

```
In [0]:
```

```
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 [0]:
```

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

```
/content/drive/My Drive/Assignments_DonorsChoose_2018
```

```
In [0]:
```

```
%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  
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
```

```

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]:

```

categories = 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 categories:
    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 category based on space "Math & Science"=> "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_categories = 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_categories:
    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 category based on space "Math & Science"=> "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
oesn'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 [0]:

```
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[0]:

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

In [0]:

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

In [0]:

```
from sklearn.neighbors import KNeighborsClassifier
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=0.3
0)
```

In [0]:

```
print("Train:",X_train.shape,len(y_train))
print("CV:",X_val.shape,len(y_val))
print("Test:",X_test.shape,len(y_test))
```

```
Train: (61178, 20) 61178
CV: (26220, 20) 26220
Test: (21850, 20) 21850
```

Function for standardizing/normalizing the data

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.preprocessing import Normalizer
def normalize_data(df,column_data):
    normalizer_vec = Normalizer()

    normalizer_vec.fit(df_tr[column_name].values.reshape(-1,1))
    normalizer_data_train = normalizer_vec.transform(df_tr[column_name].values.reshape(-1,1))
    print(normalizer_data_train.shape)

    normalizer_data_traincv = normalizer_vec.transform(df_cv[column_name].values.reshape(-1,1))
    print(normalizer_data_traincv.shape)

    normalizer_data_test = normalizer_vec.transform(df_te[column_name].values.reshape(-1,1))
    print(normalizer_data_test.shape)

    return normalizer_data_train, normalizer_data_traincv, normalizer_data_test
```

Function for vectorizing the text data

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)

    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
```

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('\\n', ' ')
        sent = sent.replace('\\t', ' ')
        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]:

```

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')

```

```

(61178, 1)
(26220, 1)
(21850, 1)

(61178, 1)
(26220, 1)
(21850, 1)

```

In [0]:

```

cat_one_hot_tr,cat_one_hot_val,cat_one_hot_te = 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 =
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 (61178, 9)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding (26220, 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 (61178, 9)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding (26220, 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)

```

Shape of matrix after one hot encoding (61178, 51)

In [0]:

```
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 =
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 =
vectorized_data(X_train,X_val,X_test,'teacher_prefix',vocab=teacher_prefix_dict)

['VT', 'WY', 'ND', 'MT', 'RI', 'AK', 'NE', 'SD', 'DE', 'NH', 'HI', 'WV', 'DC', 'ME', 'NM', 'KS', 'IA',
 'ID', 'AR', 'CO', 'MN', 'OR', 'MS', 'KY', 'NV', 'MD', 'TN', 'UT', 'CT', 'AL', 'WI', 'VA', 'AZ',
 'OK', 'NJ', 'WA', 'MA', 'LA', 'OH', 'MO', 'IN', 'PA', 'MI', 'GA', 'SC', 'IL', 'NC', 'FL', 'NY', 'TX',
 'CA']
Shape of matrix after one hot encoding (61178, 51)
['VT', 'WY', 'ND', 'MT', 'RI', 'AK', 'NE', 'SD', 'DE', 'NH', 'HI', 'WV', 'DC', 'ME', 'NM', 'KS', 'IA',
 'ID', 'AR', 'CO', 'MN', 'OR', 'MS', 'KY', 'NV', 'MD', 'TN', 'UT', 'CT', 'AL', 'WI', 'VA', 'AZ',
 'OK', 'NJ', 'WA', 'MA', 'LA', 'OH', 'MO', 'IN', 'PA', 'MI', 'GA', 'SC', 'IL', 'NC', 'FL', 'NY', 'TX',
 'CA']
Shape of matrix after one hot encoding (26220, 51)
['VT', 'WY', 'ND', 'MT', 'RI', 'AK', 'NE', 'SD', 'DE', 'NH', 'HI', 'WV', 'DC', 'ME', 'NM', 'KS', 'IA',
 'ID', 'AR', 'CO', 'MN', 'OR', 'MS', 'KY', 'NV', 'MD', 'TN', 'UT', 'CT', 'AL', 'WI', 'VA', 'AZ',
 'OK', 'NJ', 'WA', 'MA', 'LA', '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 (61178, 5)
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
Shape of matrix after one hot encoding (26220, 5)
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
Shape of matrix after one hot encoding (21850, 5)
```

In [0]:

```
grade_dict = create_dict(X_train,'project_grade_category')
grade_one_hot_tr,grade_one_hot_val,grade_one_hot_te =
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 (61178, 4)
['Grades_9to12', 'Grades_6to8', 'Grades_3to5', 'Grades_PreKto2']
Shape of matrix after one hot encoding (26220, 4)
['Grades_9to12', 'Grades_6to8', 'Grades_3to5', 'Grades_PreKto2']
Shape of matrix after one hot encoding (21850, 4)
```

In [0]:

```
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 (61178, 2)
Shape of matrix after one hot encoding (21850, 2)
Shape of matrix after one hot encoding (26220, 2)
```

In [0]:

```
one_hot_num_dig_te.shape, grade_one_hot_te.shape \
, state_one_hot_te.shape, cat_one_hot_te.shape, \
cat_sub_one_hot_te.shape, teacher_one_hot_te.shape, \
price_standardized_te.shape, project_standardized_te.shape
```

Out[0]:

```
((21850, 2),
 (21850, 4),
 (21850, 51),
 (21850, 9),
 (21850, 9),
 (21850, 5),
```

```
(21850, 1),  
(21850, 1))
```

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,teacher_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,teacher_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
```

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

In [0]:

```
def batch_predict(clf, data):  
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class  
    # not the predicted outputs  
  
    y_data_pred = []  
    tr_loop = data.shape[0] - data.shape[0]//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 until 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]:

```
#%cd -
```

In [0]:

```
import matplotlib.pyplot as plt  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.metrics import roc_auc_score  
import time  
#f = open("myfile.txt", "a")  
def optimised_k(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 = []  
    for i in range(1, 101, 1):  
        k = i  
        knn = KNeighborsClassifier(n_neighbors=k)  
        knn.fit(X_tr, y_train)  
        y_pred = knn.predict(X_cr)  
        y_score = knn.predict_proba(X_cr)[:, 1]  
        auc = roc_auc_score(y_val, y_score)  
        train_auc.append(auc)  
        cv_auc.append(auc)  
    plt.plot(range(1, 101), train_auc, label='Training AUC')  
    plt.plot(range(1, 101), cv_auc, label='Cross-validation AUC')  
    plt.xlabel('Number of Neighbors (k)')  
    plt.ylabel('AUC')  
    plt.title('KNN Performance vs Number of Neighbors')  
    plt.legend()  
    plt.show()
```

```

K = [1,15,51, 101, 151, 201, 251, 301, 401]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm = "brute")
    neigh.fit(X_tr, y_train)

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

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive 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))
    #f.write('> %d\n' % (i,))
    #f.write('> %s\n' % (roc_auc_score(y_train,y_train_pred),))
    #f.write('%s\n' % (roc_auc_score(y_val, y_cv_pred),))

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

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

plt.legend()
plt.xlabel("K: 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:",float(minutes/60))
print(max(cv_auc))
optimalK = K[np.argmax(cv_auc)]
print("optimal K is: ",optimalK)
return optimalK

```

In [0]:

```

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

    t = threshold[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):
    neigh = KNeighborsClassifier(n_neighbors=best_k,algorithm = "brute")
    neigh.fit(X_train, 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 = batch_predict(neigh, X_train)
    y_test_pred = batch_predict(neigh, 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("K: 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(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("Confusion 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()

```

Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_essay (AVG W2V)

In [0]:

```

#%cd Assignments_DonorsChoose_2018
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-sa
ve-and-load-variables-in-python/
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words = set(model.keys())

```

In [0]:

```
#%cd -
```

In [0]:

```

def avg_w2v(preprocessed_list):
    # average Word2Vec
    preprocessed_list = preprocessed_list[:]
    # compute average word2vec for each review.
    avg_w2v_vectors_list = [] # the avg-w2v for each sentence/review is stored in this list
    for sentence in tqdm(preprocessed_list): # for each review/sentence
        vector = np.zeros(30) # as word vectors are of zero length
        cnt_words = 0; # num of words with a valid vector in the sentence/review
        for word in sentence.split(): # for each word in a review/sentence
            if word in glove_words:
                vector += model[word][:30]
                cnt_words += 1
        if cnt_words != 0:
            vector /= cnt_words
        avg_w2v_vectors_list.append(vector)

    print(len(avg_w2v_vectors_list))
    print(len(avg_w2v_vectors_list[0]))
    return avg_w2v_vectors_list

```

```
In [0]:
```

```
avgw2v_vec_essay_tr = avg_w2v(textpreprocessed(X_train,'essay'))  
avgw2v_vec_essay_te = avg_w2v(textpreprocessed(X_test,'essay'))  
avgw2v_vec_essay_val = avg_w2v(textpreprocessed(X_val,'essay'))
```

```
100%|██████████| 61178/61178 [00:38<00:00, 1588.16it/s]  
100%|██████████| 61178/61178 [00:16<00:00, 3786.16it/s]  
 1%|          | 158/21850 [00:00<00:13, 1573.61it/s]
```

61178

30

```
100%|██████████| 21850/21850 [00:13<00:00, 1588.33it/s]  
100%|██████████| 21850/21850 [00:05<00:00, 3737.31it/s]  
 1%|          | 164/26220 [00:00<00:15, 1637.83it/s]
```

21850

30

```
100%|██████████| 26220/26220 [00:16<00:00, 1588.60it/s]  
100%|██████████| 26220/26220 [00:07<00:00, 3307.26it/s]
```

26220

30

```
In [0]:
```

```
avgw2v_vec_titles_tr = avg_w2v(textpreprocessed(X_train,'project_title'))  
avgw2v_vec_titles_te = avg_w2v(textpreprocessed(X_test,'project_title'))  
avgw2v_vec_titles_val = avg_w2v(textpreprocessed(X_val,'project_title'))
```

```
100%|██████████| 61178/61178 [00:01<00:00, 33165.44it/s]  
100%|██████████| 61178/61178 [00:00<00:00, 86803.29it/s]  
 16%|████| 3394/21850 [00:00<00:00, 33936.63it/s]
```

61178

30

```
100%|██████████| 21850/21850 [00:00<00:00, 34205.62it/s]  
100%|██████████| 21850/21850 [00:00<00:00, 84418.17it/s]  
 13%|████| 3388/26220 [00:00<00:00, 33873.73it/s]
```

21850

30

```
100%|██████████| 26220/26220 [00:00<00:00, 33631.82it/s]  
100%|██████████| 26220/26220 [00:00<00:00, 85093.35it/s]
```

26220

30

```
In [0]:
```

```
avgw2v_vec_resource_tr = avg_w2v(textpreprocessed(X_train,'project_resource_summary'))  
avgw2v_vec_resource_te = avg_w2v(textpreprocessed(X_test,'project_resource_summary'))  
avgw2v_vec_resource_val = avg_w2v(textpreprocessed(X_val,'project_resource_summary'))
```

```
100%|██████████| 61178/61178 [00:04<00:00, 15195.72it/s]  
100%|██████████| 61178/61178 [00:01<00:00, 35479.11it/s]  
 7%|████| 1500/21850 [00:00<00:01, 14986.15it/s]
```

61178

30

```
100%|██████████| 21850/21850 [00:01<00:00, 15182.10it/s]
100%|██████████| 21850/21850 [00:00<00:00, 36305.57it/s]
 6%|██████████| 1504/26220 [00:00<00:01, 15039.05it/s]
```

21850
30

```
100%|██████████| 26220/26220 [00:01<00:00, 15343.34it/s]
100%|██████████| 26220/26220 [00:00<00:00, 35143.17it/s]
```

26220
30

In [0]:

```
X_tr, X_cr, X_te = hstack_data(avgw2v_vec_titles_tr, avgw2v_vec_titles_val, avgw2v_vec_titles_te,
\                                 avgw2v_vec_essay_tr, avgw2v_vec_essay_val, avgw2v_vec_essay_te,\n                                 avgw2v_vec_resource_tr, avgw2v_vec_resource_val, avgw2v_vec_resource_te)
```

In [0]:

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

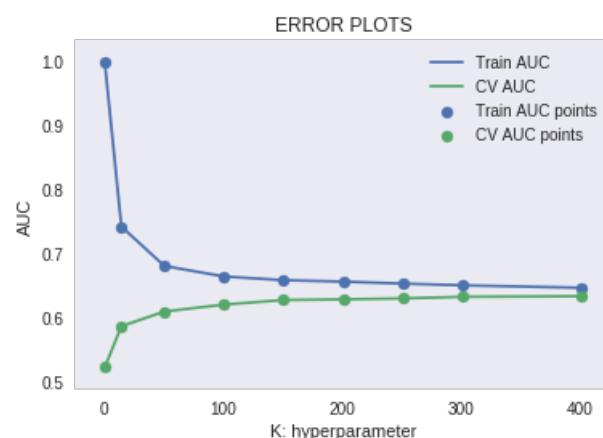
Out[0]:

```
((61178, 172), (26220, 172), (21850, 172))
```

In [0]:

```
best_k = optimised_k(X_tr,y_train,X_cr,y_val)
```

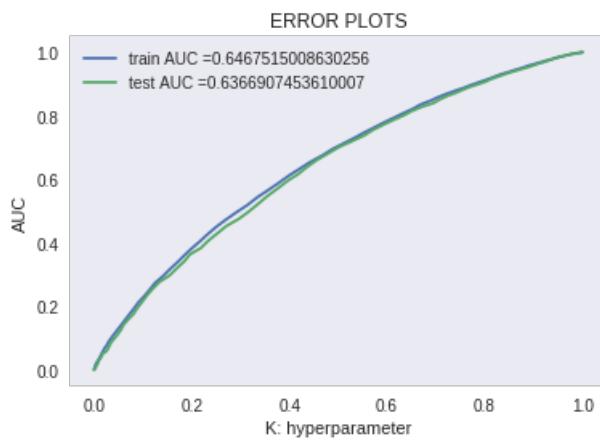
```
0%|          | 0/9 [00:00<?, ?it/s]
11%|████| 1/9 [45:40<6:05:21, 2740.17s/it]
22%|██████| 2/9 [1:32:06<5:21:17, 2753.89s/it]
33%|███████| 3/9 [2:18:21<4:36:01, 2760.22s/it]
44%|███████| 4/9 [3:04:41<3:50:31, 2766.21s/it]
56%|███████| 5/9 [3:51:02<3:04:43, 2770.81s/it]
67%|███████| 6/9 [4:37:27<2:18:44, 2774.93s/it]
78%|███████| 7/9 [5:23:51<1:32:35, 2777.79s/it]
89%|███████| 8/9 [6:10:10<46:18, 2778.13s/it]
100%|██████████| 9/9 [6:56:38<00:00, 2781.07s/it]
```



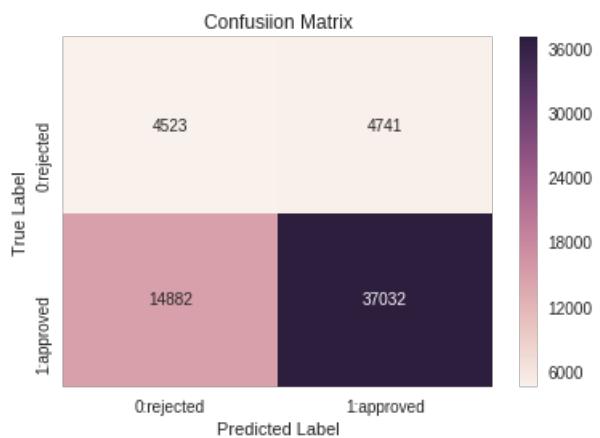
```
execution time in minutes: 416
execution time in hours: 6.933333333333334
0.6335998301870773
optimal K is: 401
```

In [0]:

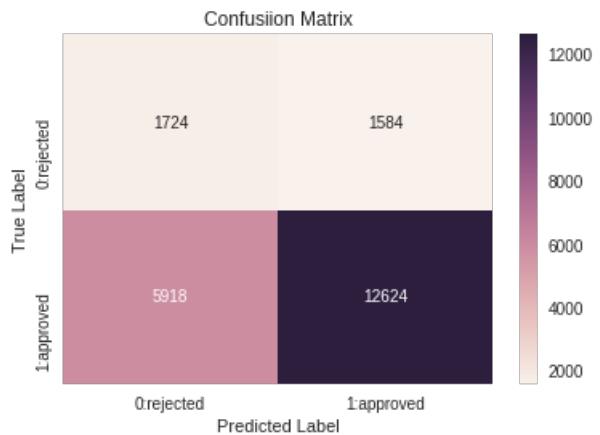
```
error_plot(x_tr,x_te,y_train,y_test)
```



```
=====  
Train confusion matrix  
the maximum value of tpr*(1-fpr) 0.24986156181299424 for threshold 0.835  
[[ 4523 4741]  
 [14882 37032]]  
Test confusion matrix  
the maximum value of tpr*(1-fpr) 0.24999415143969622 for threshold 0.84  
[[ 1724 1584]  
 [ 5918 12624]]  
the maximum value of tpr*(1-fpr) 0.24986156181299424 for threshold 0.835
```



```
the maximum value of tpr*(1-fpr) 0.24999415143969622 for threshold 0.84
```



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):
    vectorizer = TfidfVectorizer(min_df=10)
    text_tfidf_tr = vectorizer.fit_transform(preprocessed_data_tr)
    print("Shape of matrix after one hot encoding ", text_tfidf_tr.shape)

    text_tfidf_val = vectorizer.transform(preprocessed_data_val)
    print("Shape of matrix after one hot encoding ", text_tfidf_val.shape)

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

In [0]:

```
tfidf_vec_essay_tr, tfidf_vec_essay_val, tfidf_vec_essay_te =
tfidf_vec(textpreprocessed(X_train, 'essay'), textpreprocessed(X_val, 'essay'), textpreprocessed(X_test, 'essay'))
tfidf_vec_titles_tr, tfidf_vec_titles_val, tfidf_vec_titles_te = tfidf_vec(textpreprocessed(X_train, 'project_title'), textpreprocessed(X_val, 'project_title'), textpreprocessed(X_test, 'project_title'))
tfidf_vec_resource_tr, tfidf_vec_resource_val, tfidf_vec_resource_te = tfidf_vec(textpreprocessed(X_train, 'project_resource_summary'), textpreprocessed(X_val, 'project_resource_summary'), textpreprocessed(X_test, 'project_resource_summary'))
```

```
100%|██████████| 61178/61178 [00:37<00:00, 1646.00it/s]
100%|██████████| 26220/26220 [00:15<00:00, 1644.27it/s]
100%|██████████| 21850/21850 [00:13<00:00, 1642.86it/s]
```

Shape of matrix after one hot encoding (61178, 13150)
Shape of matrix after one hot encoding (26220, 13150)

```
6%|█          | 3512/61178 [00:00<00:01, 35119.11it/s]
```

Shape of matrix after one hot encoding (21850, 13150)

```
100%|██████████| 61178/61178 [00:01<00:00, 35300.24it/s]
100%|██████████| 26220/26220 [00:00<00:00, 35023.29it/s]
100%|██████████| 21850/21850 [00:00<00:00, 34172.18it/s]
```

Shape of matrix after one hot encoding (61178, 2274)
Shape of matrix after one hot encoding (26220, 2274)

```
3%|█          | 1550/61178 [00:00<00:03, 15495.10it/s]
```

Shape of matrix after one hot encoding (21850, 2274)

```
100%|██████████| 61178/61178 [00:03<00:00, 15655.91it/s]
100%|██████████| 26220/26220 [00:01<00:00, 15722.07it/s]
100%|██████████| 21850/21850 [00:01<00:00, 15475.88it/s]
```

Shape of matrix after one hot encoding (61178, 4348)
Shape of matrix after one hot encoding (26220, 4348)
Shape of matrix after one hot encoding (21850, 4348)

In [0]:

```
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)
```

```
(61178, 2)
(61178, 4)
(61178, 51)
(61178, 9)
(61178, 9)
(61178, 5)
(61178, 1)
(61178, 1)
(61178, 2274)
(61178, 13150)
```

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 [0]:

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

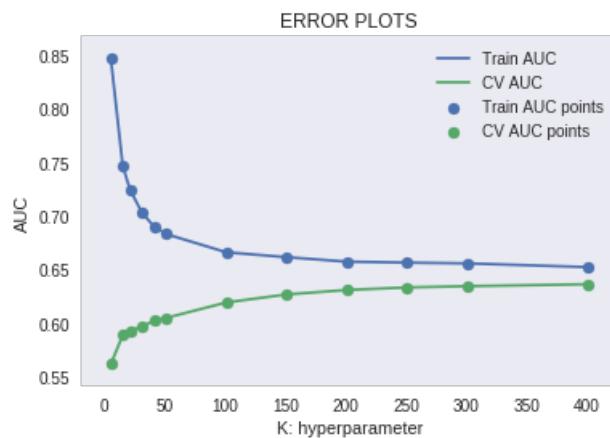
Out[0]:

```
((61178, 19854), (26220, 19854), (21850, 19854))
```

In [0]:

```
best_k = optimised_k(X_tr,y_train,X_cr,y_val)
```

```
100%|██████████| 12/12 [2:39:39<00:00, 803.62s/it]
```



```
execution time in minutes: 159
```

```
execution time in hours: 2
```

```
0.6359927037047518
```

```
optimal K is: 401
```

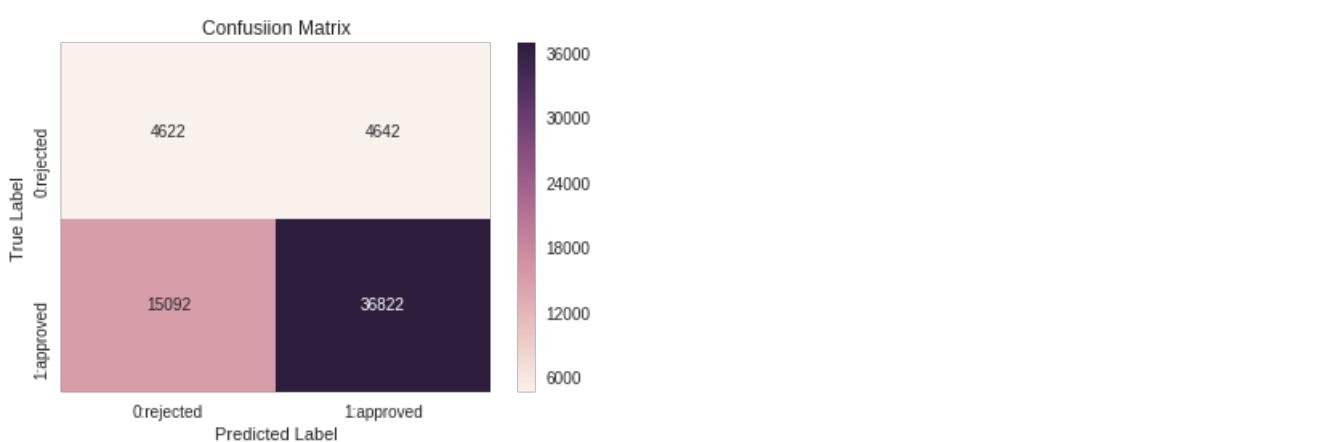
In [0]:

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

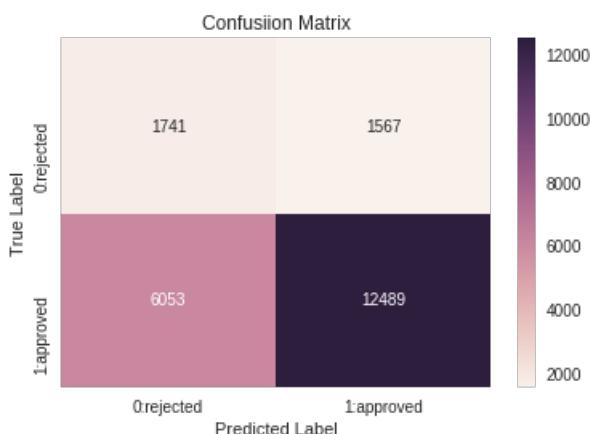




```
=====
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24999883479347693 for threshold 0.83
[[ 4622  4642]
 [15092 36822]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24998208878406972 for threshold 0.835
[[ 1741  1567]
 [ 6053 12489]]
the maximum value of tpr*(1-fpr) 0.24999883479347693 for threshold 0.83
```



```
the maximum value of tpr*(1-fpr) 0.24998208878406972 for threshold 0.835
```



[Task-2]

```
In [0]:
```

```
from sklearn.feature_selection import SelectKBest, chi2
X, y = X_tr,y_train
print(X.shape)
X_new = SelectKBest(chi2, k=2000).fit_transform(X, y)
print(X_new.shape)
selector = SelectKBest(chi2, k=2000)
selector.fit(X, y)

X_new = selector.transform(X)
X_new.shape
print(selector.get_support(indices=True))

'''indexes selected = selector.get_support(indices=True) # extracting features indexes
```

```
selected_features = [] # it will contain features names
for i in indexes_selected:
    selected_features.append(features[i])'''
```

```
(61178, 19854)
(61178, 2000)
[    0      1      2 ... 19835 19838 19841]
```

Out[0]:

```
'indexes_selected = selector.get_support(indices=True) # extracting features
indexes\nselected_features = [] # it will contain features names\nfor i in indexes_selected:\n    selected_features.append(features[i])'
```

In [0]:

```
indexes_selected = selector.get_support(indices=True) # extracting features indexes
selected_features = [] # it will contain features names
for i in indexes_selected:
    selected_features.append(i)
```

In [0]:

```
df = X_te.toarray()
df1 = pd.DataFrame(df)
X_te_new = csr_matrix(df1[list(selected_features)].values)
X_te.shape,X_te_new.shape
```

Out[0]:

```
((21850, 19854), (21850, 2000))
```

In [0]:

```
df = X_cr.toarray()
df1 = pd.DataFrame(df)
X_cr_new = csr_matrix(df1[list(selected_features)].values)
X_cr.shape,X_cr_new.shape
```

Out[0]:

```
((26220, 19854), (26220, 2000))
```

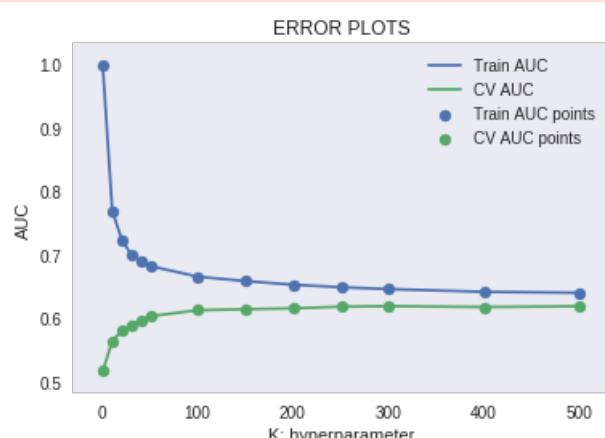
In [0]:

```
X_tr_new = X_new
```

In [0]:

```
best_k = optimised_k(X_tr_new,y_train,X_cr_new,y_val)
```

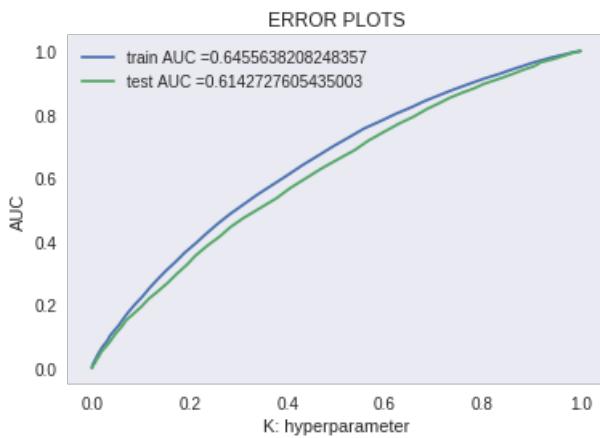
```
100%|██████████| 13/13 [1:36:22<00:00, 452.57s/it]
```



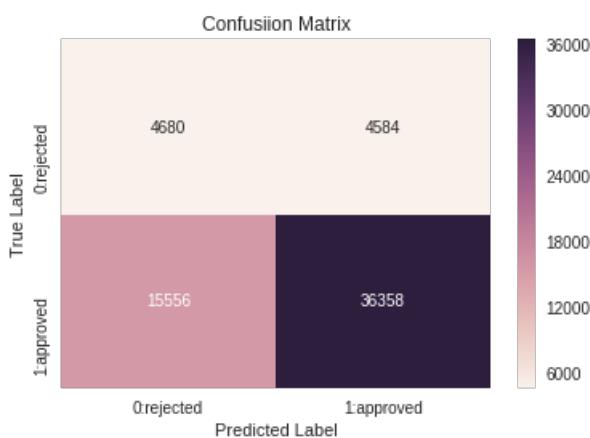
```
execution time in minutes: 96
execution time in hours: 1.6
0.6186178529986132
optimal K is: 301
```

In [0]:

```
error_plot(X_tr_new,X_te_new,y_train,y_test)
```



```
=====
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2499731536417085 for threshold 0.837
[[ 4680  4584]
 [15556 36358]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24998894256567566 for threshold 0.844
[[ 1751  1557]
 [ 6858 11684]]
the maximum value of tpr*(1-fpr) 0.2499731536417085 for threshold 0.837
```



```
the maximum value of tpr*(1-fpr) 0.24998894256567566 for threshold 0.844
```



urejected Approved
Predicted Label

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):  
    vectorizer = CountVectorizer(min_df=10)  
    text_bow_tr = vectorizer.fit_transform(preprocessed_data_tr)  
    print("Shape of matrix after one hot encoding ",text_bow_tr.shape)  
  
    text_bow_val = vectorizer.transform(preprocessed_data_val)  
    print("Shape of matrix after one hot encoding ",text_bow_val.shape)  
  
    text_bow_te = vectorizer.transform(preprocessed_data_te)  
    print("Shape of matrix after one hot encoding ",text_bow_te.shape)  
    return text_bow_tr,text_bow_val, text_bow_te
```

In [0]:

```
bow_vec_essay_tr,bow_vec_essay_val,bow_vec_essay_te = bow_vec(textpreprocessed(X_train,'essay'),te  
xtpreprocessed(X_val,'essay'), textpreprocessed(X_test,'essay'))  
bow_vec_titles_tr,bow_vec_titles_val,bow_vec_titles_te =  
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(textpreprocessed(X_train,'project_resource_summary'),textpreprocessed(X_val,'project_resour  
ce_summary'), textpreprocessed(X_test,'project_resource_summary'))
```

100% | ██████████ | 61178/61178 [00:38<00:00, 1603.54it/s]
100% | ██████████ | 26220/26220 [00:15<00:00, 1640.37it/s]
100% | ██████████ | 21850/21850 [00:13<00:00, 1636.87it/s]

Shape of matrix after one hot encoding (61178, 13145)
Shape of matrix after one hot encoding (26220, 13145)

6% | █████ | 3556/61178 [00:00<00:01, 35553.42it/s]

Shape of matrix after one hot encoding (21850, 13145)

100% | ██████████ | 61178/61178 [00:01<00:00, 35204.18it/s]
100% | ██████████ | 26220/26220 [00:00<00:00, 34550.18it/s]
100% | ██████████ | 21850/21850 [00:00<00:00, 34856.56it/s]

Shape of matrix after one hot encoding (61178, 2295)
Shape of matrix after one hot encoding (26220, 2295)

3% | █████ | 1606/61178 [00:00<00:03, 16053.74it/s]

Shape of matrix after one hot encoding (21850, 2295)

100% | ██████████ | 61178/61178 [00:03<00:00, 15566.92it/s]
100% | ██████████ | 26220/26220 [00:01<00:00, 15555.07it/s]
100% | ██████████ | 21850/21850 [00:01<00:00, 15482.56it/s]

Shape of matrix after one hot encoding (61178, 4353)
Shape of matrix after one hot encoding (26220, 4353)
Shape of matrix after one hot encoding (21850, 4353)

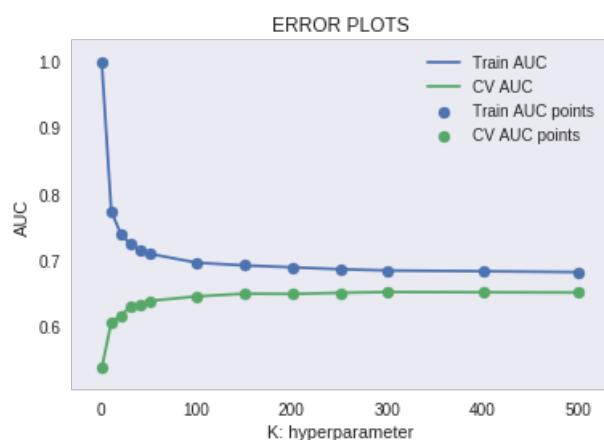
In [0]:

```
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 [0]:

```
best_k = optimised_k(X_tr,y_train,X_cr,y_val)
```

100%|██████████| 13/13 [2:54:35<00:00, 812.13s/it]



execution time in minutes: 174

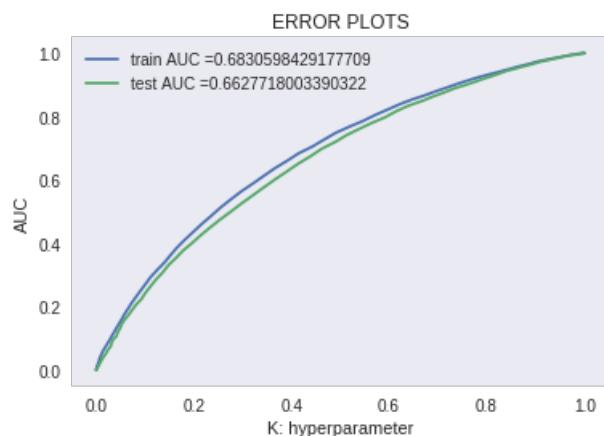
execution time in hours: 2.9

0.6504490815951093

optimal K is: 301

In [0]:

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



=====
Train confusion matrix

the maximum value of tpr*(1-fpr) 0.2499056182716315 for threshold 0.781

[[4722 4542]

[13089 38825]]

Test confusion matrix

the maximum value of tpr*(1-fpr) 0.2499004830910811 for threshold 0.794

[[1941 1367]

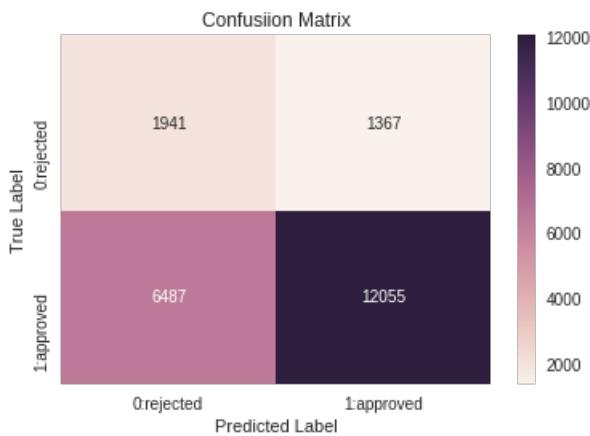
[6487 12055]]

the maximum value of tpr*(1-fpr) 0.2499056182716315 for threshold 0.781





the maximum value of $tpr * (1 - fpr)$ 0.2499004830910811 for threshold 0.794



Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)

In [0]:

```
# average Word2Vec
# compute average word2vec for each review.
def tfidf_w2v(preprocessed_list):
    # average Word2Vec
    preprocessed_list = preprocessed_list[:]
    tfidf_model = TfidfVectorizer()
    tfidf_model.fit(preprocessed_list)
    # we are converting a dictionary with word as a key, and the idf as a value
    dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
    tfidf_words = set(tfidf_model.get_feature_names())
    tfidf_w2v_vectors_list = [] # the avg-w2v for each sentence/review is stored in this list
    for sentence in tqdm(preprocessed_list): # for each review/sentence
        vector = np.zeros(30) # as word vectors are of zero length
        tf_idf_weight = 0; # num of words with a valid vector in the sentence/review
        for word in sentence.split(): # for each word in a review/sentence
            if (word in glove_words) and (word in tfidf_words):
                vec = model[word] [:30] # getting the vector for each word
                # here we are multiplying idf value(dictionary[word]) and the tf
                value((sentence.count(word)/len(sentence.split())))
                tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the
                tfidf value for each word
                vector += (vec * tf_idf) # calculating tfidf weighted w2v
                tf_idf_weight += tf_idf
        if tf_idf_weight != 0:
            vector /= tf_idf_weight
        tfidf_w2v_vectors_list.append(vector)

    print(len(tfidf_w2v_vectors_list))
    print(len(tfidf_w2v_vectors_list[0]))
    return tfidf_w2v_vectors_list
```

In [0]:

```
tfidfw2v_vec_essay_tr = tfidf_w2v(textpreprocessed(X_train, 'essay'))
tfidfw2v_vec_essay_te = tfidf_w2v(textpreprocessed(X_test, 'essay'))
tfidfw2v_vec_essay_val = tfidf_w2v(textpreprocessed(X_val, 'essay'))
```

100%|██████████| 61178/61178 [00:37<00:00, 1651.96it/s]
100%|██████████| 61178/61178 [01:43<00:00, 590.46it/s]
1%| | 162/21850 [00:00<00:13, 1616.00it/s]

```
61178
```

```
30
```

```
100%|██████████| 21850/21850 [00:13<00:00, 1649.94it/s]
100%|██████████| 21850/21850 [00:38<00:00, 572.61it/s]
 1%|          | 166/26220 [00:00<00:15, 1654.00it/s]
```

```
21850
```

```
30
```

```
100%|██████████| 26220/26220 [00:15<00:00, 1644.64it/s]
100%|██████████| 26220/26220 [00:45<00:00, 578.58it/s]
```

```
26220
```

```
30
```

```
In [0]:
```

```
tfidfw2v_vec_titles_tr = tfidf_w2v(textpreprocessed(X_train,'project_title'))
tfidfw2v_vec_titles_te = tfidf_w2v(textpreprocessed(X_test,'project_title'))
tfidfw2v_vec_titles_val = tfidf_w2v(textpreprocessed(X_val,'project_title'))
```

```
100%|██████████| 61178/61178 [00:01<00:00, 35317.46it/s]
100%|██████████| 61178/61178 [00:01<00:00, 45150.33it/s]
 16%|█         | 3547/21850 [00:00<00:00, 35463.69it/s]
```

```
61178
```

```
30
```

```
100%|██████████| 21850/21850 [00:00<00:00, 35215.37it/s]
100%|██████████| 21850/21850 [00:00<00:00, 44991.44it/s]
 14%|█         | 3555/26220 [00:00<00:00, 35541.56it/s]
```

```
21850
```

```
30
```

```
100%|██████████| 26220/26220 [00:00<00:00, 35337.65it/s]
100%|██████████| 26220/26220 [00:00<00:00, 45371.03it/s]
```

```
26220
```

```
30
```

```
In [0]:
```

```
tfidfw2v_vec_resource_tr = tfidf_w2v(textpreprocessed(X_train,'project_resource_summary'))
tfidfw2v_vec_resource_te = tfidf_w2v(textpreprocessed(X_test,'project_resource_summary'))
tfidfw2v_vec_resource_val = tfidf_w2v(textpreprocessed(X_val,'project_resource_summary'))
```

```
100%|██████████| 61178/61178 [00:03<00:00, 15673.21it/s]
100%|██████████| 61178/61178 [00:04<00:00, 14329.76it/s]
  7%|█         | 1528/21850 [00:00<00:01, 15277.57it/s]
```

```
61178
```

```
30
```

```
100%|██████████| 21850/21850 [00:01<00:00, 15461.22it/s]
100%|██████████| 21850/21850 [00:01<00:00, 14284.73it/s]
  6%|█         | 1568/26220 [00:00<00:01, 15675.01it/s]
```

```
21850
```

```
30
```

```
100%|██████████| 26220/26220 [00:01<00:00, 15586.98it/s]
100%|██████████| 26220/26220 [00:01<00:00, 14346.76it/s]
```

```
26220  
30
```

```
In [0]:
```

```
X_tr, X_cr, X_te = hstack_data(tfidfw2v_vec_titles_tr, tfidfw2v_vec_titles_val,  
tfidfw2v_vec_titles_te, \  
tfidfw2v_vec_essay_tr, tfidfw2v_vec_essay_val, tfidfw2v_vec_essay_te,\  
tfidfw2v_vec_resource_tr, tfidfw2v_vec_resource_val, tfidfw2v_vec_resource_te)
```

```
In [0]:
```

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

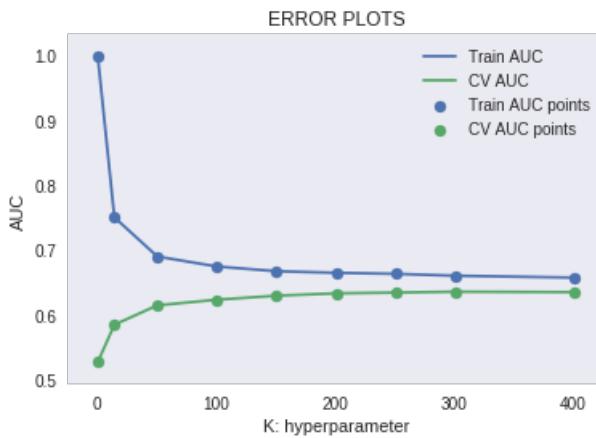
```
Out[0]:
```

```
((61178, 172), (26220, 172), (21850, 172))
```

```
In [0]:
```

```
best_k = optimised_k(X_tr,y_train,X_cr,y_val)
```

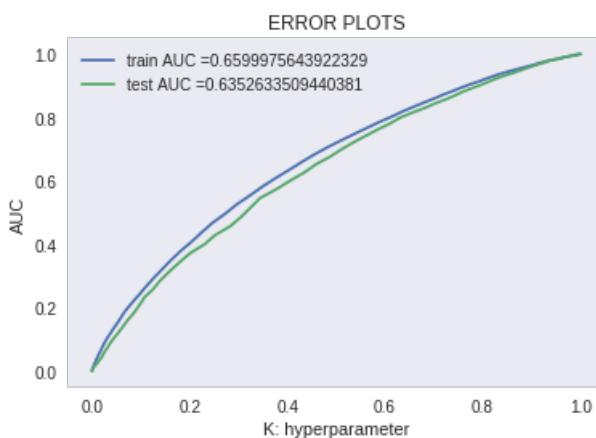
```
100%|██████████| 9/9 [6:49:47<00:00, 2730.41s/it]
```



```
execution time in minutes: 409  
execution time in hours: 6.816666666666666  
0.6353322729459712  
optimal K is: 301
```

```
In [0]:
```

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

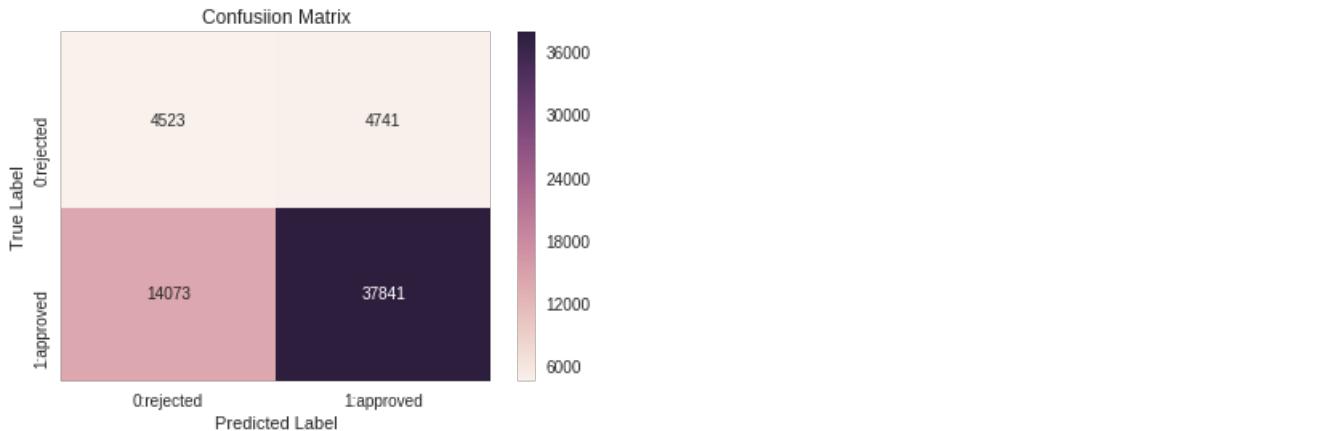


```
=====
```

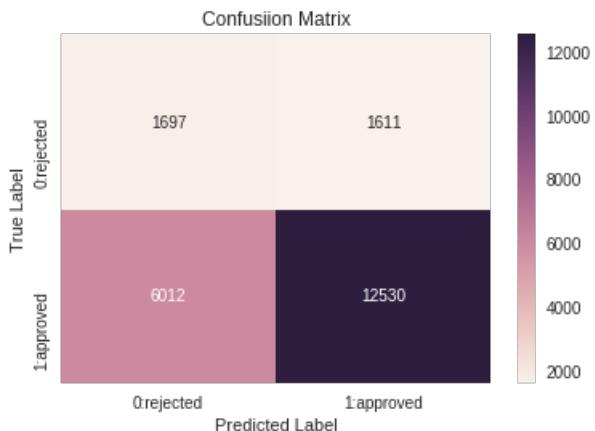
```

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24986156181299424 for threshold 0.834
[[ 4523  4741]
 [14073 37841]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24987489563975207 for threshold 0.841
[[ 1697  1611]
 [ 6012 12530]]
the maximum value of tpr*(1-fpr) 0.24986156181299424 for threshold 0.834

```



```
the maximum value of tpr*(1-fpr) 0.24987489563975207 for threshold 0.841
```



Summary

```
In [0]:
```

```

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', 301, 0.6627])
x.add_row(["TFIDF", 'Brute', 401, 0.6378])
x.add_row(["W2V", 'Brute', 401, 0.6366])
x.add_row(["TFIDFW2V", 'Brute', 301, 0.6352])

print(x)

```

Vectorizer	Model	Hyper parameter	AUC
BOW	Brute	301	0.6627
TFIDF	Brute	401	0.6378
W2V	Brute	401	0.6366
TFIDFW2V	Brute	301	0.6352

--	--	--	--