In [1]:

```
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 [2]:

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

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

In [3]:

```
%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]:

```
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', \
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [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 [14]:

```
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[14]:

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

In [0]:

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

In [0]:

```
project_data['words_in_title'] = project_data['project_title'].str.count(' ') + 1
project_data['words_in_essay'] = project_data['essay'].str.count(' ') + 1
project_data['words_in_summary'] = project_data['project_resource_summary'].str.count(' ') + 1
```

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
.30)
```

In [18]:

```
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: (87398, 23) 87398
Test: (21850, 23) 21850
```

Function for standardizing/normalizing the data

In [0]:

```
from sklearn.preprocessing import StandardScaler
def standardize_data(df_tr,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 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_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_test = normalizer_vec.transform(df_te[column_name].values.reshape(-1,1))
  print(normalizer_data_test.shape)

  return normalizer_data_train, normalizer_data_test
```

Function for vectorizing the text data

In [0]:

```
from sklearn.feature_extraction.text import CountVectorizer
def vectorized_data(df_train,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_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_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('\\"', ' ')
    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 [25]:

```
price_standardized_tr, price_standardized_te = standardize_data(X_train, X_test,'price')
print()
project_standardized_tr, project_standardized_te = standardize_data(X_train,
X_test,'teacher_number_of_previously_posted_projects')
print()
title_standardized_tr, title_standardized_te = standardize_data(X_train, X_test,'words_in_title')
print()
essay_standardized_tr, essay_standardized_te = standardize_data(X_train, X_test,'words_in_essay')
print()
resource_standardized_tr, resource_standardized_te = standardize_data(X_train, X_test,'words_in_sum
mary')
```

```
(87398, 1)
(21850, 1)

(87398, 1)
(21850, 1)

(87398, 1)
(21850, 1)

(87398, 1)
(21850, 1)

(87398, 1)
(21850, 1)
```

In [26]:

```
qty_standardized_tr, qty_standardized_te = standardize_data(X_train, X_test,'quantity')
```

```
(87398, 1)
(21850, 1)
```

In [27]:

```
cat_one_hot_tr,cat_one_hot_te = vectorized_data(X_train,X_test,'clean_categories',vocab=sorted_cat_
dict)
cat_sub_one_hot_tr,cat_sub_one_hot_te = vectorized_data(X_train,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  (87398, 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  (87398, 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 [28]:

```
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_te = vectorized_data(X_train,X_test,'school_state',vocab=school_stat
e_dict)
teacher_one_hot_tr,teacher_one_hot_te =
vectorized_data(X_train,X_test,'teacher_prefix',vocab=teacher_prefix_dict)
```

```
['VT', 'WY', 'ND', 'MT', 'RI', 'NE', 'SD', 'AK', 'DE', 'NH', 'WV', 'HI', 'ME', 'DC', 'NM', 'KS', 'I
A', 'ID', 'AR', 'CO', 'MN', 'OR', 'KY', 'MS', 'NV', 'MD', 'TN', 'CT', 'UT', 'AL', 'WI', 'VA', 'AZ',
'NJ', 'OK', 'WA', 'MA', 'LA', 'OH', 'MO', 'IN', 'PA', 'MI', 'SC', 'GA', 'IL', 'NC', 'FL', 'NY', 'TX
', 'CA']
Shape of matrix after one hot encoding  (87398, 51)
['VT', 'WY', 'ND', 'MT', 'RI', 'NE', 'SD', 'AK', 'DE', 'NH', 'WV', 'HI', 'ME', 'DC', 'NM', 'KS', 'I
A', 'ID', 'AR', 'CO', 'MN', 'OR', 'KY', 'MS', 'NV', 'MD', 'TN', 'CT', 'UT', 'AL', 'WI', 'VA', 'AZ',
'NJ', 'OK', 'WA', 'MA', 'LA', 'OH', 'MO', 'IN', 'PA', 'MI', 'SC', 'GA', '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  (87398, 5)
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
Shape of matrix after one hot encoding  (21850, 5)
```

In [29]:

```
grade_dict = create_dict(X_train,'project_grade_category')
grade_one_hot_tr,grade_one_hot_te = vectorized_data(X_train,X_test,'project_grade_category',vocab=g
rade_dict)
```

```
['Grades_9to12', 'Grades_6to8', 'Grades_3to5', 'Grades_PreKto2']
Shape of matrix after one hot encoding  (87398, 4)
['Grades_9to12', 'Grades_6to8', 'Grades_3to5', 'Grades_PreKto2']
Shape of matrix after one hot encoding  (21850, 4)
```

In [30]:

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

```
Shape of matrix after one hot encoding  (87398, 2)
Shape of matrix after one hot encoding  (21850, 2)
```

In [31]:

```
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[31]:

```
((21850, 2),
 (21850, 4),
 (21850, 51),
 (21850, 9),
 (21850, 9),
 (21850, 5),
 (21850, 1),
 (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,teac
her_one_hot_tr,price_standardized_tr,project_standardized_tr))
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_te, f2_tr,f2_te,f3_tr,f3_te):
  X_tr = hstack((f_tr, f1_tr, f2_tr, f3_tr)).tocsr()
  X_te = hstack((f_te, f1_te, f2_te, f3_te)).tocsr()
  return X_tr,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 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]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

def grid_plot(X_train,y_train):
  logistic = LogisticRegression()
  penalty = ['l1', 'l2']
  #C = [0.0001, 0.001, 0.01, 1, 100]
  C = [0.00001, 0.00005, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100]
  hyperparameters = dict(C=C, penalty=penalty)
  clf = GridSearchCV(logistic, hyperparameters, cv=5, verbose=0)
  best_model = clf.fit(X_train, y_train)
  print('Best Parameters',clf.best_params_)
  print('Best Penalty:', best_model.best_estimator_.get_params()['penalty'])
  print('Best C:', best_model.best_estimator_.get_params()['C'])
  score = clf.cv_results_
  plot_df = pd.DataFrame(score)
  plt.plot(plot_df["param_C"], 1- plot_df["mean_test_score"], "-o")
  plt.title("CV error vs C")
  plt.xlabel("C")
  plt.ylabel("Cross-validation Error")
  plt.show()
  return clf.best_estimator_.C
```

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):

  model =  LogisticRegression(penalty = 'l1', C = 1, class_weight = "balanced")
  model.fit(X_train, y_train)
  #pred = model.predict(X_train)
  #y_train_pred = model.predict_proba(X_train)

  #pred = model.predict(X_te)
  #y_test_pred = model.predict_proba(X_test)

  y_train_pred = batch_predict(model, X_train)
  y_test_pred = batch_predict(model, X_te)
  #print(len(y_train), len(y_train_pred))
  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("C: hyperparameter")
  plt.ylabel("AUC")
  plt.title("ERROR PLOTS")
  plt.grid()
  plt.show()
  print("="*100)
```

In [0]:

```
#https://datascience.stackexchange.com/questions/28493/confusion-matrix-get-items-fp-fn-tp-tn-python
def get_confusion_matrix_values(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    df = pd.DataFrame(data=cm, index=labels, columns=labels)
    print("Confusion Matrix : ")
    plt.figure(figsize=(10,7))
    sns.heatmap(df, annot=True)
    plt.show()
    TP, FP, FN, TN = cm[0][0], cm[0][1], cm[1][0], cm[1][1]

    print("True Positives :", TP)
    print("False Positives :", FP)
    print("True Negatives :", TN)
    print("False Negatives :", FN)
```

In [0]:

```
from sklearn.metrics import roc_auc_score
import time
def optimal_hyp(X_tr,y_train):
  global df1
  penalties = ['l1', 'l2']
  C = list(map(lambda x: 10 ** x, np.arange(-5, 3,dtype=float)))
  cols = ['C', 'Penalty', 'Train_AUC_Score', 'CV_AUC_Score']
  lst = []
  start = time.time()
  for c in C:
    for p in penalties:
        clf = LogisticRegression(penalty=p, C=c)
        clf.fit(X_tr, y_train)
        tr_score = roc_auc_score(y_true=np.array(y_train), y_score=clf.predict_proba(X_tr)[:,1])
        cv_score = roc_auc_score(y_true=np.array(y_test), y_score=clf.predict_proba(X_te)[:,1])
        lst.append([c,p,tr_score,cv_score])
  end = time.time()
  minutes = float((end - start)/60)
  print("execution time in minutes:",minutes)
  print("execution time in hours:",int(minutes/60))

  df1 = pd.DataFrame(lst, columns=cols)
```

In [0]:

```
def plot_auc(df1):
  df = df1[['Train_AUC_Score','CV_AUC_Score']]
  df1['C'] = df1['C'].astype(str)
  ax = df.plot(xticks=df.index, rot=45)
  ax.set_xticklabels(df1[['C', 'Penalty']].apply(lambda x: ''.join(x), axis=1))
  ax
```

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]:

```
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 [42]:

```
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%|██████████| 87398/87398 [01:43<00:00, 845.92it/s]
100%|██████████| 87398/87398 [00:44<00:00, 1979.01it/s]
  0%|          | 90/21850 [00:00<00:24, 888.43it/s]
```

```
87398
30
```

```
100%|██████████| 21850/21850 [00:25<00:00, 847.74it/s]
100%|██████████| 21850/21850 [00:10<00:00, 2033.95it/s]
```

```
21850
30
```

In [43]:

```
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%|██████████| 87398/87398 [00:04<00:00, 21437.59it/s]
100%|██████████| 87398/87398 [00:01<00:00, 53070.60it/s]
 10%|█         | 2158/21850 [00:00<00:00, 21571.84it/s]
```

```
87398
30
```

```
100%|██████████| 21850/21850 [00:01<00:00, 21324.47it/s]
100%|██████████| 21850/21850 [00:00<00:00, 52202.67it/s]
```

```
21850
30
```

In [44]:

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

```
100%|██████████| 87398/87398 [00:10<00:00, 8510.85it/s]
100%|██████████| 87398/87398 [00:04<00:00, 19808.45it/s]
  4%|▍         | 853/21850 [00:00<00:02, 8522.33it/s]
```

```
87398
30
```

```
100%|██████████| 21850/21850 [00:02<00:00, 8451.30it/s]
100%|██████████| 21850/21850 [00:01<00:00, 20180.96it/s]
```

```
21850
30
```

In [0]:

```
X_tr, X_te = hstack_data(avgw2v_vec_titles_tr, avgw2v_vec_titles_te, \
                   avgw2v_vec_essay_tr, avgw2v_vec_essay_te, \
                   avgw2v_vec_resource_tr, avgw2v_vec_resource_te)
```

In [46]:

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

Out[46]:

```
((87398, 172), (21850, 172))
```

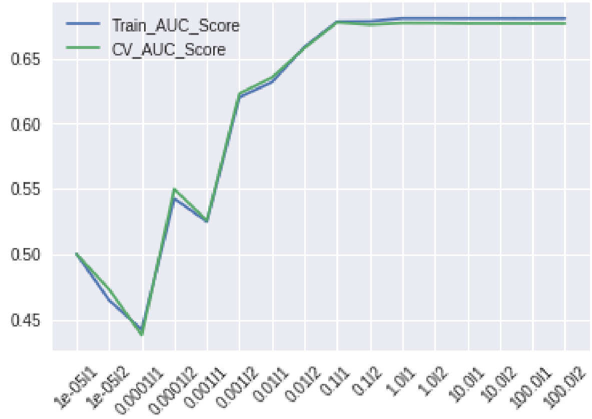
In [47]:

```
optimal_hyp(X_tr,y_train)
```

```
execution time in minutes: 21.90871280034383
execution time in hours: 0
```

In [48]:

```
plot_auc(df1)
```



In [49]:

```
best_c,p,tr_auc_score,cv_auc_score = df1.iloc[df1.CV_AUC_Score.argmax()]
print("optimal c:",best_c,"\nRegularizer:",p,"\nCV AUC score:", cv_auc_score)
```

```
optimal c: 0.1
Regularizer: l1
CV AUC score: 0.6775136599584954
```

In [50]:

```
model = LogisticRegression(C=float(best_c), penalty=p,class_weight = "balanced")
model.fit(X_tr, y_train)
```

Out[50]:

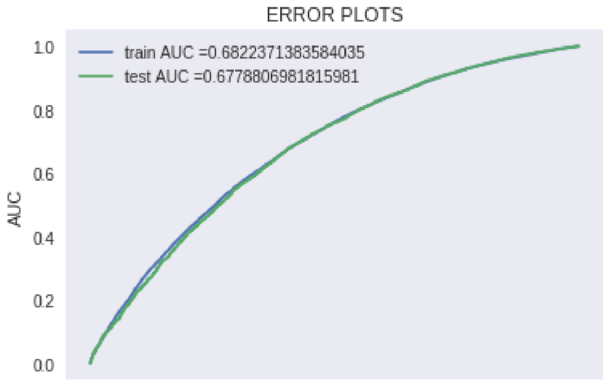
```
LogisticRegression(C=0.1, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l1', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
```

In [0]:

```
y_pred = model.predict(X_te)
```

In [52]:

```
start = time.time()
error_plot(X_tr,X_te,y_train,y_test)
end = time.time()
minutes = float((end - start)/60)
print("execution time in minutes:",minutes)
print("execution time in hours:",int(minutes/60))
```



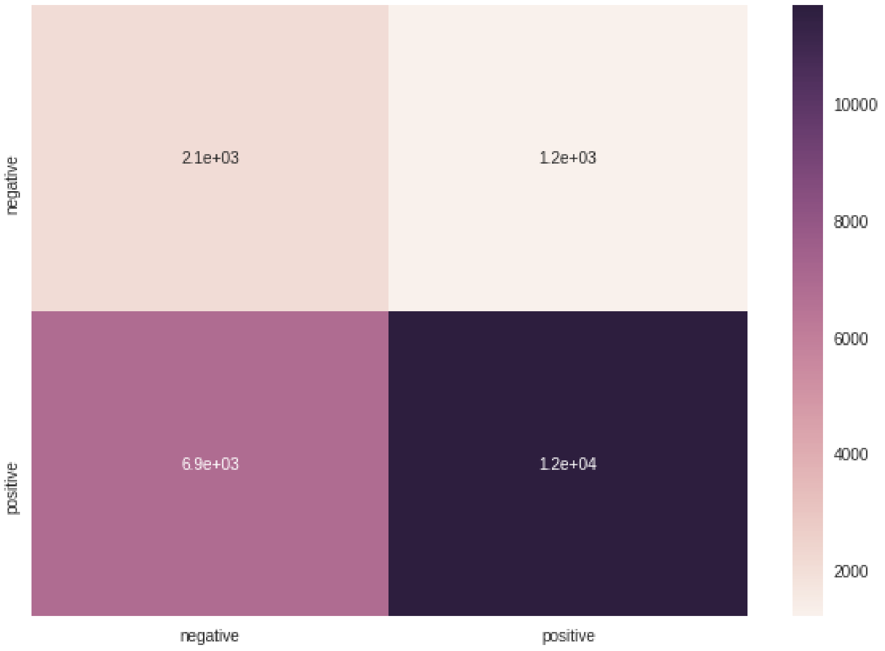
```
====================================================================================================

execution time in minutes: 58.060552549362185
execution time in hours: 0
```

In [53]:

```
labels = ["negative","positive"]
get_confusion_matrix_values(np.array(y_test), y_pred)
```

```
Confusion Matrix :
```



```
True Positives : 2087
False Positives : 1221
True Negatives : 11665
False Negatives : 6877
```

In [54]:

```
# Printing roc auc score
y_pred = model.predict_proba(X_te)
roc_auc_score(y_true=y_test, y_score=y_pred[:,1])
```

Out[54]:

```
0.6788061764285063
```

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_te):
  vectorizer = TfidfVectorizer(min_df=10,ngram_range=(2,2), max_features=5000)
  text_tfidf_tr = vectorizer.fit_transform(preprocessed_data_tr)
  print("Shape of matrix after one hot encodig ",text_tfidf_tr.shape)

  text_tfidf_te = vectorizer.transform(preprocessed_data_te)
  print("Shape of matrix after one hot encodig ",text_tfidf_te.shape)
```

```
    return text_tfidf_tr, text_tfidf_te
```

In [0]:

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

```
100%|██████████| 87398/87398 [01:47<00:00, 811.63it/s]
100%|██████████| 21850/21850 [00:27<00:00, 787.54it/s]
```

```
Shape of matrix after one hot encodig  (87398, 5000)
Shape of matrix after one hot encodig  (21850, 5000)
```

```
100%|██████████| 87398/87398 [00:04<00:00, 19174.31it/s]
100%|██████████| 21850/21850 [00:01<00:00, 19068.83it/s]
```

```
Shape of matrix after one hot encodig  (87398, 2480)
```

```
  1%|          | 785/87398 [00:00<00:11, 7845.24it/s]
```

```
Shape of matrix after one hot encodig  (21850, 2480)
```

```
100%|██████████| 87398/87398 [00:11<00:00, 7793.34it/s]
100%|██████████| 21850/21850 [00:02<00:00, 7683.40it/s]
```

```
Shape of matrix after one hot encodig  (87398, 5000)
Shape of matrix after one hot encodig  (21850, 5000)
```

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)
```

```
(87398, 2)
(87398, 4)
(87398, 51)
(87398, 9)
(87398, 9)
(87398, 5)
(87398, 1)
(87398, 1)
(87398, 2480)
(87398, 5000)
```

In [0]:

```
X_tr, X_te = hstack_data(tfidf_vec_titles_tr, tfidf_vec_titles_te, \
                   tfidf_vec_essay_tr, tfidf_vec_essay_te,\
                   tfidf_vec_resource_tr, tfidf_vec_resource_te)
```

In [0]:

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

Out[0]:

```
((87398, 12562), (21850, 12562))
```

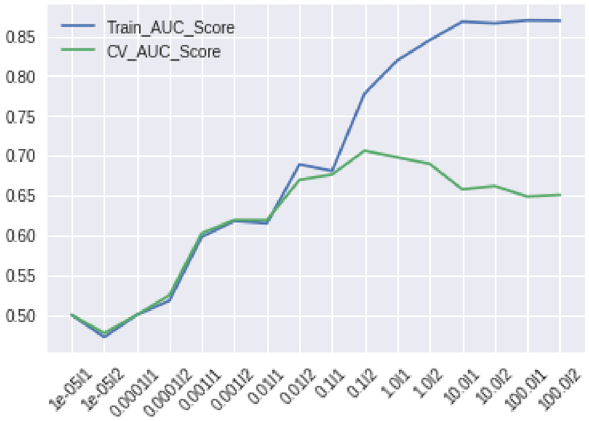
In [0]:

```
optimal_hyp(X_tr,y_train)
```

```
execution time in minutes: 2.5380945046742758
execution time in hours: 0
```

In [0]:

```
plot_auc(df1)
```



In [0]:

```
best_c,p,tr_auc_score,cv_auc_score = df1.iloc[df1.CV_AUC_Score.argmax()]
print("optimal c:",best_c,"\nRegularizer:",p,"\nCV AUC score:", cv_auc_score)
```

```
optimal c: 0.1
Regularizer: l2
CV AUC score: 0.7061634037278941
```

In [0]:

```
model = LogisticRegression(C=float(best_c), penalty=p,class_weight = "balanced")
model.fit(X_tr, y_train)
```

Out[0]:

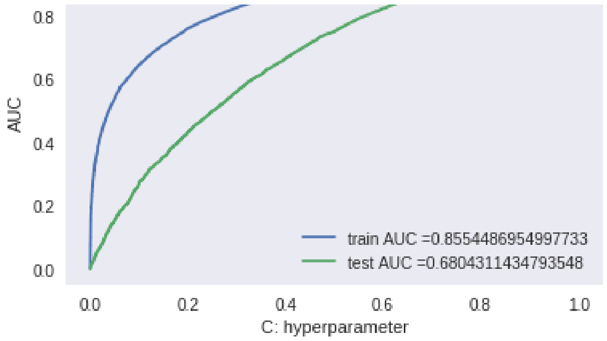
```
LogisticRegression(C=0.1, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
```

In [0]:

```
y_pred = model.predict(X_te)
```

In [0]:

```
start = time.time()
error_plot(X_tr,X_te,y_train,y_test)
end = time.time()
minutes = float((end - start)/60)
print("execution time in minutes:",minutes)
print("execution time in hours:",int(minutes/60))
```

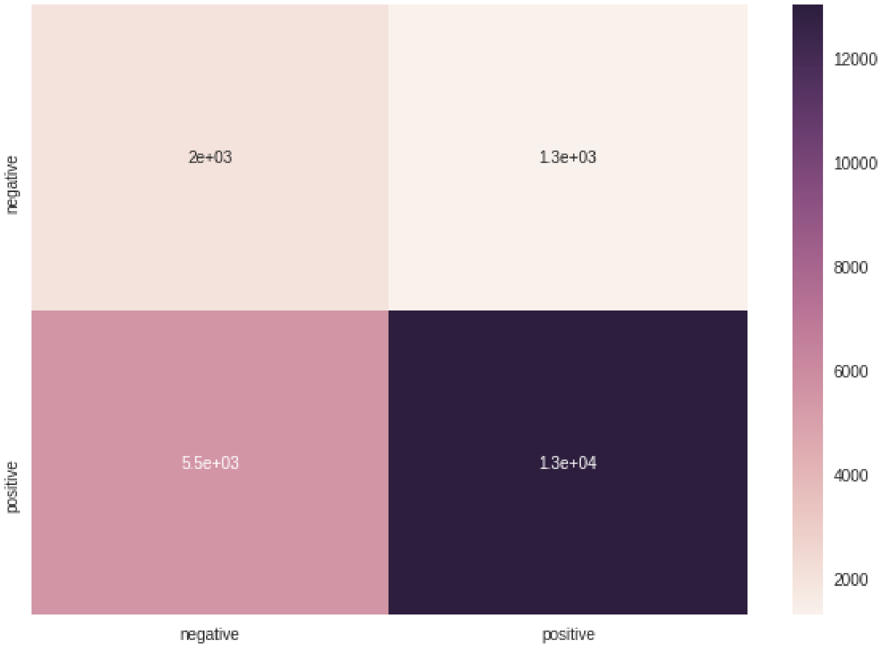


```
====================================================================================================
```

In [0]:

```
labels = ["negative","positive"]
get_confusion_matrix_values(np.array(y_test), y_pred)
```

```
Confusion Matrix :
```



```
True Positives : 1995
False Positives : 1313
True Negatives : 13006
False Negatives : 5536
```

In [0]:

```
# Printing roc auc score
y_pred = model.predict_proba(X_te)
roc_auc_score(y_true=y_test, y_score=y_pred[:,1])
```

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

In [0]:

```
def bow_vec(preprocessed_data_tr,preprocessed_data_te):
  vectorizer = CountVectorizer(min_df=10,ngram_range=(2,2), max_features=5000)
  text_bow_tr = vectorizer.fit_transform(preprocessed_data_tr)
  print("Shape of matrix after one hot encodig ",text_bow_tr.shape)

  text_bow_te = vectorizer.transform(preprocessed_data_te)
  print("Shape of matrix after one hot encodig ",text_bow_te.shape)
```

```
    return text_bow_tr, text_bow_te
```

In [0]:

```
bow_vec_essay_tr,bow_vec_essay_te = bow_vec(textpreprocessed(X_train,'essay'),
textpreprocessed(X_test,'essay'))
bow_vec_titles_tr,bow_vec_titles_te = bow_vec(textpreprocessed(X_train,'project_title'),
textpreprocessed(X_test,'project_title'))
bow_vec_resource_tr,bow_vec_resource_te =
bow_vec(textpreprocessed(X_train,'project_resource_summary'),
textpreprocessed(X_test,'project_resource_summary'))
```

```
100%|██████████| 87398/87398 [00:54<00:00, 1608.69it/s]
100%|██████████| 21850/21850 [00:13<00:00, 1625.04it/s]
```

```
Shape of matrix after one hot encodig  (87398, 5000)
Shape of matrix after one hot encodig  (21850, 5000)
```

```
100%|██████████| 87398/87398 [00:02<00:00, 34758.63it/s]
100%|██████████| 21850/21850 [00:00<00:00, 34673.15it/s]
  0%|          | 0/87398 [00:00<?, ?it/s]
```

```
Shape of matrix after one hot encodig  (87398, 2480)
Shape of matrix after one hot encodig  (21850, 2480)
```

```
100%|██████████| 87398/87398 [00:05<00:00, 15371.91it/s]
100%|██████████| 21850/21850 [00:01<00:00, 15322.12it/s]
```

```
Shape of matrix after one hot encodig  (87398, 5000)
Shape of matrix after one hot encodig  (21850, 5000)
```

In [0]:

```
X_tr, X_te = hstack_data(bow_vec_titles_tr, bow_vec_titles_te, \
                         bow_vec_essay_tr, bow_vec_essay_te,\
                         bow_vec_resource_tr, bow_vec_resource_te)
```

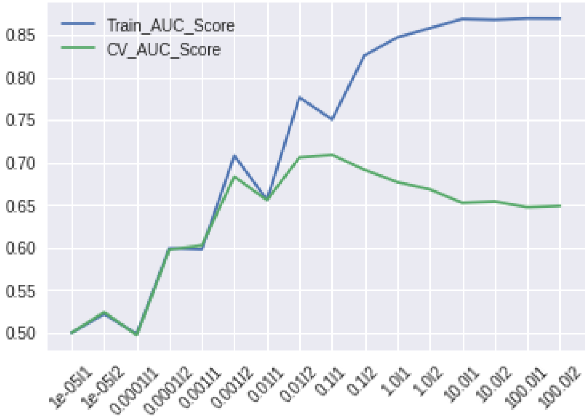
In [0]:

```
optimal_hyp(X_tr,y_train)
```

```
execution time in minutes: 2.9789655486742657
execution time in hours: 0
```

In [0]:

```
plot_auc(df1)
```



In [0]:

```
best_c,p,tr_auc_score,cv_auc_score = df1.iloc[df1.CV_AUC_Score.argmax()]
print("optimal c:",best_c,"\nRegularizer:",p,"\nCV AUC score:", cv_auc_score)
```

```
optimal c: 0.1
Regularizer: l1
CV AUC score: 0.7090231813992143
```

In [0]:

```
model = LogisticRegression(C=float(best_c), penalty=p,class_weight = "balanced")
model.fit(X_tr, y_train)
```

Out[0]:

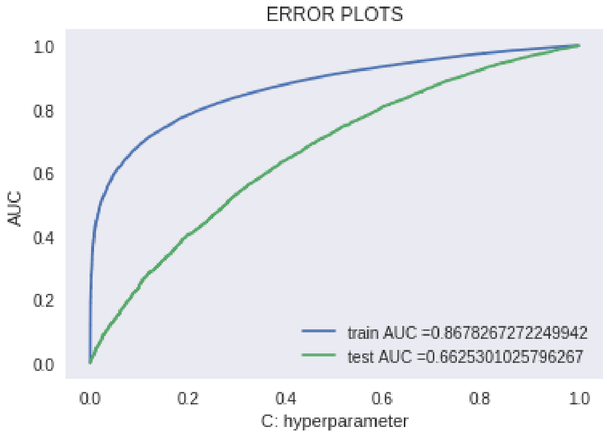
```
LogisticRegression(C=0.1, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l1', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
```

In [0]:

```
y_pred = model.predict(X_te)
```

In [0]:

```
start = time.time()
error_plot(X_tr,X_te,y_train,y_test)
end = time.time()
minutes = float((end - start)/60)
print("execution time in minutes:",minutes)
print("execution time in hours:",int(minutes/60))
```

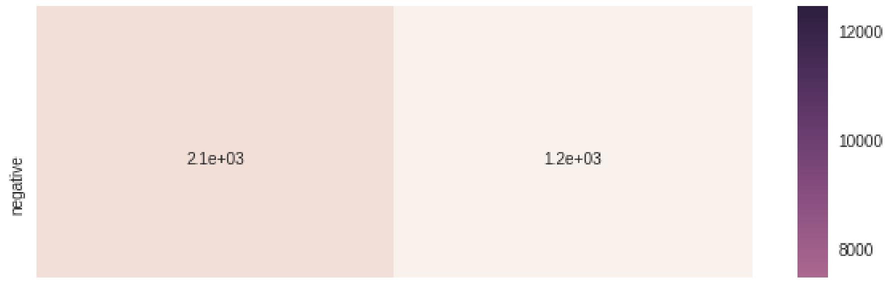


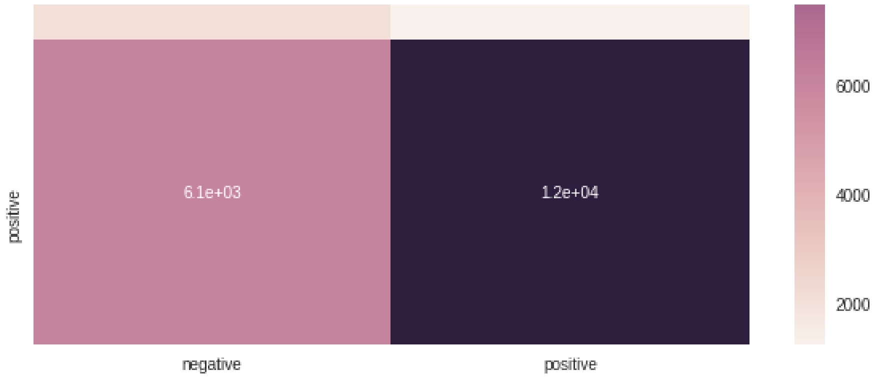
```
===================================================================================================
```

In [0]:

```
labels = ["negative","positive"]
get_confusion_matrix_values(np.array(y_test), y_pred)
```

```
Confusion Matrix :
```





```
True Positives : 2064
False Positives : 1244
True Negatives : 12431
False Negatives : 6111
```

In [0]:

```
# Printing roc auc score
y_pred = model.predict_proba(X_te)
roc_auc_score(y_true=y_test, y_score=y_pred[:,1])
```

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'))
```

```
100%|██████████| 87398/87398 [00:53<00:00, 1620.65it/s]
100%|██████████| 87398/87398 [02:34<00:00, 566.08it/s]
  1%|          | 159/21850 [00:00<00:13, 1589.26it/s]
```

```
87398
30
```

```
100%|██████████| 21850/21850 [00:13<00:00, 1615.05it/s]
100%|██████████| 21850/21850 [00:39<00:00, 553.82it/s]
```

```
21850
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'))
```

```
100%|██████████| 87398/87398 [00:02<00:00, 34562.34it/s]
100%|██████████| 87398/87398 [00:02<00:00, 42689.67it/s]
 16%|█▌        | 3414/21850 [00:00<00:00, 34138.24it/s]
```

```
87398
30
```

```
100%|██████████| 21850/21850 [00:00<00:00, 33895.31it/s]
100%|██████████| 21850/21850 [00:00<00:00, 41315.68it/s]
```

```
21850
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'))
```

```
100%|██████████| 87398/87398 [00:05<00:00, 15211.21it/s]
100%|██████████| 87398/87398 [00:06<00:00, 13503.62it/s]
  7%|▋         | 1482/21850 [00:00<00:01, 14817.65it/s]
```

```
87398
30
```

```
100%|██████████| 21850/21850 [00:01<00:00, 15172.02it/s]
100%|██████████| 21850/21850 [00:01<00:00, 13798.55it/s]
```

```
21850
30
```

In [0]:

```
X_tr, X_te = hstack_data(tfidfw2v_vec_titles_tr, tfidfw2v_vec_titles_te, \
                   tfidfw2v_vec_essay_tr, tfidfw2v_vec_essay_te,\
                   tfidfw2v_vec_resource_tr, tfidfw2v_vec_resource_te)
```

In [0]:

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

Out[0]:

```
((87398, 172), (21850, 172))
```

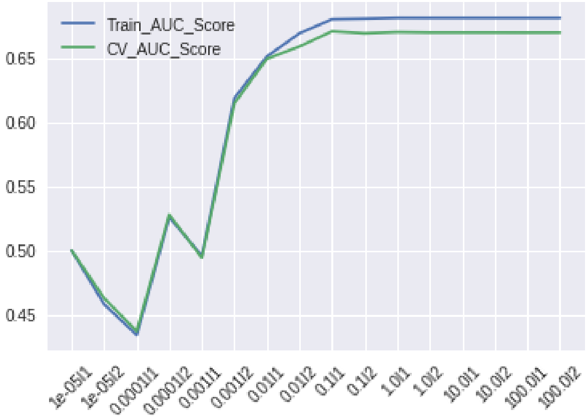
In [0]:

```
optimal_hyp(X_tr,y_train)
```

```
execution time in minutes: 12.353877480824789
execution time in hours: 0
```

In [0]:

```
plot_auc(df1)
```



In [0]:

```
best_c,p,tr_auc_score,cv_auc_score = df1.iloc[df1.CV_AUC_Score.argmax()]
print("optimal c:",best_c,"\nRegularizer:",p,"\nCV AUC score:", cv_auc_score)
```

```
optimal c: 0.1
Regularizer: l1
CV AUC score: 0.6712630379841602
```

In [0]:

```
model = LogisticRegression(C=float(best_c), penalty=p,class_weight = "balanced")
model.fit(X_tr, y_train)
```

Out[0]:

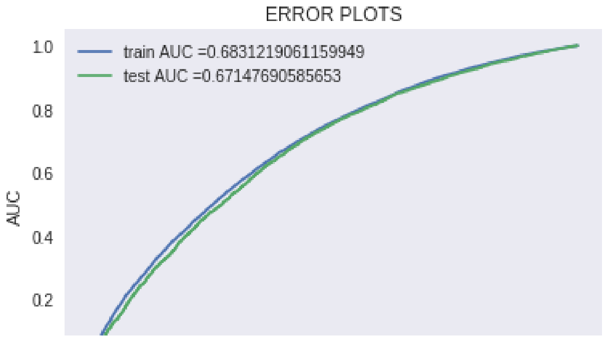
```
LogisticRegression(C=0.1, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l1', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
```

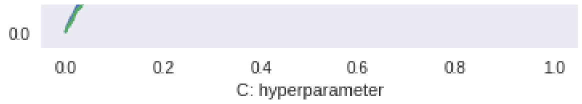
In [0]:

```
y_pred = model.predict(X_te)
```

In [0]:

```
start = time.time()
error_plot(X_tr,X_te,y_train,y_test)
end = time.time()
minutes = float((end - start)/60)
print("execution time in minutes:",minutes)
print("execution time in hours:",int(minutes/60))
```



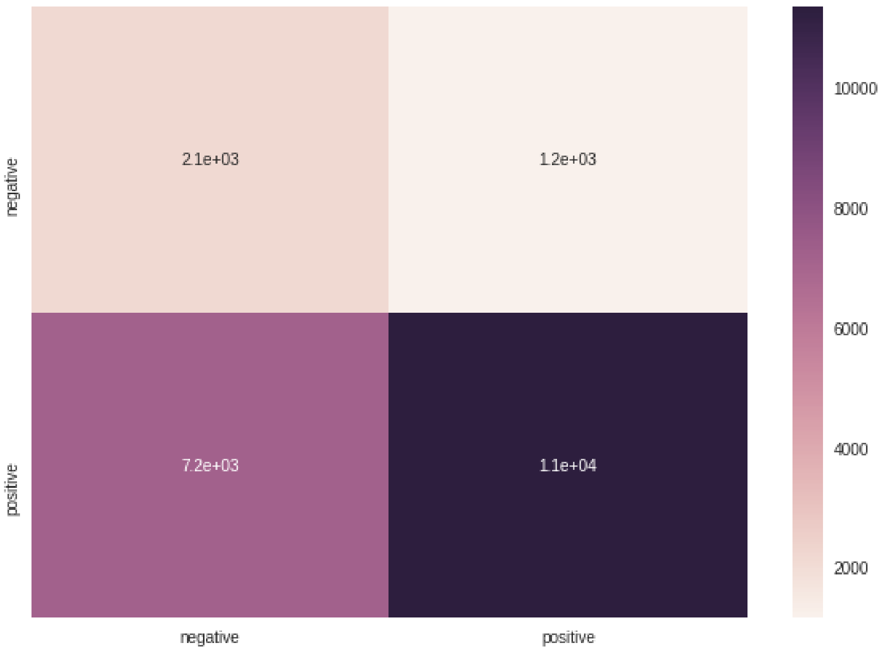


```
====================================================================================================
```

In [0]:

```
labels = ["negative","positive"]
get_confusion_matrix_values(np.array(y_test), y_pred)
```

```
Confusion Matrix :
```



```
True Positives : 2144
False Positives : 1164
True Negatives : 11331
False Negatives : 7211
```

In [0]:

```
# Printing roc auc score
y_pred = model.predict_proba(X_te)
roc_auc_score(y_true=y_test, y_score=y_pred[:,1])
```

Out[0]:

```
0.6726087361129354
```

Set 5

In [55]:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')
sid = SentimentIntensityAnalyzer()
p = project_data.copy()
sentiments = []
for i in range(p.shape[0]):
  line = p['essay'].iloc[i]
  sentiment = sid.polarity_scores(line)
  sentiments.append([sentiment['neg'], sentiment['pos'], sentiment['neu'], sentiment['compound']])
  p[['neg', 'pos', 'neu', 'compound']] = pd.DataFrame(sentiments)
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

In [0]:

```
project_data = p
```

In [58]:

```
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)
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: (87398, 27) 87398
Test: (21850, 27) 21850
```

In [60]:

```
neg_standardized_tr, neg_standardized_te = standardize_data(X_train, X_test,'neg')
print()
pos_standardized_tr, pos_standardized_te = standardize_data(X_train, X_test,'pos')
print()
neu_standardized_tr, neu_standardized_te = standardize_data(X_train, X_test,'neu')
print()
comp_standardized_tr, comp_standardized_te = standardize_data(X_train, X_test,'compound')
```

```
(87398, 1)
(21850, 1)

(87398, 1)
(21850, 1)

(87398, 1)
(21850, 1)

(87398, 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,teac
her_one_hot_tr,price_standardized_tr,\
               project_standardized_tr,title_standardized_tr,
essay_standardized_tr,resource_standardized_tr,\
               neg_standardized_tr,pos_standardized_tr, neu_standardized_tr,comp_standardized_tr,qt
y_standardized_tr)).tocsr()
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,title_standardized_te,
essay_standardized_te,resource_standardized_te,\
               neg_standardized_te,pos_standardized_te, neu_standardized_te,comp_standardized_te,qt
y_standardized_te)).tocsr()
```

In [0]:

```
X_tr = f_tr; X_te = f_te
```

In [81]:

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

Out[81]:

```
((87398, 90), (21850, 90))
```

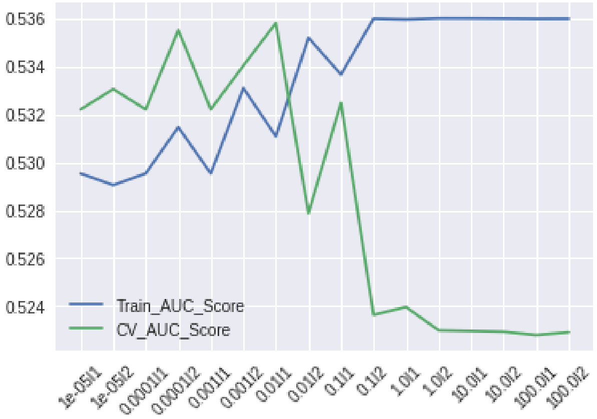
In [82]:

```
optimal_hyp(f_tr,y_train)
```

```
execution time in minutes: 0.9532934586207072
execution time in hours: 0
```

In [83]:

```
plot_auc(df1)
```



In [84]:

```
best_c,p,tr_auc_score,cv_auc_score = df1.iloc[df1.CV_AUC_Score.argmax()]
print("optimal c:",best_c,"\nRegularizer:",p,"\nCV AUC score:", cv_auc_score)
```

```
optimal c: 0.01
Regularizer: l1
CV AUC score: 0.5358045925215436
```

In [85]:

```
model = LogisticRegression(C=float(best_c), penalty=p,class_weight = "balanced")
model.fit(f_tr, y_train)
```

Out[85]:

```
LogisticRegression(C=0.01, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l1', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
```

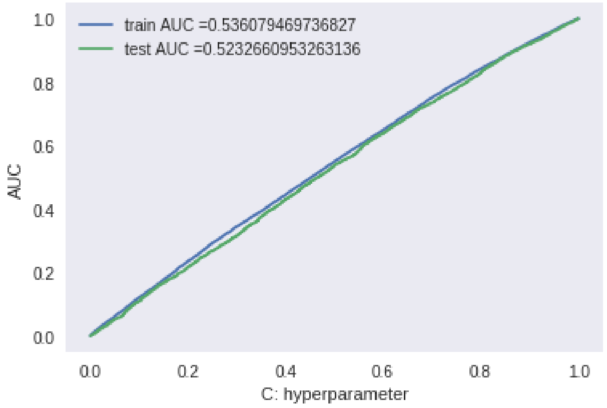
In [0]:

```
y_pred = model.predict(f_te)
```

In [87]:

```
start = time.time()
error_plot(f_tr,f_te,y_train,y_test)
end = time.time()
minutes = float((end - start)/60)
print("execution time in minutes:",minutes)
print("execution time in hours:",int(minutes/60))
```

ERROR PLOTS



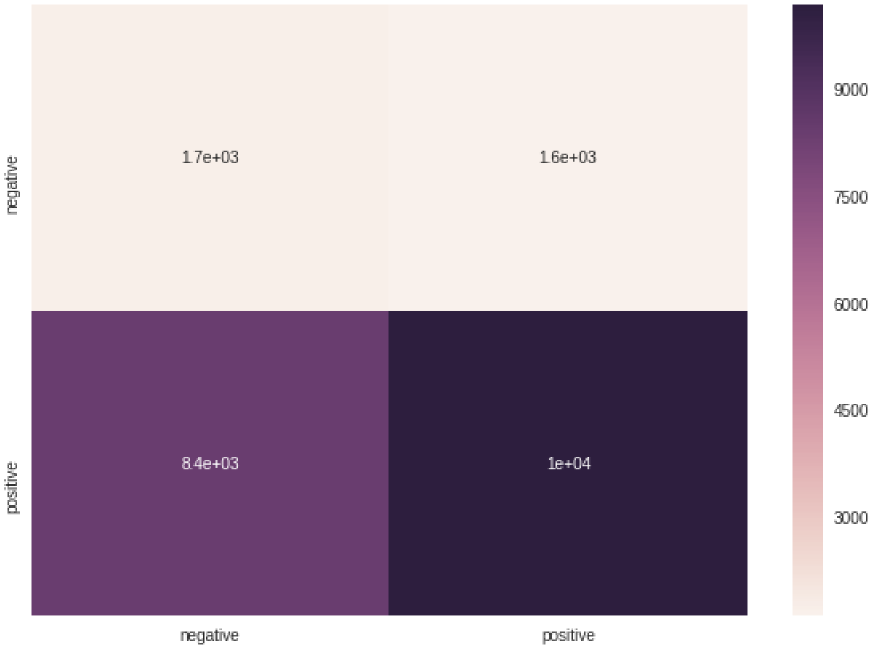
```
====================================================================================================

execution time in minutes: 3.8256627957026166
execution time in hours: 0
```

In [88]:

```
labels = ["negative","positive"]
get_confusion_matrix_values(np.array(y_test), y_pred)
```

```
Confusion Matrix :
```



```
True Positives : 1696
False Positives : 1612
True Negatives : 10174
False Negatives : 8368
```

In [89]:

```
# Printing roc auc score
y_pred = model.predict_proba(X_te)
roc_auc_score(y_true=y_test, y_score=y_pred[:,1])
```

Out[89]:

```
0.5361119766399809
```

Summary

In [0]:

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

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

x.add_row(["BOW", 'L1', 0.1, 0.6625])
x.add_row(["TFIDF", 'L2',0.1, 0.6804])
x.add_row(["W2V", 'L1',0.1, 0.6778])
x.add_row(["TFIDFW2V", 'L1',0.1, 0.6714])

print(x)'''
```

In [94]:

```
list_of_lists = []
row1 = ['L1', 0.1, 0.6625]
row2 = ['L2', 0.1, 0.6804]
row3 = ['L1', 0.1, 0.6778]
row4 = ['L1', 0.1, 0.6714]
list_of_lists.append(row1)
list_of_lists.append(row2)
list_of_lists.append(row3)
list_of_lists.append(row4)
columns = ["Regularizer", "Hyper parameter", "AUC"]
df = pd.DataFrame(list_of_lists,columns=columns,index=["BOW","TFIDF","W2V","TFIDFW2V"])
#df.index.name = "Vectorizer"
df
```

Out[94]:

	Regularizer	Hyper parameter	AUC
BOW	L1	0.1	0.6625
TFIDF	L2	0.1	0.6804
W2V	L1	0.1	0.6778
TFIDFW2V	L1	0.1	0.6714