5       16968
Grades_6_8        7750
Grades_9_12       4966
Name: project_grade_category, dtype: int64
```

### 1.6 Making dependant(label) and independant variables

In [15]:

```
y = data['project_is_approved'].values
data.drop(['project_is_approved'], axis=1, inplace=True)
data.head(1)
x=data
```

### 1.7 Traing and Test split

In [16]:

```
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,random_state=42)
X_train, X_cv, Y_train, Y_cv = train_test_split(X_train, Y_train, test_size=0.33, stratify=Y_train,rand
om_state=42)
```

# 2.TEXT VECTORIZATION AND ENCODING OF CATEGORIES

### 2.1 converting the essay to vectors using BOW

In [17]:

```
print(X_train.shape, Y_train.shape)
print(X_cv.shape, Y_cv.shape)
print(X_test.shape, Y_test.shape)

print("="*100)



from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['cleaned_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 = vectorizer.transform(X_train['cleaned_essay'].values)
X_cv_essay_bow = vectorizer.transform(X_cv['cleaned_essay'].values)
X_test_essay_bow = vectorizer.transform(X_test['cleaned_essay'].values)
essay_bow_vect=vectorizer.get_feature_names()
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, 21) (22445,)
(11055, 21) (11055,)
(16500, 21) (16500,)
====================================================================================================
After vectorizations
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
====================================================================================================
```

### 2.2 converting the title to vectors using BOW

## 2.2 converting the title to vectors using BOW

In [18]:

```
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['cleaned_project_title'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_bow = vectorizer.transform(X_train['cleaned_project_title'].values)
X_cv_title_bow = vectorizer.transform(X_cv['cleaned_project_title'].values)
X_test_title_bow = vectorizer.transform(X_test['cleaned_project_title'].values)
title_bow_vect=vectorizer.get_feature_names()
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)
```

```
After vectorizations
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
====================================================================================================
```

## 2.3 converting the essay to vectors using TFIDF

In [19]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['cleaned_essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer.transform(X_train['cleaned_essay'].values)
X_cv_essay_tfidf = vectorizer.transform(X_cv['cleaned_essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['cleaned_essay'].values)
essay_tfidf_vect=vectorizer.get_feature_names()
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)
```

```
After vectorizations
(22445, 8869) (22445,)
(11055, 8869) (11055,)
(16500, 8869) (16500,)
====================================================================================================
```

## 2.4 converting the title to vectors using TFIDF

In [20]:

```
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['cleaned_project_title'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_tfidf = vectorizer.transform(X_train['cleaned_project_title'].values)
X_cv_title_tfidf = vectorizer.transform(X_cv['cleaned_project_title'].values)
X_test_title_tfidf = vectorizer.transform(X_test['cleaned_project_title'].values)
title_tfidf_vect=vectorizer.get_feature_names()
print("After vectorizations")
print(X_train_title_tfidf.shape, Y_train.shape)
print(X_cv_title_tfidf.shape, Y_cv.shape)
print(X_test_title_tfidf.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1229) (22445,)
(11055, 1229) (11055,)
```

```
(16500, 1229) (16500,)
====================================================================================================
```

### 2.5 one hot encoding the catogorical features: teacher_prefix

In [21]:

```
vectorizer = CountVectorizer()
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 = vectorizer.transform(X_train['teacher_prefix'].values)
X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)
teacher_prefix_vect=vectorizer.get_feature_names()
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(vectorizer.get_feature_names())
print("="*100)
```

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

### 2.6 one hot encoding the catogorical features: project Grade

In [22]:

```
vectorizer = CountVectorizer()
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 = vectorizer.transform(X_train['project_grade_category'].values)
X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].values)
project_grade_vect=vectorizer.get_feature_names()
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(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']
====================================================================================================
```

### 2.7 one hot encoding the catogorical features: state

In [23]:

```
vectorizer = CountVectorizer()
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 = vectorizer.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)
state_vect=vectorizer.get_feature_names()

print("After vectorizations")
```

```
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(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']
====================================================================================================
```

### 2.8 Normalizing the numerical features: Price

In [24]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

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

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

# 3. APPLY NAIVE BAYES ON BOW

### 3.1 BOW:Concatinating all the features

In [25]:

```
from scipy.sparse import hstack
X_tr_bow = hstack((X_train_essay_bow,X_train_title_bow, X_train_state_ohe, X_train_teacher_ohe, X_train
_grade_ohe, X_train_price_norm)).tocsr()
X_cr_bow = hstack((X_cv_essay_bow,X_cv_title_bow, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_c
v_price_norm)).tocsr()
X_te_bow = hstack((X_test_essay_bow,X_test_title_bow, X_test_state_ohe, X_test_teacher_ohe, X_test_grad
e_ohe, X_test_price_norm)).tocsr()

print("Final Data matrix")
print(X_tr_bow.shape, Y_train.shape)
print(X_cr_bow.shape, Y_cv.shape)
print(X_te_bow.shape, Y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 7065) (22445,)
(11055, 7065) (11055,)
(16500, 7065) (16500,)
====================================================================================================
```

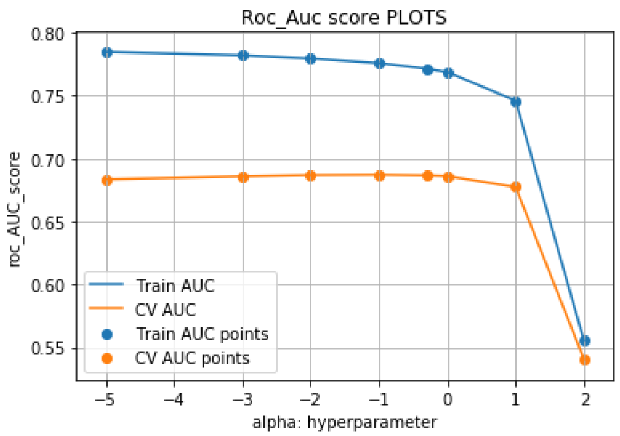
### 3.2 Hyper parameter Tuning:simple for loop for Train and cross validation

In [26]:

```
#https://stackabuse.com/understanding-roc-curves-with-python/
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
    # not the predicted outputs

    data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 5000, then your cr_loop will be 4998- 4998%100 = 4900
    # in this for loop we will iterate unti the last 100 multiplier
    for i in range(0, tr_loop, 1000):
        data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return data_pred
```

In [27]:

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

train_auc = []
cv_auc = []

alpha_values=[0.00001,0.001,0.01,0.1,0.5,1,10,100]

for alpha in alpha_values:
    model_bow = MultinomialNB(alpha = alpha)
    model_bow.fit(X_tr_bow,Y_train)
    y_tr_prob = batch_predict(model_bow,X_tr_bow)
    y_cr_prob =batch_predict(model_bow, X_cr_bow)

    train_auc.append(roc_auc_score(Y_train,y_tr_prob))
    cv_auc.append(roc_auc_score(Y_cv,y_cr_prob))


plt.plot(np.log10(alpha_values), train_auc, label='Train AUC')
plt.plot(np.log10(alpha_values), cv_auc, label='CV AUC')

plt.scatter(np.log10(alpha_values), train_auc, label='Train AUC points')
plt.scatter(np.log10(alpha_values), cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("roc_AUC_score")
plt.title("Roc_Auc score PLOTS")
plt.grid()
plt.show()
```



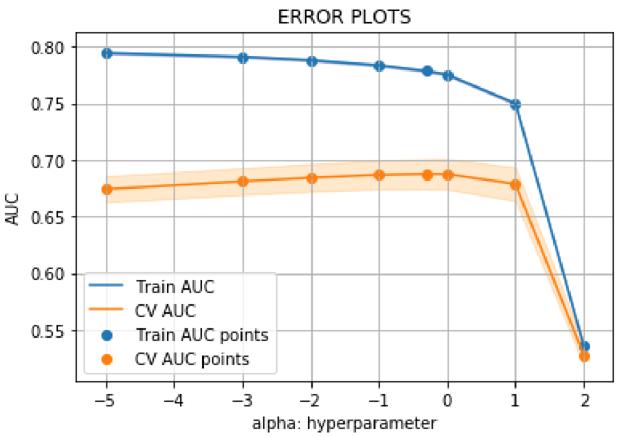
## observations

1.By observing plot of auc score of each aplha for train and cross validation we understand alpha=0.1 is best hyperparameter as cross validation auc is high at this value and it not leads to overfitting ang underfitting problem

## 3.3 Method 2: GridSearch

In [28]:

```
### from sklearn.model_selection import GridSearchCV


#parameters = {'alpha':[1, 5, 10, 15, 21, 31, 41, 51,101]}
parameters = {'alpha':[0.00001,0.001,0.01,0.1,0.5,1,10,100]}
model_bow = MultinomialNB(alpha = alpha)
clf = GridSearchCV(model_bow, parameters,scoring='roc_auc',cv=10,return_train_score=True)
clf.fit(X_tr_bow, Y_train)


train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']


plt.plot(np.log10(parameters['alpha']), train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(np.log10(parameters['alpha']),train_auc - train_auc_std,train_auc + train_auc_st
d,alpha=0.2,color='darkblue')

plt.plot(np.log10(parameters['alpha']), cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(np.log10(parameters['alpha']),cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,
color='darkorange')

plt.scatter(np.log10(parameters['alpha']), train_auc, label='Train AUC points')
plt.scatter(np.log10(parameters['alpha']), cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



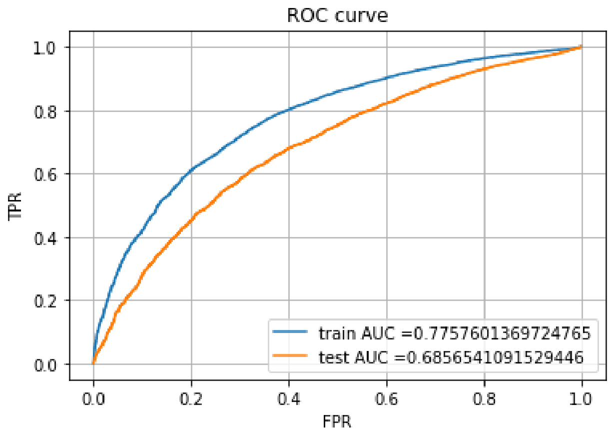

## observations

1.By observing plot of auc score of each aplha for train and cross validation we understand alpha=0.1 is best hyperparameter as cross validation auc is high at this value and it not leads to overfitting ang underfitting problem

### 3.4 ROC curve with best K

In [29]:

```
from sklearn.metrics import roc_curve, auc
alpha_bow=0.1 # for value 31 it test AUC is 56%


model_bow = MultinomialNB(alpha = alpha_bow)
model_bow.fit(X_tr_bow, Y_train)


y_train_pred = batch_predict(model_bow, X_tr_bow)
y_test_pred = batch_predict(model_bow, X_te_bow)

train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, y_test_pred)

auc_bow=auc(test_fpr, test_tpr)
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC curve")
plt.grid()
plt.show()
```




### observations

1.By looking ROC curve of Training FPR and TPR it looks sensible as it is greater than diagonal line or 0.5

2.ROC curve of Test FPR and TPR is sensible as it is greater than o.5.This model perform good,it is generalized model.

### 3.5 confusion matrix

In [30]:

```
# https://tatwan.github.io/How-To-Plot-A-Confusion-Matrix-In-Python/
def myplot_matrix1(data):
    plt.clf()
    plt.imshow(data, interpolation='nearest', cmap=plt.cm.Wistia)
    classNames = ['Negative','Positive']
    plt.title('Approved not approved matrix')
    tick_marks = np.arange(len(classNames))

    plt.xticks(tick_marks, classNames, rotation=45)
    plt.yticks(tick_marks, classNames)
    s = [['TN','FN'], ['FP', 'TP']]
    for i in range(2):
        for j in range(2):
            plt.text(j,i, str(s[i][j])+" = "+str(data[i][j]))
```

```
    plt.show()
```

In [31]:

```
# 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(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))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [32]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

y_train_predicted_withthroshold=predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)
y_test_predicted_withthroshold=predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)

cm_train=confusion_matrix(Y_train,y_train_predicted_withthroshold,labels=[0, 1])



print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(cm_train)
print("="*100)
print("Accuracy score  for Train")
print(accuracy_score(Y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)))
print("="*100)

cm_test=confusion_matrix(Y_test,y_test_predicted_withthroshold,labels=[0, 1])

print("Test confusion matrix")
print(cm_test)
print("="*100)
print("Accuracy score  for Test")
print(accuracy_score(Y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)))
print("="*100)
```

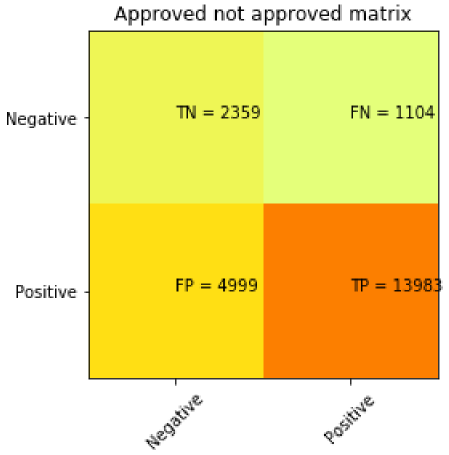
```
====================================================================================================
Train confusion matrix
[[ 2359  1104]
 [ 4999 13983]]
====================================================================================================
Accuracy score  for Train
0.7280908888393851
====================================================================================================
Test confusion matrix
[[1720  826]
 [5444 8510]]
====================================================================================================
Accuracy score  for Test
0.62
====================================================================================================
```

In [33]:

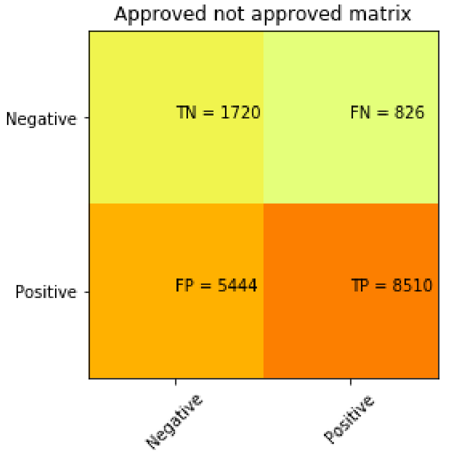
```
print("confusion matrix for train data")
print("="*100)
```

```
myplot_matrix1(cm_train)
print("confusion matrix for Test data")

print("="*100)
myplot_matrix1(cm_test)
```

confusion matrix for train data
====================================================================================================




confusion matrix for Test data
====================================================================================================




### observations

1.TN and TP of test data is high.

2.Model perform good on train data than test data.

3.Accuracy score on train data is 70% and test data is 59%.

4.TPR rate of test data is 91% .FPR rate of test data is 76%.TPR rate of test data is more than FPR rate of test data

5.TNR rate of testdata is 23% .FNR of test data is 8%.TNR rate of test data is more than FNR rate of test data.

6.TPR and TNR is higher than FPR and FNR so model is sensible for alpha=0.1

### 3.6 Feature importance on BOW

In [34]:

```
feature_names=[]

feature_names.extend(essay_bow_vect)
feature_names.extend(title_bow_vect)
feature_names.extend(state_vect)
feature_names.extend(teacher_prefix_vect)
feature_names.extend(project_grade_vect)
feature_names.extend(['price'])
print("lenghth of feature_names matches dimensions of hstack")

print(len(feature_names))

print("="*100)

print("Top 10 negative features are given below")
print("="*100)
max_ind_neg=np.argsort((model_bow.feature_log_prob_)[0])[::-1][0:10]
top_neg=np.take(feature_names,max_ind_neg)
print(top_neg)
print("="*100)
print("Top 10 positive features are given below")
print("="*100)
max_ind_pos=np.argsort((model_bow.feature_log_prob_)[1])[::-1][0:10]
top_pos=np.take(feature_names,max_ind_pos)
print(top_pos)
```

```
lenghth of feature_names matches dimensions of hstack
7065
====================================================================================================
Top 10 negative features are given below
====================================================================================================
['students' 'school' 'learning' 'my' 'classroom' 'not' 'they' 'help'
 'learn' 'the']
====================================================================================================
Top 10 positive features are given below
====================================================================================================
['students' 'school' 'my' 'learning' 'classroom' 'the' 'they' 'not'
 'learn' 'my students']
```

# 4. APPLY NAIVE BAYES ON TFIDF

### 4.1 Tfidf:Concatinating all the features

In [35]:

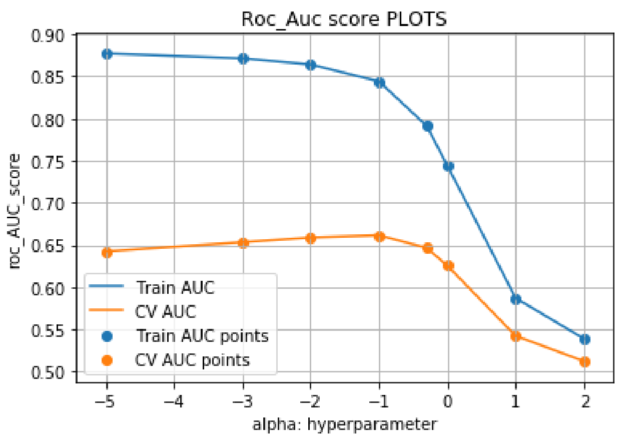
```
from scipy.sparse import hstack
X_tr_tfidf = hstack((X_train_essay_tfidf,X_train_title_tfidf, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_price_norm)).tocsr()
X_cr_tfidf = hstack((X_cv_essay_tfidf,X_cv_title_tfidf, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_price_norm)).tocsr()
X_te_tfidf = hstack((X_test_essay_tfidf,X_test_title_tfidf, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_price_norm)).tocsr()

print("Final Data matrix")
print(X_tr_tfidf.shape, Y_train.shape)
print(X_cr_tfidf.shape, Y_cv.shape)
print(X_te_tfidf.shape, Y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 10159) (22445,)
(11055, 10159) (11055,)
(16500, 10159) (16500,)
====================================================================================================
```

### 4.2 Hyper parameter Tuning:simple for loop for Train and cross validation

In [36]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []


alpha_values=[0.00001,0.001,0.01,0.1,0.5,1,10,100]

for alpha in alpha_values:
    model_tfidf = MultinomialNB(alpha = alpha)
    model_tfidf.fit(X_tr_tfidf,Y_train)
    y_tr_prob = batch_predict(model_tfidf,X_tr_tfidf)
    y_cr_prob =batch_predict(model_tfidf, X_cr_tfidf)

    train_auc.append(roc_auc_score(Y_train,y_tr_prob))
    cv_auc.append(roc_auc_score(Y_cv,y_cr_prob))


plt.plot(np.log10(alpha_values), train_auc, label='Train AUC')
plt.plot(np.log10(alpha_values), cv_auc, label='CV AUC')

plt.scatter(np.log10(alpha_values), train_auc, label='Train AUC points')
plt.scatter(np.log10(alpha_values), cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("roc_AUC_score")
plt.title("Roc_Auc score PLOTS")
plt.grid()
plt.show()
```




## observations

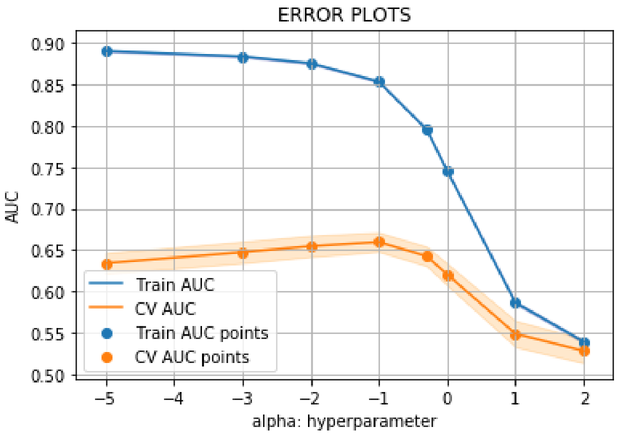
1.By observing plot of auc score of each aplha for train and cross validation we understand alpha=0.1 is best hyperparameter as cross validation auc is high at this value and it not leads to overfitting ang underfitting problem

## 4.3 Method 2: GridSearch

In [37]:

```
### from sklearn.model_selection import GridSearchCV

parameters = {'alpha':[0.00001,0.001,0.01,0.1,0.5,1,10,100]}
model_tfidf = MultinomialNB(alpha = alpha)
clf = GridSearchCV(model_tfidf, parameters,scoring='roc_auc',cv=10,return_train_score=True)
clf.fit(X_tr_tfidf, Y_train)


train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

```
plt.plot(np.log10(parameters['alpha']), train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(np.log10(parameters['alpha']),train_auc - train_auc_std,train_auc + train_auc_st
d,alpha=0.2,color='darkblue')

plt.plot(np.log10(parameters['alpha']), cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(np.log10(parameters['alpha']),cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,
color='darkorange')

plt.scatter(np.log10(parameters['alpha']), train_auc, label='Train AUC points')
plt.scatter(np.log10(parameters['alpha']), cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



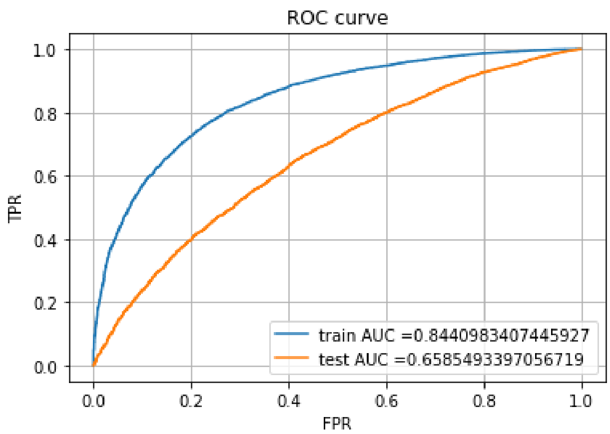

## observations

1.By observing plot of auc score of each aplha for train and cross validation we understand alpha=0.1 is best hyperparameter as cross validation auc is high at this value and it not leads to overfitting ang underfitting problem

## 4.4 ROC curve with best K

In [38]:

```
from sklearn.metrics import roc_curve, auc
alpha_tfidf=0.1 # for all value AUC is 49

model_tfidf = MultinomialNB(alpha = alpha_tfidf)
model_tfidf.fit(X_tr_tfidf, Y_train)


y_train_pred = batch_predict(model_tfidf, X_tr_tfidf)
y_test_pred = batch_predict(model_tfidf, X_te_tfidf)

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)

auc_tfidf=auc(test_fpr, test_tpr)
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC curve")
plt.grid()
plt.show()
```




## observations

1.By looking ROC curve of Training FPR and TPR it looks sensible as it is greater than diagonal line or 0.5

2.ROC curve of Test FPR and TPR is sensible as it is greater than o.5.This model perform good,it is generalized model.

## 4.5 confusion matrix

In [39]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

y_train_predicted_withthroshold=predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)
y_test_predicted_withthroshold=predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)

cm_train=confusion_matrix(Y_train,y_train_predicted_withthroshold,labels=[0, 1])



print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(cm_train)
print("="*100)
print("Accuracy score  for Train")
print(accuracy_score(Y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)))
print("="*100)

cm_test=confusion_matrix(Y_test,y_test_predicted_withthroshold,labels=[0, 1])

print("Test confusion matrix")
print(cm_test)
print("="*100)
print("Accuracy score  for Test")
print(accuracy_score(Y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)))
print("="*100)
```

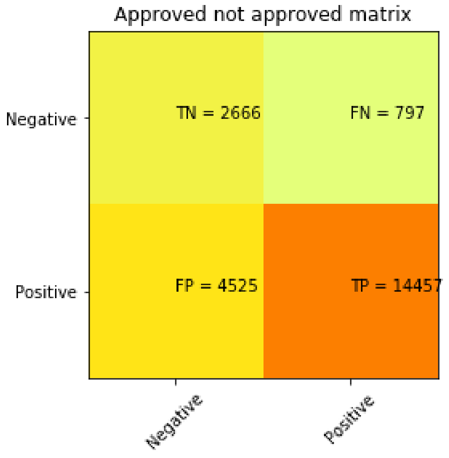
```
====================================================================================================
Train confusion matrix
[[ 2666   797]
 [ 4525 14457]]
====================================================================================================
Accuracy score  for Train
0.7628870572510581
====================================================================================================
Test confusion matrix
[[ 1097  1449]
 [ 3138 10816]]
====================================================================================================
Accuracy score  for Test
0.722
```

====================================================================================================

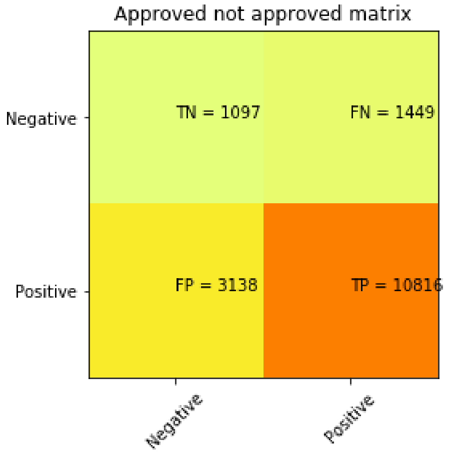
In [40]:

```
print("confusion matrix for train data")
print("="*100)
myplot_matrix1(cm_train)
print("confusion matrix for Test data")

print("="*100)
myplot_matrix1(cm_test)
```

confusion matrix for train data
====================================================================================================




confusion matrix for Test data
====================================================================================================




## observations

1.TN of test data is high than FN rate

2.Model perform good on train data than test data.

3.Accuracy score on train data is54% and test data is 47%

4.TPR rate of test data is 87%.FPR rate of test data is 82%.TPR rate of test data is more than FPR rate of test data

5.TNR rate of testdata is 17%.FNR of test data is 12%.TNR rate of test data is more than FNR rate of test data.

6.TPR and TNR is higher than FPR and FNR so model is sensible for alpha=0.1

## 4.6 Feature importance

In [41]:

```
feature_names_tfidf=[]

feature_names_tfidf.extend(essay_tfidf_vect)
feature_names_tfidf.extend(title_tfidf_vect)
feature_names_tfidf.extend(state_vect)
feature_names_tfidf.extend(teacher_prefix_vect)
feature_names_tfidf.extend(project_grade_vect)
feature_names_tfidf.extend(['price'])
print("lenghth of feature_names matches dimensions of hstack")

print(len(feature_names_tfidf))

print("="*100)

print("Top 10 negative features are given below")
print("="*100)
max_ind_neg=np.argsort((model_tfidf.feature_log_prob_)[0])[::-1][0:10]
top_neg=np.take(feature_names_tfidf,max_ind_neg)
print(top_neg)
print("="*100)
print("Top 10 positive features are given below")
print("="*100)
max_ind_pos=np.argsort((model_tfidf.feature_log_prob_)[1])[::-1][0:10]
top_pos=np.take(feature_names_tfidf,max_ind_pos)
print(top_pos)
```

```
lenghth of feature_names matches dimensions of hstack
10159
====================================================================================================
Top 10 negative features are given below
====================================================================================================
['price' 'mrs' 'grades_prek_2' 'ms' 'grades_3_5' 'grades_6_8' 'ca'
 'students' 'grades_9_12' 'mr']
====================================================================================================
Top 10 positive features are given below
====================================================================================================
['price' 'mrs' 'grades_prek_2' 'ms' 'grades_3_5' 'grades_6_8' 'ca'
 'students' 'grades_9_12' 'mr']
```

# 5.MODEL PERFORMANCE TABLE

## 5.1 performance table of BOW and TFIDF

In [42]:

```
from prettytable import PrettyTable
x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Hyper Parameter(k)", "AUC"]
x.add_row(["Bow", "Brute", alpha_bow,auc_bow])
x.add_row(["TFIDF","Brute",alpha_tfidf,auc_tfidf])

print(x)
```

```
+------------+-------+--------------------+--------------------+
| Vectorizer | Model | Hyper Parameter(k) |        AUC         |
+------------+-------+--------------------+--------------------+
|    Bow     | Brute |        0.1         | 0.6856541091529446 |
|   TFIDF    | Brute |        0.1         | 0.6585493397056719 |
+------------+-------+--------------------+--------------------+
```

# 6.SUMMARY

1.Naive bayes is much better model than Knn model for Text classification for donors choose dataset.

2.Naive Bayes with Bow model has more accuracy than TfIdf model here.

3.AUC is high for BOW than tfidf with same optimal value of alpha=0.1