In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

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

In [2]:
```
%cd drive/My Drive
```

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

In [3]:
```
%cd Assignments_DonorsChoose_2018
```

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

In [4]:
```
%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/2582163/4084
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_da

#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/
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/13148611/4084
project_data = project_data[cols]
#project_data.head(2)
```

In [0]:
```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).re
#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

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-i
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", "W
        if 'The' in j.split(): # this will split each of the catogory based on sp
            j=j.replace('The','') # if we have the words "The" we are going to re
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty)
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the traili
        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.co

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-i

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", "
        if 'The' in j.split(): # this will split each of the catogory based on sp
            j=j.replace('The','') # if we have the words "The" we are going to re
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty)
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the traili
        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/22898595/4
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', "y
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', '
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itsel
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because'
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'th
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all'
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than',
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "di
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn'
            '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
```

In [15]:
```
project_data['project_grade_category']=project_data['project_grade_category'].str
project_data['project_grade_category']=project_data['project_grade_category'].str
set(project_data['project_grade_category'])
```

Out[15]:
```
{'Grades_3to5', 'Grades_6to8', 'Grades_9to12', 'Grades_PreKto2'}
```

In [0]:
```
project_data['teacher_prefix']=project_data['teacher_prefix'].fillna('Mrs.')
```

In [0]:
```
#project_data = project_data[:1000]
```

In [0]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score
from collections import Counter
import matplotlib.pyplot as plt
import numpy as np
from scipy.sparse import csr_matrix
import time

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

In [19]:
```
X_train.shape,X_val.shape,X_test.shape
```

Out[19]:
```
((61178, 20), (26220, 20), (21850, 20))
```

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.
  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].value
  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.re
  print(standardized_data_test.shape)

  return standardized_data_train, standardized_data_traincv, standardized_data_te
```

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,
  else:
    vectorizer = CountVectorizer(lowercase=False, binary=True)

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

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

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

In [0]:
```
def create_dict(df,column_name):
  my_counter = Counter()
  for word in df[column_name].values:
    my_counter.update(word.split())

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

In [0]:
```
def num_hot_encode(df,column_name):
  one_hot_num_dig = pd.get_dummies(df[column_name].values)
  print("Shape of matrix after one hot encoding ",one_hot_num_dig.shape)
  return one_hot_num_dig
```

In [0]:
```
from tqdm import tqdm
def textpreprocessed(df,column_name):
  # Combining all the above stundents
  preprocessed_list = []
  # tqdm is for printing the status bar
  for sentance in tqdm(df[column_name].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed_list.append(sent.lower().strip())
  return preprocessed_list
```

In [0]:
```
from sklearn.preprocessing import Normalizer
def normalize_data(df,column_data):
  normalizer = Normalizer()
  # normalizer.fit(X_train['price'].values)
  # this will rise an error Expected 2D array, got 1D array instead:
  # array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
  # Reshape your data either using
  # array.reshape(-1, 1) if your data has a single feature
  # array.reshape(1, -1)  if it contains a single sample.
  normalizer.fit(df[column_data].values.reshape(-1,1))

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

In [26]:
```
price_standardized_tr, price_standardized_val, price_standardized_te = standardiz
print()
project_standardized_tr, project_standardized_val, project_standardized_te = stan
```

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

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

In [27]:
```
cat_one_hot_tr,cat_one_hot_val,cat_one_hot_te,cat_one_hot_list = vectorized_data(
cat_sub_one_hot_tr,cat_sub_one_hot_val,cat_sub_one_hot_te,cat_sub_one_hot_list =
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'S
pecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding  (61178, 9)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'S
pecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding  (26220, 9)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'S
pecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding  (21850, 9)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'S
pecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding  (61178, 9)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'S
pecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding  (26220, 9)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'S
pecialNeeds', '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_val,state_one_hot_te,state_one_hot_list = vectoriz
teacher_one_hot_tr,teacher_one_hot_val,teacher_one_hot_te,teacher_one_hot_list = 
```

```
['VT', 'WY', 'ND', 'MT', 'RI', 'SD', 'NE', 'AK', 'DE', 'NH', 'WV', 'HI', 'ME',
'NM', 'DC', 'KS', 'ID', 'IA', 'AR', 'CO', 'OR', 'MN', 'KY', 'MS', 'NV', 'MD',
'CT', 'TN', 'AL', 'UT', 'WI', 'VA', 'AZ', 'NJ', 'WA', 'OK', 'MA', 'OH', 'LA',
'MO', 'IN', 'PA', 'MI', 'SC', 'GA', 'IL', 'NC', 'FL', 'NY', 'TX', 'CA']
Shape of matrix after one hot encoding  (61178, 51)
['VT', 'WY', 'ND', 'MT', 'RI', 'SD', 'NE', 'AK', 'DE', 'NH', 'WV', 'HI', 'ME',
'NM', 'DC', 'KS', 'ID', 'IA', 'AR', 'CO', 'OR', 'MN', 'KY', 'MS', 'NV', 'MD',
'CT', 'TN', 'AL', 'UT', 'WI', 'VA', 'AZ', 'NJ', 'WA', 'OK', 'MA', 'OH', 'LA',
'MO', 'IN', 'PA', 'MI', 'SC', 'GA', 'IL', 'NC', 'FL', 'NY', 'TX', 'CA']
Shape of matrix after one hot encoding  (26220, 51)
['VT', 'WY', 'ND', 'MT', 'RI', 'SD', 'NE', 'AK', 'DE', 'NH', 'WV', 'HI', 'ME',
'NM', 'DC', 'KS', 'ID', 'IA', 'AR', 'CO', 'OR', 'MN', 'KY', 'MS', 'NV', 'MD',
'CT', 'TN', 'AL', 'UT', 'WI', 'VA', 'AZ', 'NJ', 'WA', 'OK', 'MA', 'OH', 'LA',
'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  (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 [29]:
```
grade_dict = create_dict(X_train,'project_grade_category')
grade_one_hot_tr,grade_one_hot_val,grade_one_hot_te,grade_one_hot_list = vectoriz
```

```
['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 [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')
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)
```

[Task-1] Apply Support Vector Machines(SGDClassifier with hinge loss: Linear SVM) on these feature sets

In [0]:
```
from scipy.sparse import hstack

f_tr = hstack((one_hot_num_dig_tr,grade_one_hot_tr,state_one_hot_tr,cat_one_hot_t
f_cr = hstack((one_hot_num_dig_val,grade_one_hot_val,state_one_hot_val,cat_one_ho
f_te = hstack((one_hot_num_dig_te,grade_one_hot_te,state_one_hot_te,cat_one_hot_t

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

In [32]:
```
feature_list_x = ['dig_0','dig_1']+grade_one_hot_list+state_one_hot_list+cat_one_
len(feature_list_x)
```

Out[32]: 82

In [0]:
```
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import SGDClassifier
import time
def optimal_hyp(X_tr,y_train,X_cr,y_val):
  global df1
  global clf
  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 = SGDClassifier( class_weight='balanced', alpha=c, penalty=p, loss='h
        clf.fit(X_tr, y_train)
        tr_score = roc_auc_score(y_true=np.array(y_train), y_score=clf.decision_f
        cv_score = roc_auc_score(y_true=np.array(y_val), y_score=clf.decision_fun
        #print(tr_score)
        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]:
```
#https://datascience.stackexchange.com/questions/28493/confusion-matrix-get-items
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)
    #tn, fp, fn, tp = cm.ravel()
    #print(tn,fp,fn,tp)
    #print("Confusion Matrix : ")
    plt.figure(figsize=(10,7))
    sns.heatmap(df, annot=True,fmt = "d")
    plt.title("Confusion Matrix",fontsize=20)
    plt.xlabel("Predicted Label",fontsize=15)
    plt.ylabel("True Label",fontsize=15)
    plt.show()
    TN, FP, FN, TP = 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]:
```
#https://machinelearningmastery.com/calibrated-classification-model-in-scikit-lea
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.htm
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix
import seaborn as sns
def error_plot(X_train,X_te,y_train,y_test):
  global y_train_pred1
  model =  SGDClassifier(class_weight='balanced', alpha=float(best_c), penalty=p,
  model.fit(X_train, y_train)
  #pred = model.predict(X_train)
  y_train_pred = model.decision_function(X_train)
  y_train_pred1 = model.predict(X_train)

  #pred = model.predict(X_te)
  y_test_pred = model.decision_function(X_te)

  #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("FPR")
  plt.ylabel("TPR")
  plt.title("ROC Curve")
  plt.grid()
  plt.show()
  print("="*100)
```

In [36]:
```
'''# 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",
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions'''
```

Out[36]:
```
'# we are writing our own function for predict, with defined thresould\n# we wi
ll pick a threshold that will give the least fpr\ndef predict(proba, threshoul
d, fpr, tpr):\n    \n    t = threshould[np.argmax(fpr*(1-tpr))]\n    \n    # (t
pr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high\n
\n    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshol
d", np.round(t,3))\n    predictions = []\n    for i in proba:\n        if i>=
t:\n            predictions.append(1)\n        else:\n            predictions.a
ppend(0)\n    return predictions'
```

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,figsize=(15,7),style='.-', ms=13)
  ax.set_xticklabels(df1[['C', 'Penalty']].apply(lambda x: ''.join(x), axis=1))
  ax.set_ylabel("AUC Score")
  ax.set_xlabel("Hyperparameters: (C, Penalty)")
  ax.set_xticklabels(df1[['C', 'Penalty']].apply(lambda x: ','.join(x), axis=1))
  ax
```

In [0]:
```
# https://matplotlib.org/examples/mplot3d/2dcollections3d_demo.html
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt


def plot_3D(x,y,z1,z2):
  fig = plt.figure(figsize=(10,5))
  ax = fig.gca(projection='3d')
  ax.scatter3D(x, y, z1, color="r", label='Train')
  ax.scatter3D(x, y, z2, color="b", label='CV')

  # Make legend, set axes limits and labels
  ax.legend()
  ax.set_xlabel('C',weight='bold')
  ax.set_ylabel('Penalty',weight='bold')
  ax.set_zlabel('AUC',rotation=90,weight='bold')

  # Customize the view angle so it's easier to see that the scatter points lie
  # on the plane y=0
  ax.view_init(elev=20., azim=-35)

  plt.show()
```

In [0]:
```
def plot_heatmap():
  p = df1.Penalty.nunique()
  c = df1.C.nunique()
  r1=[];s1=[]
  count = 0
  for i in range(p):
    r2=[];s2=[]
    for j in range(c):
      d1 = df1.query('C=="{}" and Penalty=="{}"'.format(str(df1['C'][count]),str(
      d2 = df1.query('C=="{}" and Penalty=="{}"'.format(str(df1['C'][count]),str(
      count = count+1
      r2.append(d1)
      s2.append(d2)
    r1.append(r2)
    s1.append(s2)
  ################ Train AUC ##################
  df = pd.DataFrame(r1, columns=list(map(lambda x: 10 ** x, np.arange(-5, 3,dtype
  sns.heatmap(df, annot=True,fmt='.4g')
  plt.rcParams["axes.labelweight"] = "bold"
  plt.rcParams["axes.titleweight"] = "bold"
  plt.xlabel('Regularizer',fontsize=12)
  plt.ylabel('C',rotation=0,fontsize=12)
  plt.title("Train AUC Score")
  plt.show()
  ################ CV AUC ####################
  df = pd.DataFrame(s1, columns=list(map(lambda x: 10 ** x, np.arange(-5, 3,dtype
  sns.heatmap(df, annot=True,fmt='.4g')
  plt.rcParams["axes.labelweight"] = "bold"
  plt.rcParams["axes.titleweight"] = "bold"
  plt.xlabel('Regularizer',fontsize=12)
  plt.ylabel('C',rotation=0,fontsize=12)
  plt.title("CV AUC Score")
  plt.show()
```

In [0]:
```
BLUE = '34m'
def display_colored_text(color, text):
  colored_text = f"\033[{color}{text}\033[00m"
  return colored_text
```

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

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

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

In [0]:
```
def bow_vec(preprocessed_data_tr,preprocessed_data_val,preprocessed_data_te):
  global vectorizer_bow
  vectorizer_bow = CountVectorizer(min_df=10)
  text_bow_tr = vectorizer_bow.fit_transform(preprocessed_data_tr)
  print("Shape of matrix after one hot encodig ",text_bow_tr.shape)
  vectorizer_list = vectorizer_bow.get_feature_names()
  text_bow_val = vectorizer_bow.transform(preprocessed_data_val)
  print("Shape of matrix after one hot encodig ",text_bow_val.shape)

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

In [0]:
```
bow_vec_essay_tr,bow_vec_essay_val,bow_vec_essay_te,bow_vec_essay_list = bow_vec(
bow_vec_titles_tr,bow_vec_titles_val,bow_vec_titles_te,bow_vec_titles_list = bow_
bow_vec_resource_tr,bow_vec_resource_val,bow_vec_resource_te,bow_vec_resource_lis
```

```
100%|██████████| 61178/61178 [00:37<00:00, 1632.51it/s]
100%|██████████| 26220/26220 [00:16<00:00, 1630.63it/s]
100%|██████████| 21850/21850 [00:13<00:00, 1635.85it/s]

Shape of matrix after one hot encodig  (61178, 13127)
Shape of matrix after one hot encodig  (26220, 13127)

  6%|▌         | 3388/61178 [00:00<00:01, 33876.56it/s]

Shape of matrix after one hot encodig  (21850, 13127)

100%|██████████| 61178/61178 [00:01<00:00, 34151.32it/s]
100%|██████████| 26220/26220 [00:00<00:00, 33840.39it/s]
100%|██████████| 21850/21850 [00:00<00:00, 34105.25it/s]

Shape of matrix after one hot encodig  (61178, 2270)
Shape of matrix after one hot encodig  (26220, 2270)

  3%|▎          | 1555/61178 [00:00<00:03, 15547.90it/s]

Shape of matrix after one hot encodig  (21850, 2270)

100%|██████████| 61178/61178 [00:04<00:00, 15227.36it/s]
100%|██████████| 26220/26220 [00:01<00:00, 15236.70it/s]
100%|██████████| 21850/21850 [00:01<00:00, 15214.15it/s]

Shape of matrix after one hot encodig  (61178, 4352)
Shape of matrix after one hot encodig  (26220, 4352)
Shape of matrix after one hot encodig  (21850, 4352)
```

In [0]:
```
X_tr, X_cr, X_te = hstack_data(bow_vec_titles_tr, bow_vec_titles_val, bow_vec_tit
                   bow_vec_essay_tr, bow_vec_essay_val, bow_vec_essay_te,\
                   bow_vec_resource_tr, bow_vec_resource_val, bow_vec_resource_te
```

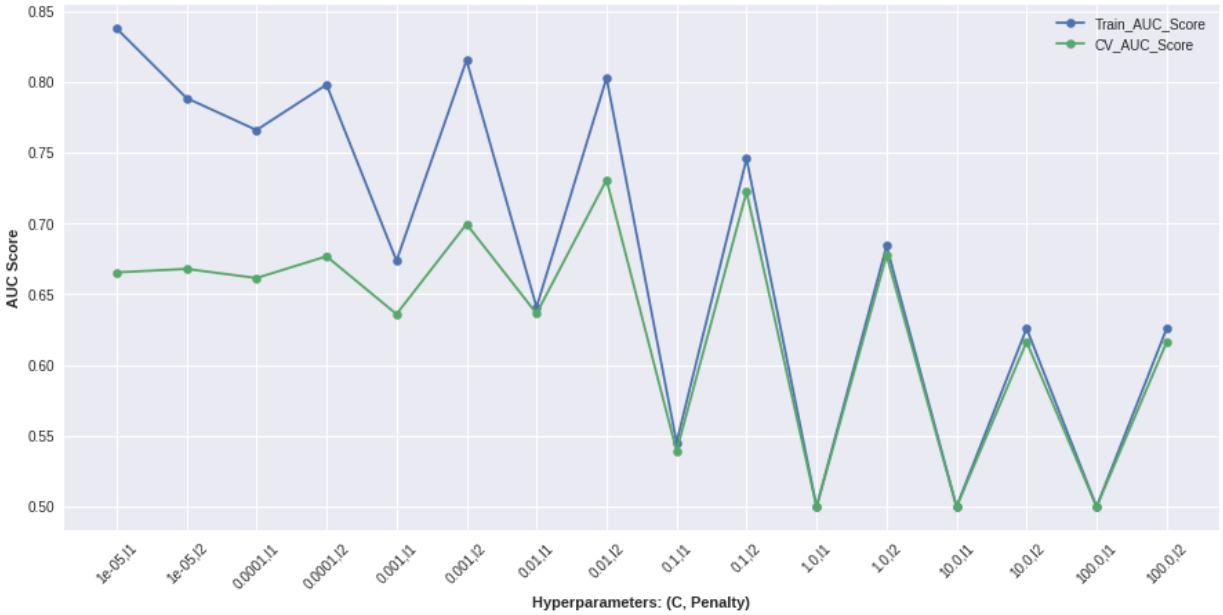
In [0]:
```
X_tr.shape,X_cr.shape,X_te.shape
```

Out[559]:
```
((61178, 19831), (26220, 19831), (21850, 19831))
```

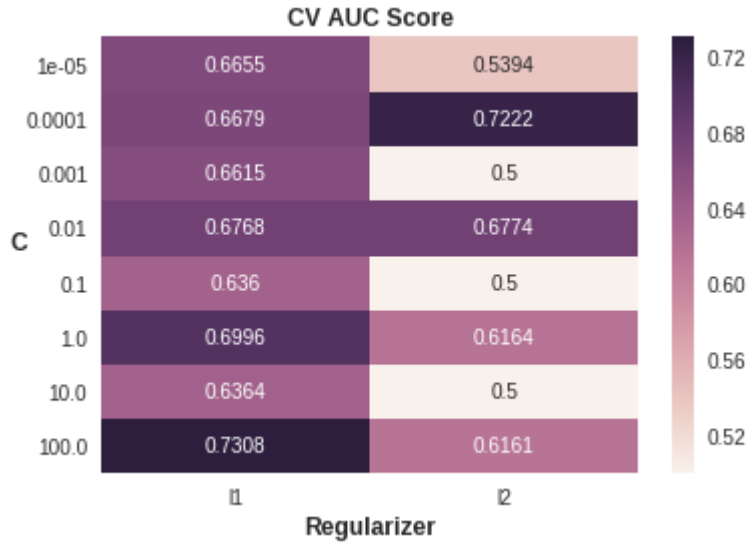
In [0]:
```
optimal_hyp(X_tr,y_train,X_cr,y_val)
```

```
execution time in minutes: 0.1454118847846985
execution time in hours: 0
```

In [0]:
```
plot_auc(df1)
```



In [0]:
```
plot_heatmap()
```



In [0]:
```
feature_list = feature_list_x.copy()
feature_names = [*feature_list , *bow_vec_titles_list, *bow_vec_essay_list, *bow_
print(len(feature_names))
```

```
19831
```

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: 0.01
Regularizer: l2
CV AUC score: 0.730801245294767
```

In [0]:
```
show_most_informative_features(feature_names,clf,n=10)
```

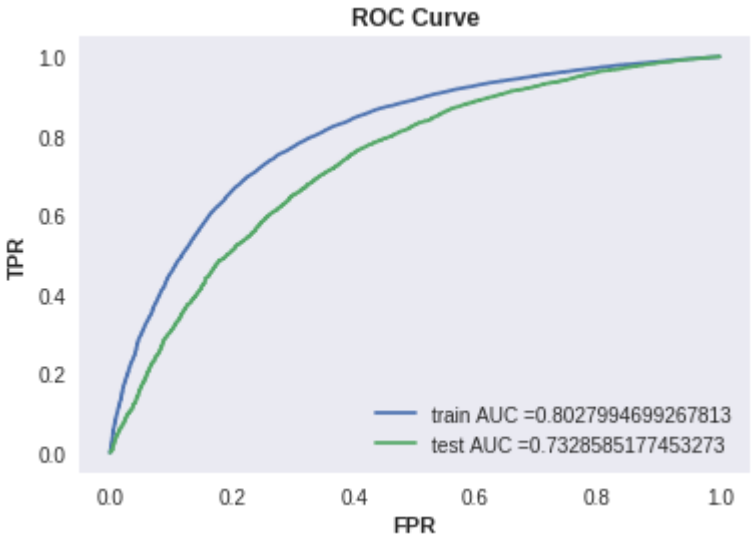
```
                Positive                          Negative
 ============================================================
        -0.0011 materials                0.0026  students
        -0.0010 price                    0.0013  books
        -0.0007 supplies                 0.0010  reading
        -0.0004 supplies                 0.0009  use
        -0.0004 materials                0.0009  teacher_number_of_previously_po
sted_projects
        -0.0003 hands                    0.0007  classroom
        -0.0003 art                      0.0007  read
        -0.0003 learning                 0.0006  technology
        -0.0002 dig_0                    0.0006  would
        -0.0002 science                  0.0005  day
```

In [0]:
```
model = SGDClassifier(class_weight='balanced', alpha=float(best_c), penalty=p, lo
model.fit(X_tr, y_train)
```

Out[566]:
```
SGDClassifier(alpha=0.01, average=False, class_weight='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=42, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False)
```

In [0]:
```
y_pred = model.predict(X_te)
```

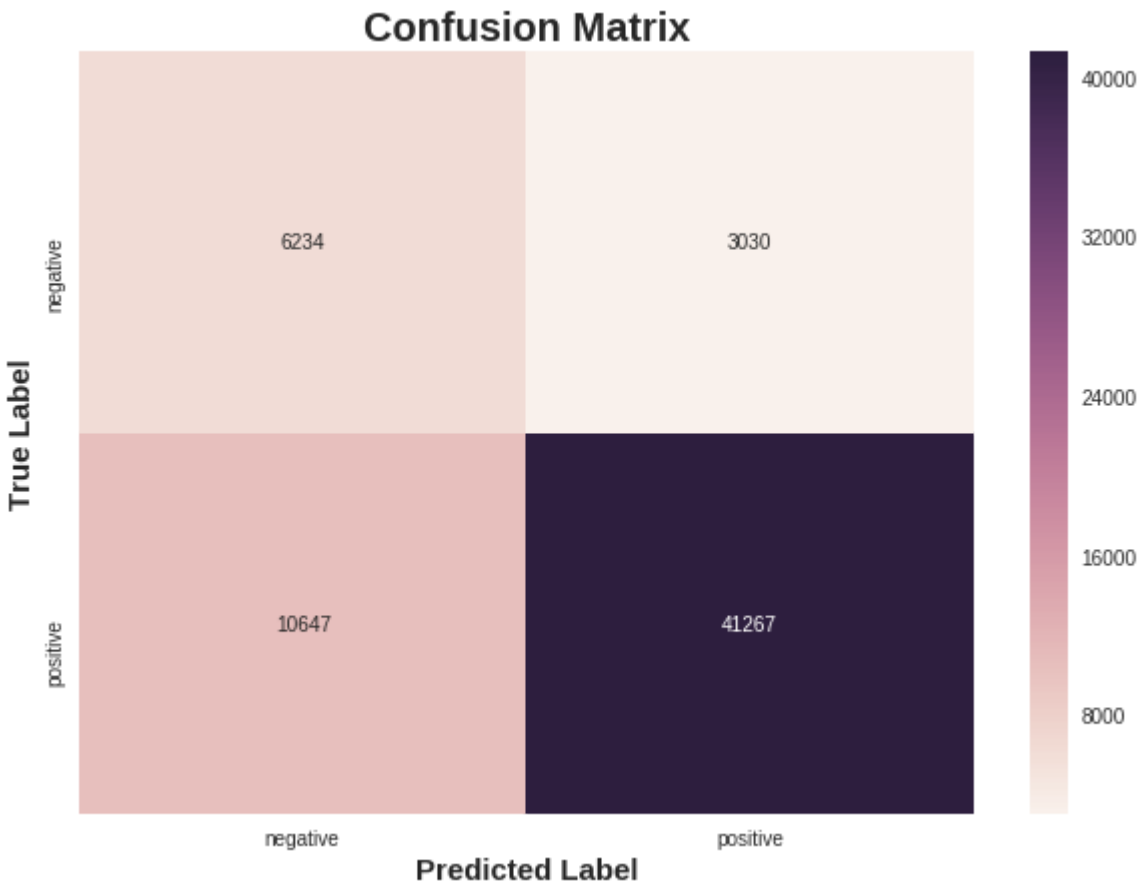
In [0]:
```
start = time.time()
error_plot(X_tr,X_te,y_train,y_test)
end = time.time()
minutes = float((end - start)/60)
print("execution time in minutes:",minutes)
print("execution time in hours:",int(minutes/60))
```




```
================================================================================
====================
execution time in minutes: 0.011029434204101563
execution time in hours: 0
```

confusion matrix for train data

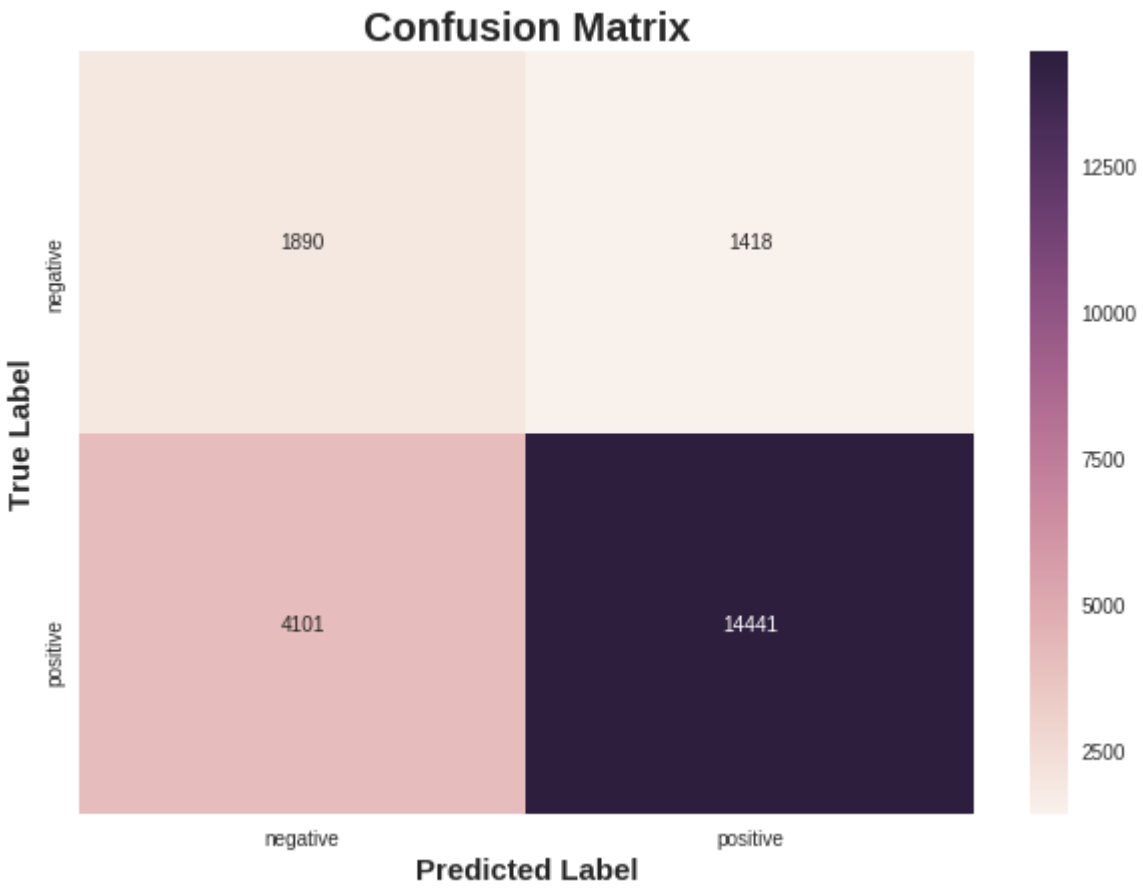
In [0]:
```
labels = ["negative","positive"]
get_confusion_matrix_values(np.array(y_train), y_train_pred1)
```




```
True Positives : 41267
False Positives : 3030
True Negatives : 6234
False Negatives : 10647
```

confusion matrix for test data

In [0]:
```
labels = ["negative","positive"]
get_confusion_matrix_values(np.array(y_test), y_pred)
```




```
True Positives : 14441
False Positives : 1418
True Negatives : 1890
False Negatives : 4101
```

In [0]:
```
# Printing roc auc score
y_pred = model.decision_function(X_te)
roc_auc_score(y_true=y_test, y_score=y_pred)
```

Out[380]:
```
0.7289469920701614
```

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

In [0]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
def tfidf_vec(preprocessed_data_tr,preprocessed_data_val,preprocessed_data_te):
  global vectorizer_tfidf
  vectorizer_tfidf = TfidfVectorizer(min_df=10)
  text_tfidf_tr = vectorizer_tfidf.fit_transform(preprocessed_data_tr)
  vectorizer_tf = vectorizer_tfidf.get_feature_names()
  print("Shape of matrix after one hot encodig ",text_tfidf_tr.shape)

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

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

In [0]:
```
tfidf_vec_essay_tr,tfidf_vec_essay_val,tfidf_vec_essay_te,tfidf_vec_essay_list = 
tfidf_vec_titles_tr,tfidf_vec_titles_val,tfidf_vec_titles_te,tfidf_vec_titles_lis
tfidf_vec_resource_tr,tfidf_vec_resource_val,tfidf_vec_resource_te,tfidf_vec_reso
```

```
100%|██████████| 61178/61178 [00:37<00:00, 1614.04it/s]
100%|██████████| 26220/26220 [00:16<00:00, 1624.80it/s]
100%|██████████| 21850/21850 [00:13<00:00, 1640.09it/s]

Shape of matrix after one hot encodig  (61178, 13174)
Shape of matrix after one hot encodig  (26220, 13174)

  5%|▌         | 3325/61178 [00:00<00:01, 33244.72it/s]

Shape of matrix after one hot encodig  (21850, 13174)

100%|██████████| 61178/61178 [00:01<00:00, 34218.14it/s]
100%|██████████| 26220/26220 [00:00<00:00, 34045.14it/s]
100%|██████████| 21850/21850 [00:00<00:00, 34076.63it/s]

Shape of matrix after one hot encodig  (61178, 2292)
Shape of matrix after one hot encodig  (26220, 2292)

  3%|▎          | 1541/61178 [00:00<00:03, 15404.84it/s]

Shape of matrix after one hot encodig  (21850, 2292)

100%|██████████| 61178/61178 [00:04<00:00, 15292.60it/s]
100%|██████████| 26220/26220 [00:01<00:00, 15165.00it/s]
100%|██████████| 21850/21850 [00:01<00:00, 15150.08it/s]

Shape of matrix after one hot encodig  (61178, 4369)
Shape of matrix after one hot encodig  (26220, 4369)
Shape of matrix after one hot encodig  (21850, 4369)
```

In [0]:
```
feature_list = feature_list_x.copy()
feature_names = [*feature_list , *tfidf_vec_titles_list, *tfidf_vec_essay_list, *
print(len(feature_names))
```

```
19917
```

In [0]:
```
X_tr, X_cr, X_te = hstack_data(tfidf_vec_titles_tr, tfidf_vec_titles_val, tfidf_v
                   tfidf_vec_essay_tr, tfidf_vec_essay_val, tfidf_vec_essay_te,\
                   tfidf_vec_resource_tr, tfidf_vec_resource_val, tfidf_vec_resou
```

In [0]:
```
X_tr.shape,X_cr.shape,X_te.shape
```

Out[617]:
```
((61178, 19917), (26220, 19917), (21850, 19917))
```

In [0]:
```
optimal_hyp(X_tr,y_train,X_cr,y_val)
```

```
execution time in minutes: 0.14380438327789308
execution time in hours: 0
```

In [0]:
```
plot_auc(df1)
```

In [0]:
```
plot_heatmap()
```

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: 0.001
Regularizer: l2
CV AUC score: 0.7120557665638355
```

In [0]:
```
show_most_informative_features(feature_names,clf,n=10)
```

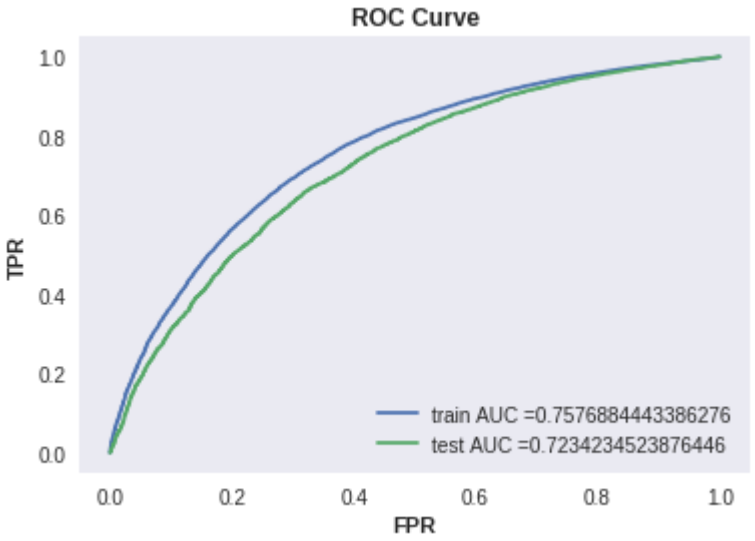
```
                Positive                          Negative
 ============================================================
        -0.0010 price                     0.0009  teacher_number_of_previously_po
sted_projects
        -0.0002 dig_0                     0.0003  Literacy_Language
        -0.0001 TX                        0.0002  dig_1
        -0.0001 Math_Science              0.0001  Grades_3to5
        -0.0001 SpecialNeeds              0.0001  books
        -0.0001 SpecialNeeds              0.0000  CA
        -0.0001 supplies                  0.0000  reading
        -0.0001 materials                 0.0000  books
        -0.0001 AppliedLearning           0.0000  chromebooks
        -0.0001 materials                 0.0000  chromebooks
```

In [0]:
```
model = SGDClassifier(class_weight='balanced', alpha=float(best_c), penalty=p, lo
model.fit(X_tr, y_train)
```

Out[623]:
```
SGDClassifier(alpha=0.001, average=False, class_weight='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=42, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False)
```

In [0]:
```
y_pred = model.predict(X_te)
```

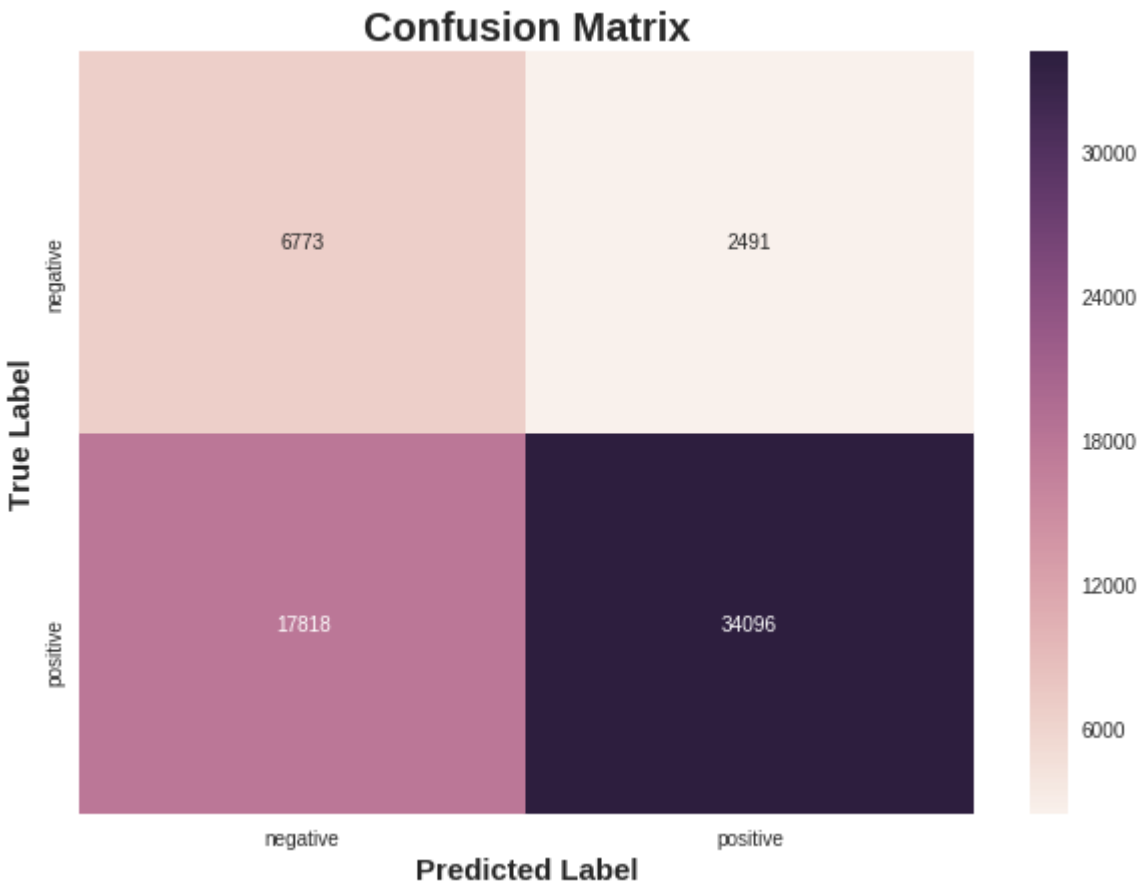
In [0]:
```
start = time.time()
error_plot(X_tr,X_te,y_train,y_test)
end = time.time()
minutes = float((end - start)/60)
print("execution time in minutes:",minutes)
print("execution time in hours:",int(minutes/60))
```




```
================================================================================
====================
execution time in minutes: 0.012538317839304607
execution time in hours: 0
```

confusion matrix for train data

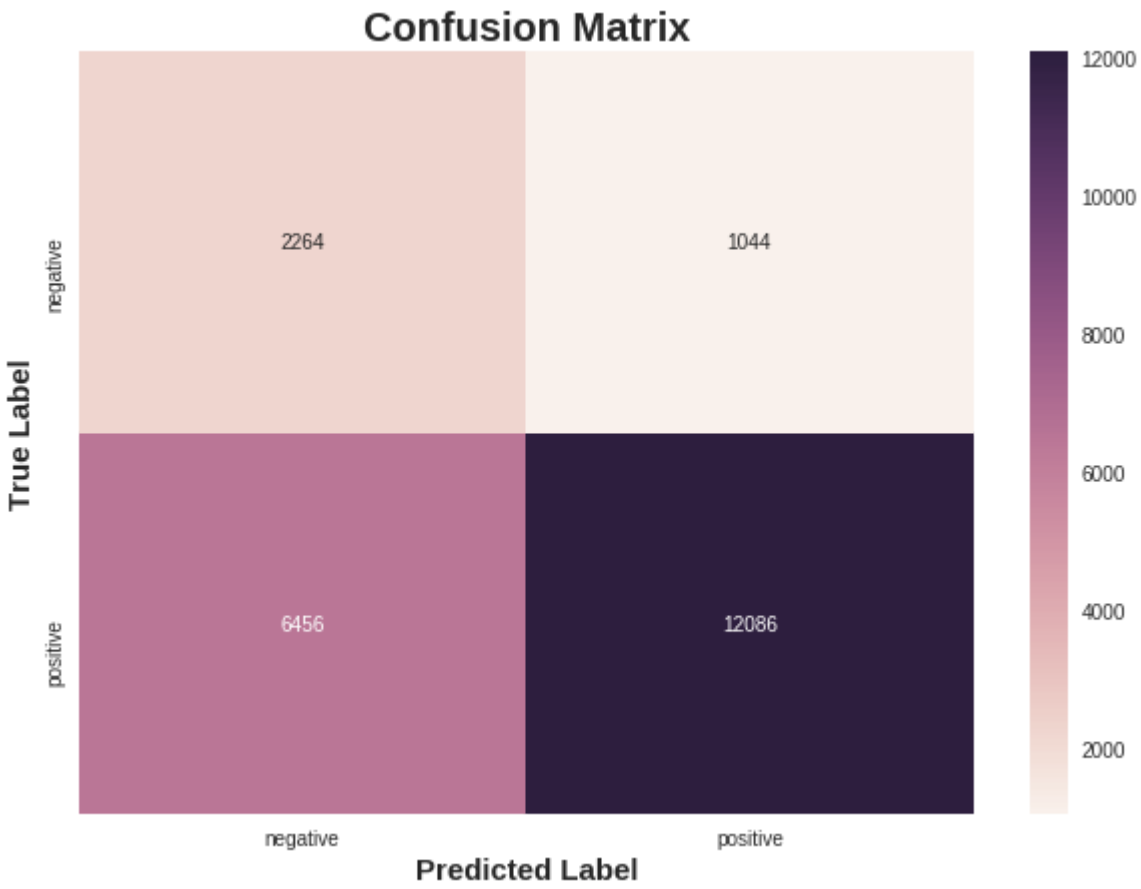
In [0]:
```
labels = ["negative","positive"]
get_confusion_matrix_values(np.array(y_train), y_train_pred1)
```




```
True Positives : 34096
False Positives : 2491
True Negatives : 6773
False Negatives : 17818
```

confusion matrix for test data

In [0]:
```
labels = ["negative","positive"]
get_confusion_matrix_values(np.array(y_test), y_pred)
```




```
True Positives : 12086
False Positives : 1044
True Negatives : 2264
False Negatives : 6456
```

In [0]:
```
# Printing roc auc score
y_pred = model.decision_function(X_te)
roc_auc_score(y_true=y_test, y_score=y_pred)
```

Out[628]:
```
0.7234234523876446
```

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

In [0]:
```
#%cd -
```

In [0]:
```
#%cd Assignments_DonorsChoose_2018
# stronging variables into pickle files python: http://www.jessicayung.com/how-to
# 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 t
  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 [772]:
```
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 [00:37<00:00, 1623.16it/s]
100%|██████████| 61178/61178 [00:16<00:00, 3703.48it/s]
  1%|          | 160/21850 [00:00<00:13, 1588.53it/s]

61178
30

100%|██████████| 21850/21850 [00:13<00:00, 1622.09it/s]
100%|██████████| 21850/21850 [00:05<00:00, 3701.41it/s]
  1%|          | 160/26220 [00:00<00:16, 1592.29it/s]

21850
30

100%|██████████| 26220/26220 [00:16<00:00, 1621.29it/s]
100%|██████████| 26220/26220 [00:07<00:00, 3613.27it/s]

26220
30
```

In [773]:
```
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, 34026.09it/s]
100%|██████████| 61178/61178 [00:00<00:00, 86912.87it/s]
 15%|██         | 3386/21850 [00:00<00:00, 33857.93it/s]

61178
30

100%|██████████| 21850/21850 [00:00<00:00, 33970.71it/s]
100%|██████████| 21850/21850 [00:00<00:00, 83809.75it/s]
 13%|█          | 3439/26220 [00:00<00:00, 34387.74it/s]

21850
30

100%|██████████| 26220/26220 [00:00<00:00, 34417.28it/s]
100%|██████████| 26220/26220 [00:00<00:00, 86640.26it/s]

26220
30
```

In [774]:
```
avgw2v_vec_resource_tr = avg_w2v(textpreprocessed(X_train,'project_resource_summa
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 [00:03<00:00, 15363.55it/s]
100%|██████████| 61178/61178 [00:01<00:00, 35414.83it/s]
  7%|▊          | 1475/21850 [00:00<00:01, 14747.10it/s]

61178
30

100%|██████████| 21850/21850 [00:01<00:00, 15219.00it/s]
100%|██████████| 21850/21850 [00:00<00:00, 34917.44it/s]
  6%|▌          | 1466/26220 [00:00<00:01, 14655.75it/s]

21850
30

100%|██████████| 26220/26220 [00:01<00:00, 15127.62it/s]
100%|██████████| 26220/26220 [00:00<00:00, 35339.89it/s]

26220
30
```

In [0]:
```
X_tr, X_cr, X_te = hstack_data(avgw2v_vec_titles_tr, avgw2v_vec_titles_val, avgw2v
                   avgw2v_vec_essay_tr, avgw2v_vec_essay_val, avgw2v_vec_essay_te
                   avgw2v_vec_resource_tr, avgw2v_vec_resource_val, avgw2v_vec_re
```

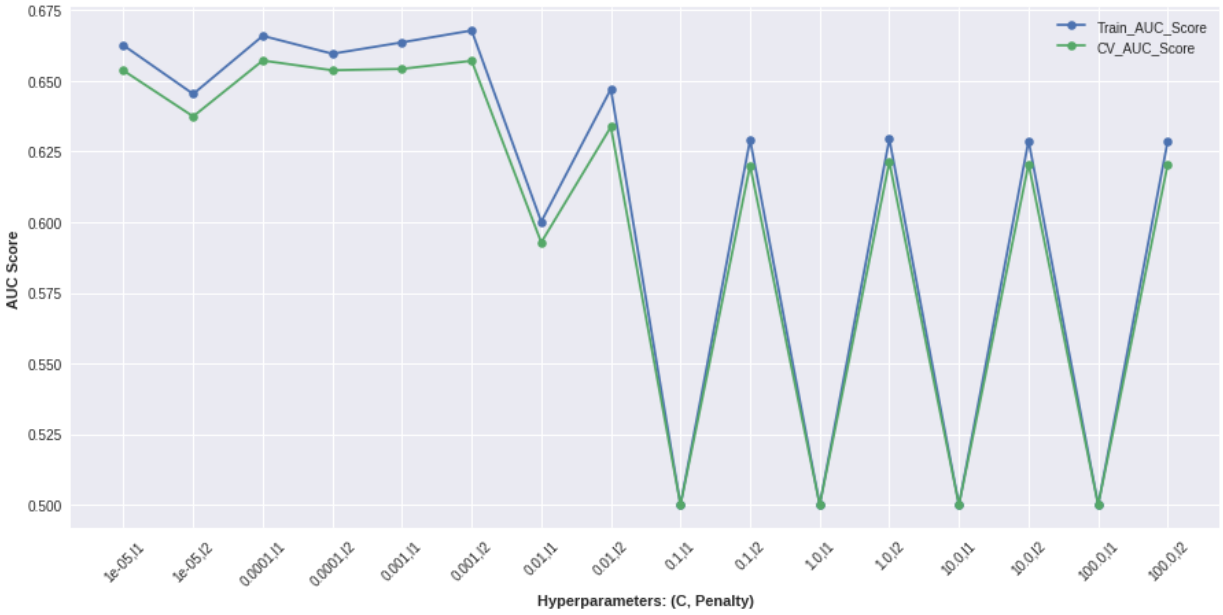
In [776]:
```
X_tr.shape,X_cr.shape,X_te.shape
```

Out[776]:
```
((61178, 172), (26220, 172), (21850, 172))
```

In [777]:
```
optimal_hyp(X_tr,y_train,X_cr,y_val)
```

```
execution time in minutes: 0.12700416644414267
execution time in hours: 0
```

In [778]:
```
plot_auc(df1)
```



In [779]:
```
plot_heatmap()
```

In [780]:
```
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)
```

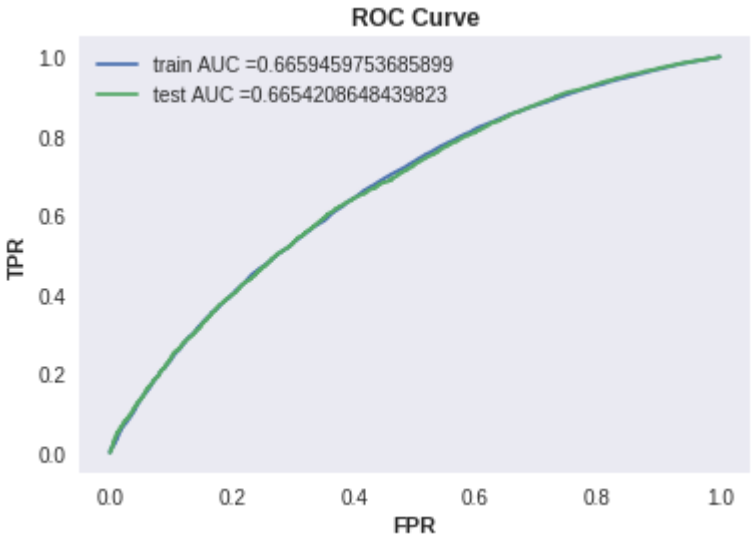
```
optimal c: 0.0001
Regularizer: l1
CV AUC score: 0.6572138001301899
```

In [781]:
```
model = SGDClassifier(class_weight='balanced', alpha=float(best_c), penalty=p, lo
model.fit(X_tr, y_train)
```

Out[781]:
```
SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l1',
       power_t=0.5, random_state=42, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False)
```

In [0]:
```
y_pred = model.predict(X_te)
```

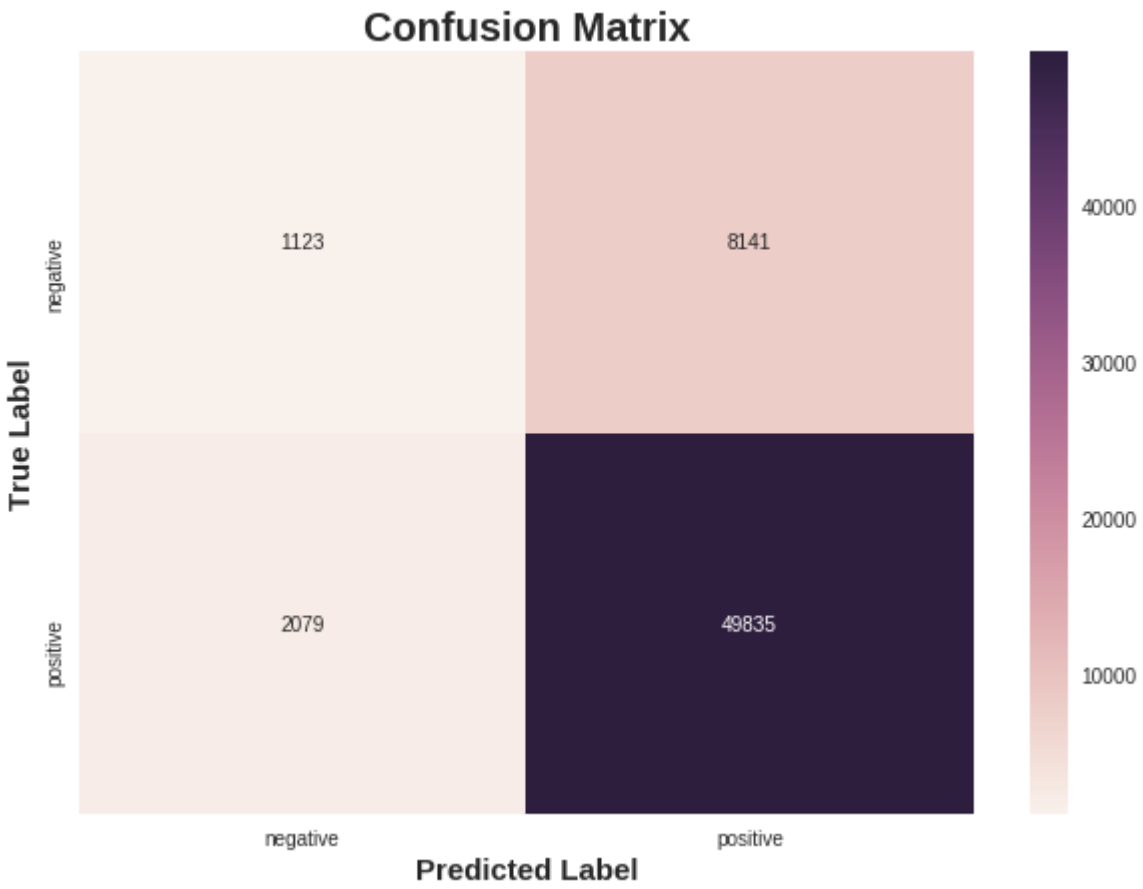
In [783]:
```
start = time.time()
error_plot(X_tr,X_te,y_train,y_test)
end = time.time()
minutes = float((end - start)/60)
print("execution time in minutes:",minutes)
print("execution time in hours:",int(minutes/60))
```




```
==========================================================================
====================
execution time in minutes: 0.01321113109588623
execution time in hours: 0
```

confusion matrix for train data

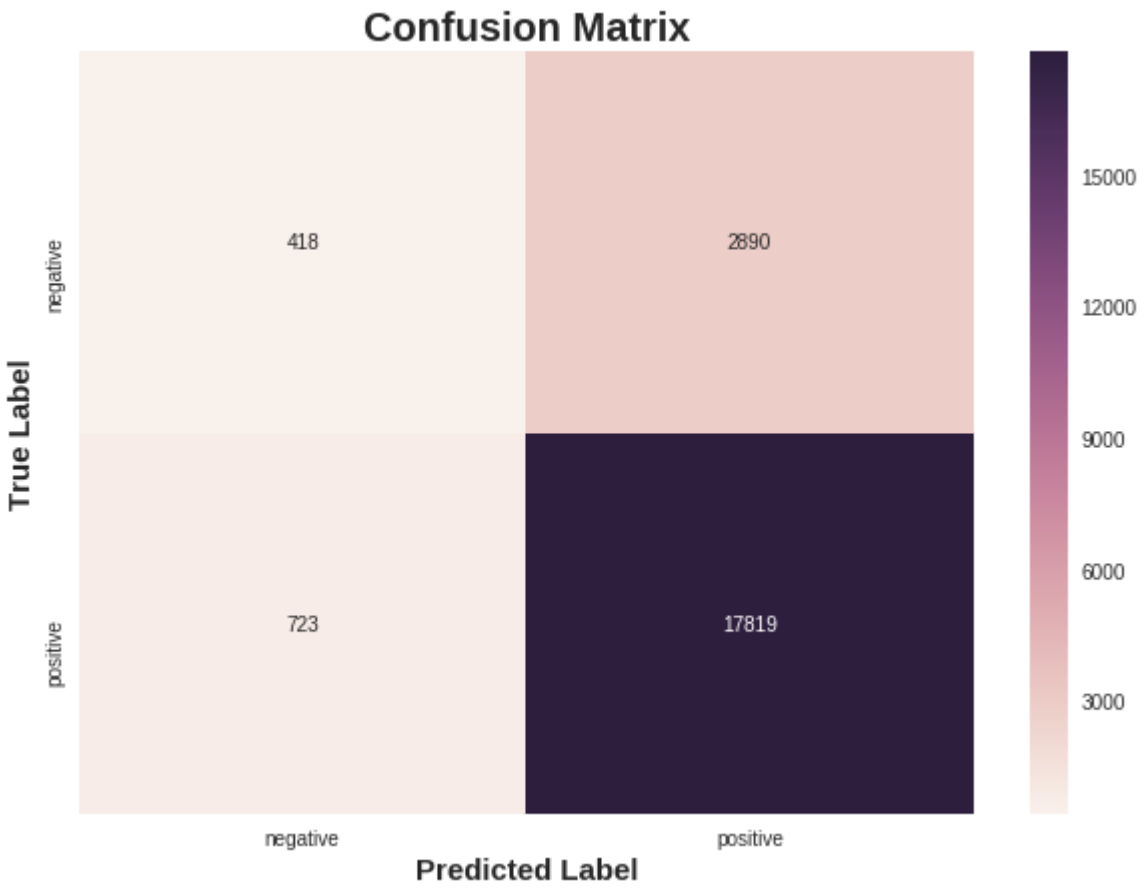
In [784]:
```
labels = ["negative","positive"]
get_confusion_matrix_values(np.array(y_train), y_train_pred1)
```




```
True Positives : 49835
False Positives : 8141
True Negatives : 1123
False Negatives : 2079
```

confusion matrix for test data

In [785]:
```
labels = ["negative","positive"]
get_confusion_matrix_values(np.array(y_test), y_pred)
```




```
True Positives : 17819
False Positives : 2890
True Negatives : 418
False Negatives : 723
```

In [786]:
```
# Printing roc auc score
y_pred = model.decision_function(X_te)
roc_auc_score(y_true=y_test, y_score=y_pred)
```

Out[786]: 0.6654208648439823

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

In [0]:
```
#%cd Assignments_DonorsChoose_2018
# stronging variables into pickle files python: http://www.jessicayung.com/how-to
# 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]:
```
# 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 i
  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 va
              tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split(
              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 [789]:
```
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, 1631.66it/s]
100%|██████████| 61178/61178 [01:47<00:00, 569.40it/s]
  1%|          | 165/21850 [00:00<00:13, 1639.50it/s]

61178
30

100%|██████████| 21850/21850 [00:13<00:00, 1618.38it/s]
100%|██████████| 21850/21850 [00:38<00:00, 560.73it/s]
  1%|          | 160/26220 [00:00<00:16, 1592.66it/s]

21850
30

100%|██████████| 26220/26220 [00:16<00:00, 1604.35it/s]
100%|██████████| 26220/26220 [00:45<00:00, 570.40it/s]

26220
30
```

In [790]:
```
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, 34276.99it/s]
100%|██████████| 61178/61178 [00:01<00:00, 43769.59it/s]
 16%|██        | 3503/21850 [00:00<00:00, 35022.85it/s]

61178
30

100%|██████████| 21850/21850 [00:00<00:00, 34027.19it/s]
100%|██████████| 21850/21850 [00:00<00:00, 44026.43it/s]
 13%|█         | 3354/26220 [00:00<00:00, 33536.11it/s]

21850
30

100%|██████████| 26220/26220 [00:00<00:00, 34105.81it/s]
100%|██████████| 26220/26220 [00:00<00:00, 43717.36it/s]

26220
30
```

In [791]:
```
tfidfw2v_vec_resource_tr = tfidf_w2v(textpreprocessed(X_train,'project_resource_s
tfidfw2v_vec_resource_te = tfidf_w2v(textpreprocessed(X_test,'project_resource_su
tfidfw2v_vec_resource_val = tfidf_w2v(textpreprocessed(X_val,'project_resource_su
```

```
100%|██████████| 61178/61178 [00:04<00:00, 15268.20it/s]
100%|██████████| 61178/61178 [00:04<00:00, 13700.99it/s]
  7%|▋         | 1475/21850 [00:00<00:01, 14746.04it/s]

61178
30

100%|██████████| 21850/21850 [00:01<00:00, 15053.49it/s]
100%|██████████| 21850/21850 [00:01<00:00, 13944.26it/s]
  6%|▌         | 1514/26220 [00:00<00:01, 15134.93it/s]

21850
30

100%|██████████| 26220/26220 [00:01<00:00, 15313.30it/s]
100%|██████████| 26220/26220 [00:01<00:00, 14136.70it/s]

26220
30
```

In [0]:
```
X_tr, X_cr, X_te = hstack_data(tfidfw2v_vec_titles_tr, tfidfw2v_vec_titles_val, t
                   tfidfw2v_vec_essay_tr, tfidfw2v_vec_essay_val, tfidfw2v_vec_es
                   tfidfw2v_vec_resource_tr, tfidfw2v_vec_resource_val, tfidfw2v_
```

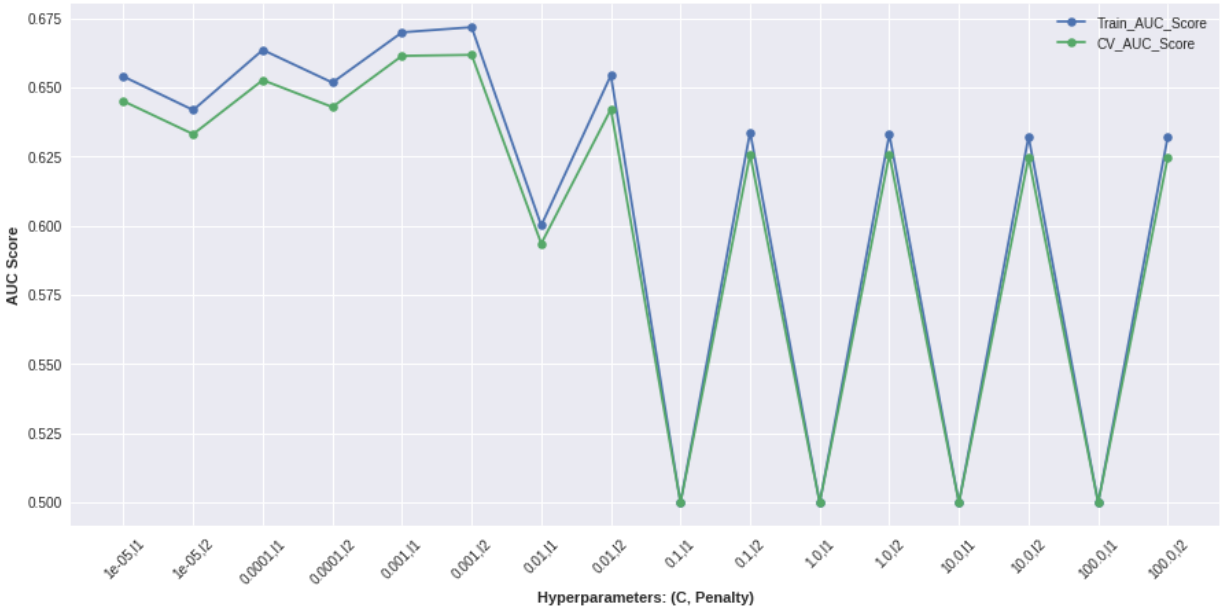
In [793]:
```
X_tr.shape,X_cr.shape,X_te.shape
```

Out[793]:
```
((61178, 172), (26220, 172), (21850, 172))
```

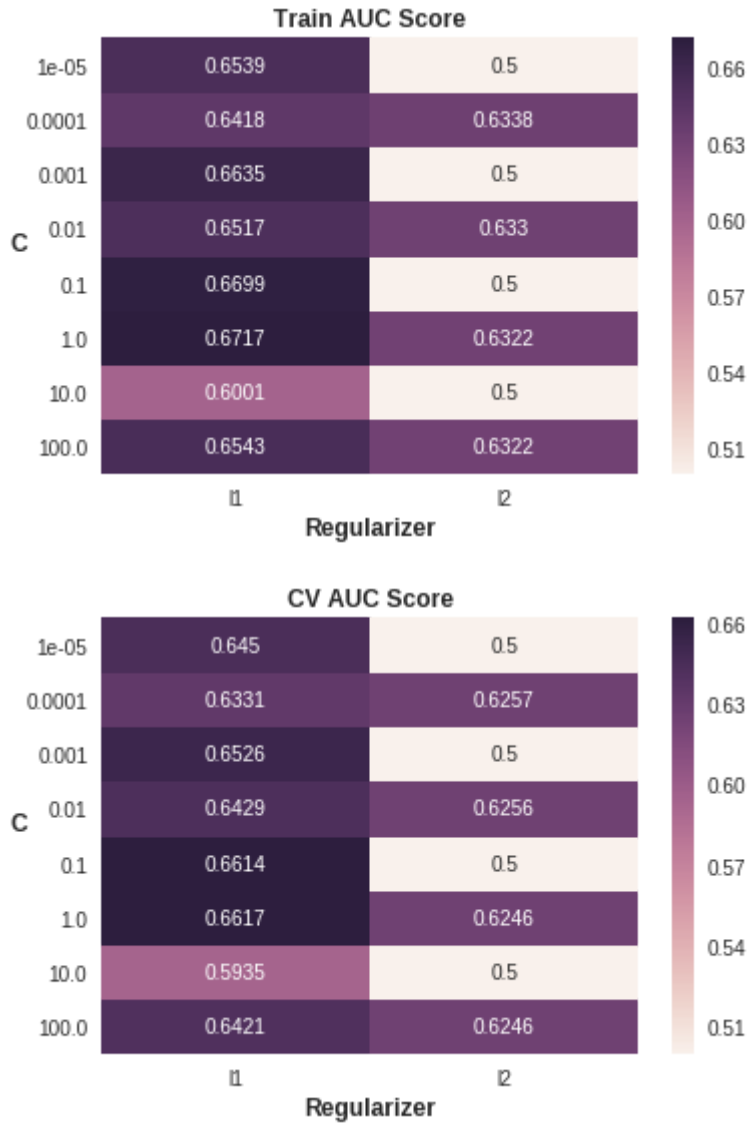
In [795]:
```
optimal_hyp(X_tr,y_train,X_cr,y_val)
```

```
execution time in minutes: 0.12984533707300822
execution time in hours: 0
```

In [796]:
```
plot_auc(df1)
```



In [797]:
```
plot_heatmap()
```



In [798]:
```
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)
```

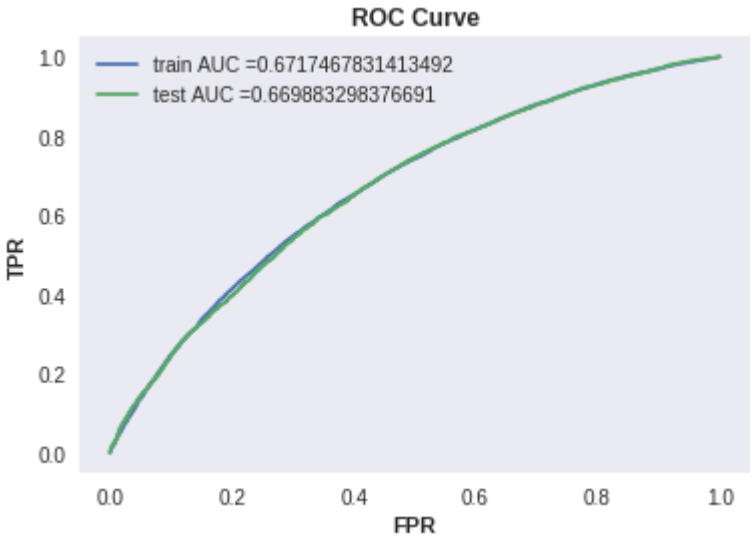
```
optimal c: 0.001
Regularizer: l2
CV AUC score: 0.661743729657827
```

In [799]:
```
model = SGDClassifier(class_weight='balanced', alpha=float(best_c), penalty=p, lo
model.fit(X_tr, y_train)
```

Out[799]:
```
SGDClassifier(alpha=0.001, average=False, class_weight='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=42, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False)
```

In [0]:
```
y_pred = model.predict(X_te)
```

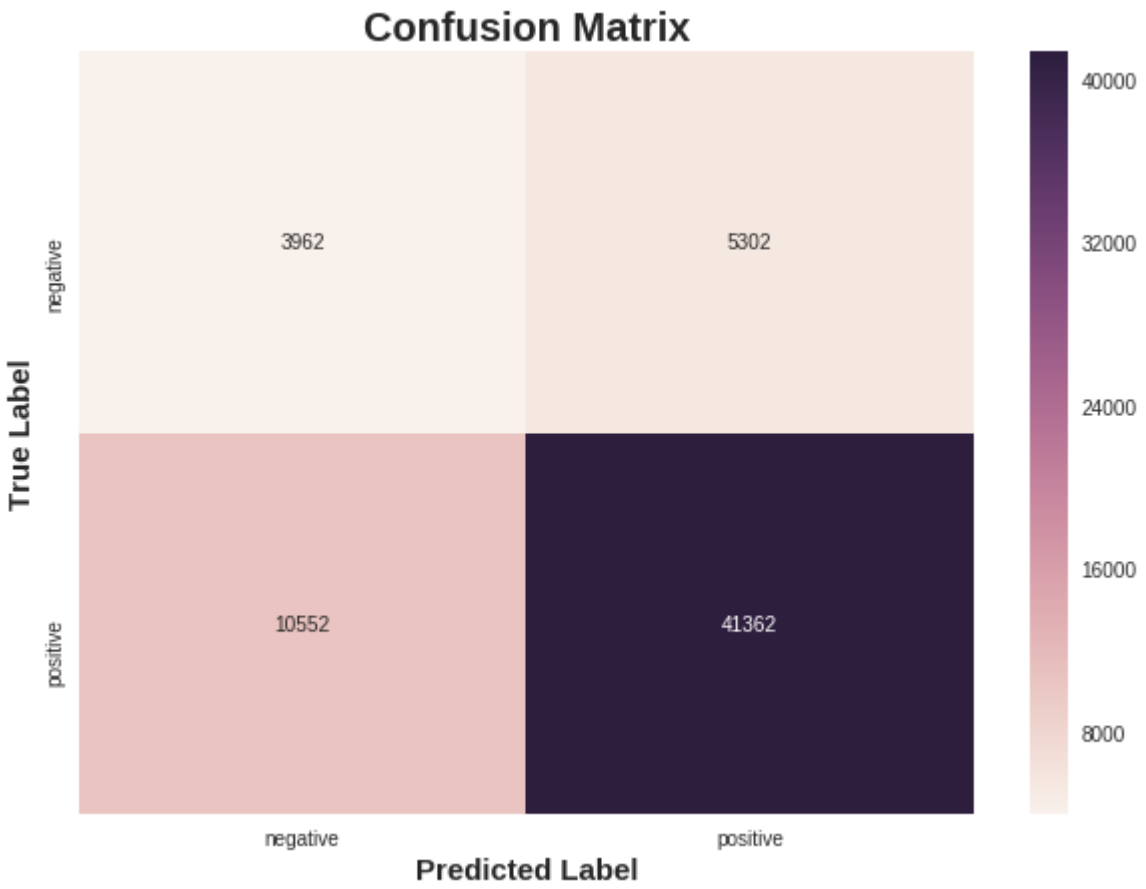
In [801]:
```
start = time.time()
error_plot(X_tr,X_te,y_train,y_test)
end = time.time()
minutes = float((end - start)/60)
print("execution time in minutes:",minutes)
print("execution time in hours:",int(minutes/60))
```




```
==========================================================================================
====================
execution time in minutes: 0.010188098748524983
execution time in hours: 0
```

confusion matrix for train data

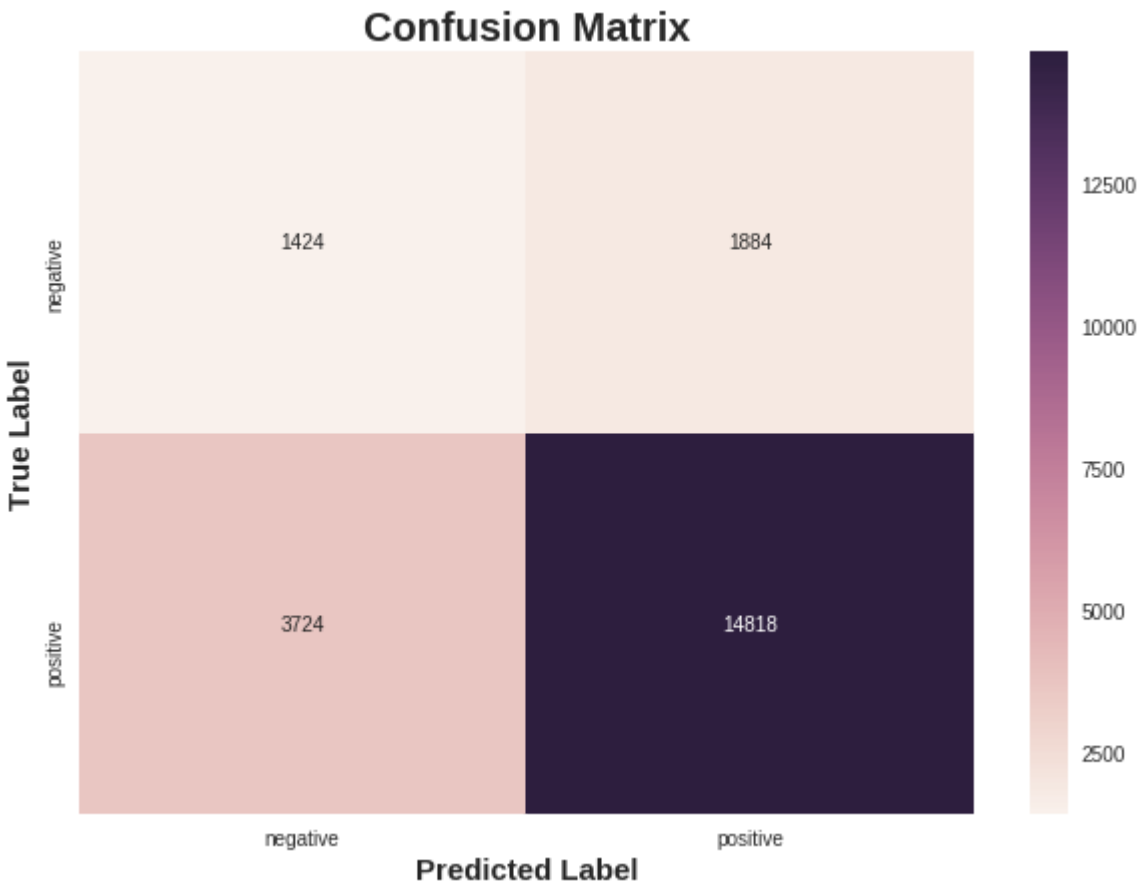
In [802]:
```
labels = ["negative","positive"]
get_confusion_matrix_values(np.array(y_train), y_train_pred1)
```




```
True Positives : 41362
False Positives : 5302
True Negatives : 3962
False Negatives : 10552
```

confusion matrix for test data

In [803]:
```
labels = ["negative","positive"]
get_confusion_matrix_values(np.array(y_test), y_pred)
```




```
True Positives : 14818
False Positives : 1884
True Negatives : 1424
False Negatives : 3724
```

In [805]:
```
# Printing roc auc score
y_pred = model.decision_function(X_te)
roc_auc_score(y_true=y_test, y_score=y_pred)
```

Out[805]: 0.669883298376691

[Task-2] Apply the Support Vector Machines on these features by finding the best hyper paramter

- school_state : categorical data
- clean_categories : categorical data
- clean_subcategories : categorical data
- project_grade_category :categorical data
- teacher_prefix : categorical data
- quantity : numerical data
- teacher_number_of_previously_posted_projects : numerical data
- price : numerical data
- sentiment score's of each of the essay : numerical data

- number of words in the title : numerical data
- number of words in the combine essays : numerical data
- Apply TruncatedSVD on TfidfVectorizer of essay text, choose the number of components (n_components) using elbow method : numerical data

In [0]:
```
"""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'], sentime
  p[['neg', 'pos', 'neu', 'compound']] = pd.DataFrame(sentiments)
  """
```

Out[38]:
```
"import nltk\nfrom nltk.sentiment.vader import SentimentIntensityAnalyzer\nnlt
k.download('vader_lexicon')\nsid = SentimentIntensityAnalyzer()\np = project_da
ta.copy()\nsentiments = []\nfor i in range(p.shape[0]):\n  line = p['essay'].il
oc[i]\n  sentiment = sid.polarity_scores(line)\n  sentiments.append([sentiment
['neg'], sentiment['pos'], sentiment['neu'], sentiment['compound']])\n  p[['ne
g', 'pos', 'neu', 'compound']] = pd.DataFrame(sentiments)\n  "
```

In [45]:
```
#project_data = p
%cd -
```

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

In [0]:
```
project_data= pd.read_pickle('set5')
```

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.c
```

In [0]:
```
#project_data = project_data[:70000]
```

In [0]:
```
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, str
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, stratify=y_tr
```

In [0]:
```
#X_train.shape, X_val.shape, X_test.shape
```

In [0]:
```
neg_standardized_tr,neg_standardized_val, neg_standardized_te = standardize_data(X
print()
pos_standardized_tr,pos_standardized_val, pos_standardized_te = standardize_data(X
print()
neu_standardized_tr,neu_standardized_val, neu_standardized_te = standardize_data(X
print()
comp_standardized_tr,comp_standardized_val, comp_standardized_te = standardize_da
qty_standardized_tr,qty_standardized_val, qty_standardized_te = standardize_data(X
price_standardized_tr,price_standardized_val, price_standardized_te = standardize_
print()
project_standardized_tr,project_standardized_val, project_standardized_te = stand
print()
title_standardized_tr, title_standardized_val, title_standardized_te = standardiz
print()
essay_standardized_tr,essay_standardized_val, essay_standardized_te = standardize_
print()
resource_standardized_tr,resource_standardized_val, resource_standardized_te = st
cat_one_hot_tr,cat_one_hot_val,cat_one_hot_te,_ = vectorized_data(X_train,X_val,X_
cat_sub_one_hot_tr,cat_sub_one_hot_val,cat_sub_one_hot_te,_ = vectorized_data(X_t
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_
teacher_one_hot_tr,teacher_one_hot_val,teacher_one_hot_te,_ = vectorized_data(X_t
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_
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')
```

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_t
               project_standardized_tr,title_standardized_tr, essay_standardized_
               neg_standardized_tr,pos_standardized_tr, neu_standardized_tr,comp_
f_val = hstack((one_hot_num_dig_val,grade_one_hot_val,state_one_hot_val,cat_one_h
               project_standardized_val,title_standardized_val, essay_standardize
               neg_standardized_val,pos_standardized_val, neu_standardized_val,co
f_te = hstack((one_hot_num_dig_te,grade_one_hot_te,state_one_hot_te,cat_one_hot_t
               project_standardized_te,title_standardized_te, essay_standardized_
               neg_standardized_te,pos_standardized_te, neu_standardized_te,comp_
```

In [0]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD

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 encodig ",text_tfidf_tr.shape)

  text_tfidf_val = vectorizer.transform(preprocessed_data_val)
  print("Shape of matrix after one hot encodig ",text_tfidf_val.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_val, text_tfidf_te
```

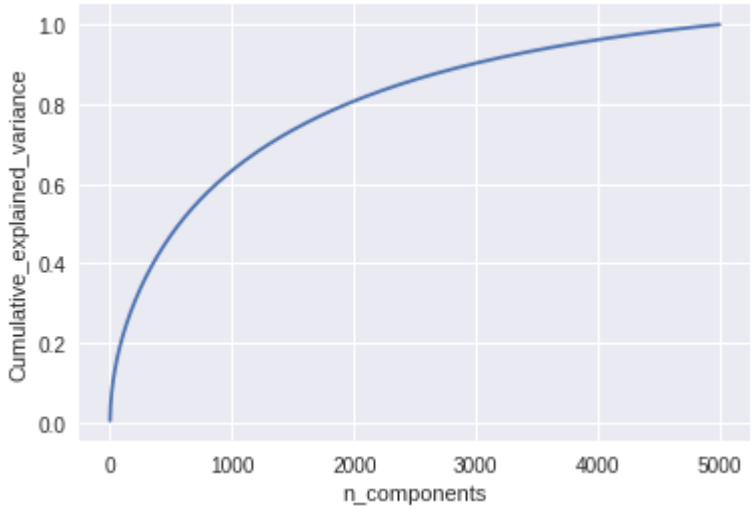
In [54]:
```
tfidf_vec_essay_tr,tfidf_vec_essay_val,tfidf_vec_essay_te = tfidf_vec(textpreproc
tfidf_vec_titles_tr,tfidf_vec_titles_val,tfidf_vec_titles_te = tfidf_vec(textprep
tfidf_vec_resource_tr,tfidf_vec_resource_val,tfidf_vec_resource_te = tfidf_vec(te
```

```
100%|██████████| 61178/61178 [00:37<00:00, 1610.82it/s]
100%|██████████| 26220/26220 [00:16<00:00, 1594.71it/s]
100%|██████████| 21850/21850 [00:13<00:00, 1599.75it/s]

Shape of matrix after one hot encodig  (61178, 13194)
Shape of matrix after one hot encodig  (26220, 13194)

  6%|▌         | 3479/61178 [00:00<00:01, 34783.48it/s]

Shape of matrix after one hot encodig  (21850, 13194)

100%|██████████| 61178/61178 [00:01<00:00, 35226.80it/s]
100%|██████████| 26220/26220 [00:16<00:00, 1608.36it/s]
100%|██████████| 21850/21850 [00:00<00:00, 34455.40it/s]

Shape of matrix after one hot encodig  (61178, 2265)

  0%|          | 0/61178 [00:00<?, ?it/s]

Shape of matrix after one hot encodig  (26220, 2265)
Shape of matrix after one hot encodig  (21850, 2265)

100%|██████████| 61178/61178 [00:03<00:00, 15334.73it/s]
100%|██████████| 26220/26220 [00:16<00:00, 1607.81it/s]
100%|██████████| 21850/21850 [00:01<00:00, 15262.35it/s]

Shape of matrix after one hot encodig  (61178, 4351)
Shape of matrix after one hot encodig  (26220, 4351)
Shape of matrix after one hot encodig  (21850, 4351)
```

In [0]:
```
%%time
# https://chrisalbon.com/machine_learning/feature_engineering/select_best_number_
X = tfidf_vec_essay_tr
# Create and run an TSVD with one less than number of features
tsvd = TruncatedSVD(n_components=4999)
X_tsvd = tsvd.fit(X) ##################
```

In [0]:
```
# List of explained variances
tsvd_var_ratios = tsvd.explained_variance_ratio_
```

In [85]:
```
percentage_var_explained = tsvd.explained_variance_ / np.sum(tsvd.explained_varia
cum_var_explained = np.cumsum(percentage_var_explained)

plt.clf()
plt.plot(cum_var_explained)
plt.xlabel('n_components')
plt.ylabel('Cumulative_explained_variance')
plt.show()
```



In [0]:
```
# Create a function
def select_n_components(var_ratio, goal_var: float) -> int:
    # Set initial variance explained so far
    total_variance = 0.0

    # Set initial number of features
    n_components = 0

    # For the explained variance of each feature:
    for explained_variance in var_ratio:

        # Add the explained variance to the total
        total_variance += explained_variance

        # Add one to the number of components
        n_components += 1

        # If we reach our goal level of explained variance
        if total_variance >= goal_var:
            # End the loop
            break

    # Return the number of components
    return n_components
```

In [0]:
```
# Run function
n=select_n_components(tsvd_var_ratios, 0.85)
```

In [0]:
```
# Run function
ncomp=select_n_components(tsvd_var_ratios, 0.95)
```

In [0]:
```
X = tfidf_vec_essay_tr
# Create and run an TSVD with one less than number of features
tsvd = TruncatedSVD(n_components=n)
X_tsvd = tsvd.fit(X)
```

In [0]:
```
tfidf_essay_tr = tsvd.transform(X)
tfidf_essay_val = tsvd.transform(tfidf_vec_essay_val)
tfidf_essay_te = tsvd.transform(tfidf_vec_essay_te)
```

In [0]:
```
from scipy.sparse import hstack

def hstack_data(f1_tr, f1_cr, f1_te):
  X_tr = hstack((f_tr, f1_tr)).tocsr()
  X_cr = hstack((f_val, f1_cr)).tocsr()
  X_te = hstack((f_te, f1_te)).tocsr()
  return X_tr,X_cr,X_te
```

In [0]:
```
X_tr, X_cr, X_te = hstack_data(tfidf_essay_tr, tfidf_essay_val, tfidf_essay_te)
```

In [59]:
```
xtrain = pd.DataFrame(X_tr.toarray())
xtrain['label'] = y_train
final = xtrain
final.sort_values(by='label', ascending=False, inplace=True)
n_zeros = final[final['label']==0]
n_ones = final[final['label']==1]
diff = n_ones.shape[0] - n_zeros.shape[1]
n_zeros.shape, n_ones.shape
```

Out[59]:
```
((9264, 691), (51914, 691))
```

In [0]:
```
n_ones = n_ones[:40000]
from sklearn.utils import resample
final_neg = n_zeros
final_neg_upsampled = resample(final_neg, replace=True, n_samples=diff, random_st
final = pd.concat([n_ones, final_neg_upsampled])
```

In [61]:
```
final.shape
```

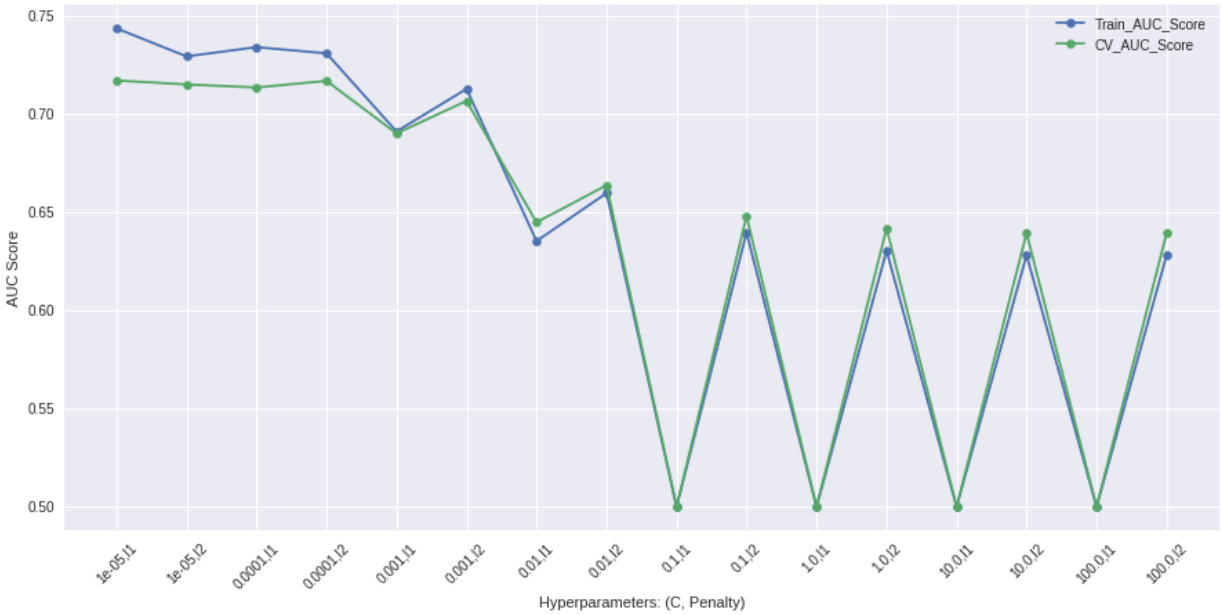
Out[61]:
```
(91223, 691)
```

In [0]:
```
import scipy
y_train = final['label']
X_tr = final.drop('label', axis=1)
X_tr = scipy.sparse.csr_matrix(X_tr.values)
```

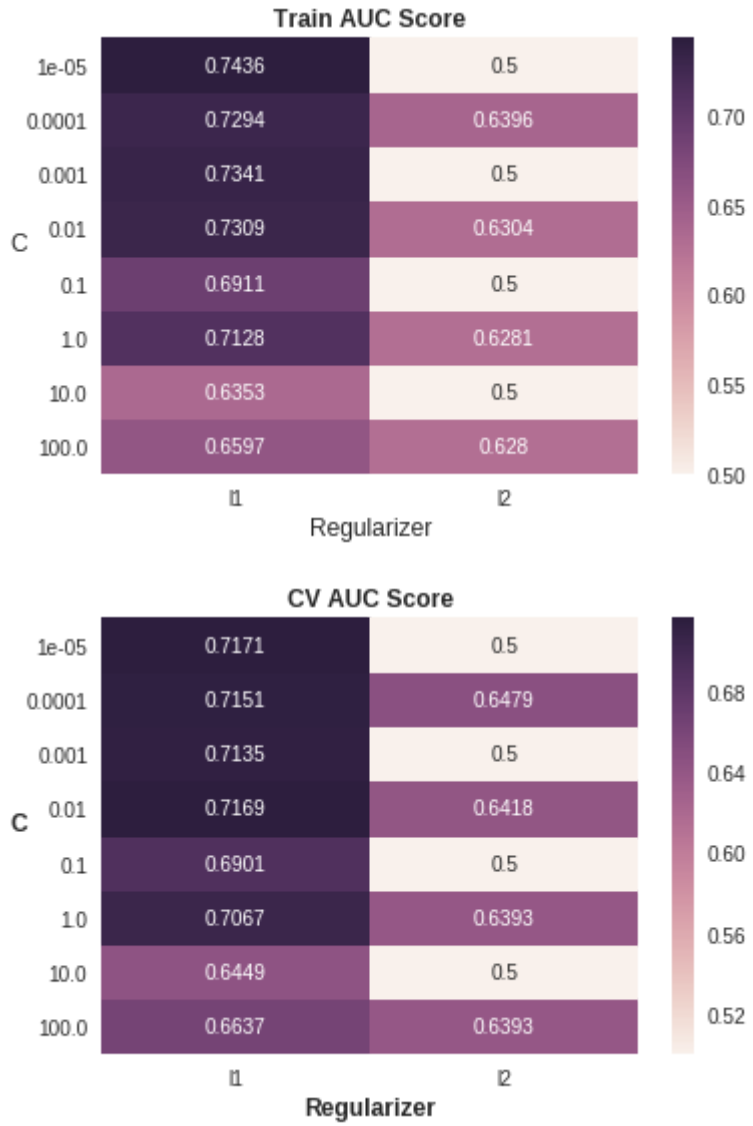
In [63]:
```
optimal_hyp(X_tr,y_train,X_cr,y_val)
```

```
execution time in minutes: 0.8826952576637268
execution time in hours: 0
```

In [64]:
```
plot_auc(df1)
```



In [65]:
```
plot_heatmap()
```



In [66]:
```
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)
```

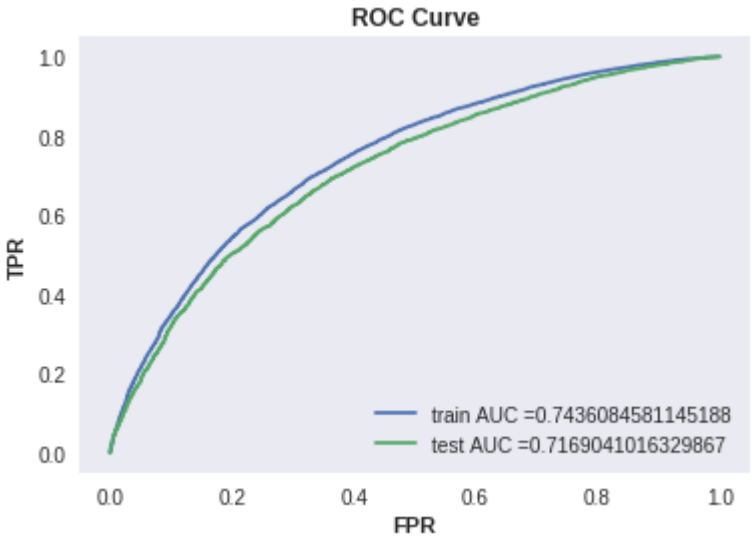
```
optimal c: 1e-05
Regularizer: l1
CV AUC score: 0.7171011122746441
```

In [67]:
```
model = SGDClassifier(class_weight='balanced', alpha=float(best_c), penalty=p, lo
model.fit(X_tr, y_train)
```

Out[67]:
```
SGDClassifier(alpha=1e-05, average=False, class_weight='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l1',
       power_t=0.5, random_state=42, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False)
```

In [0]:
```
y_pred = model.predict(X_te)
```

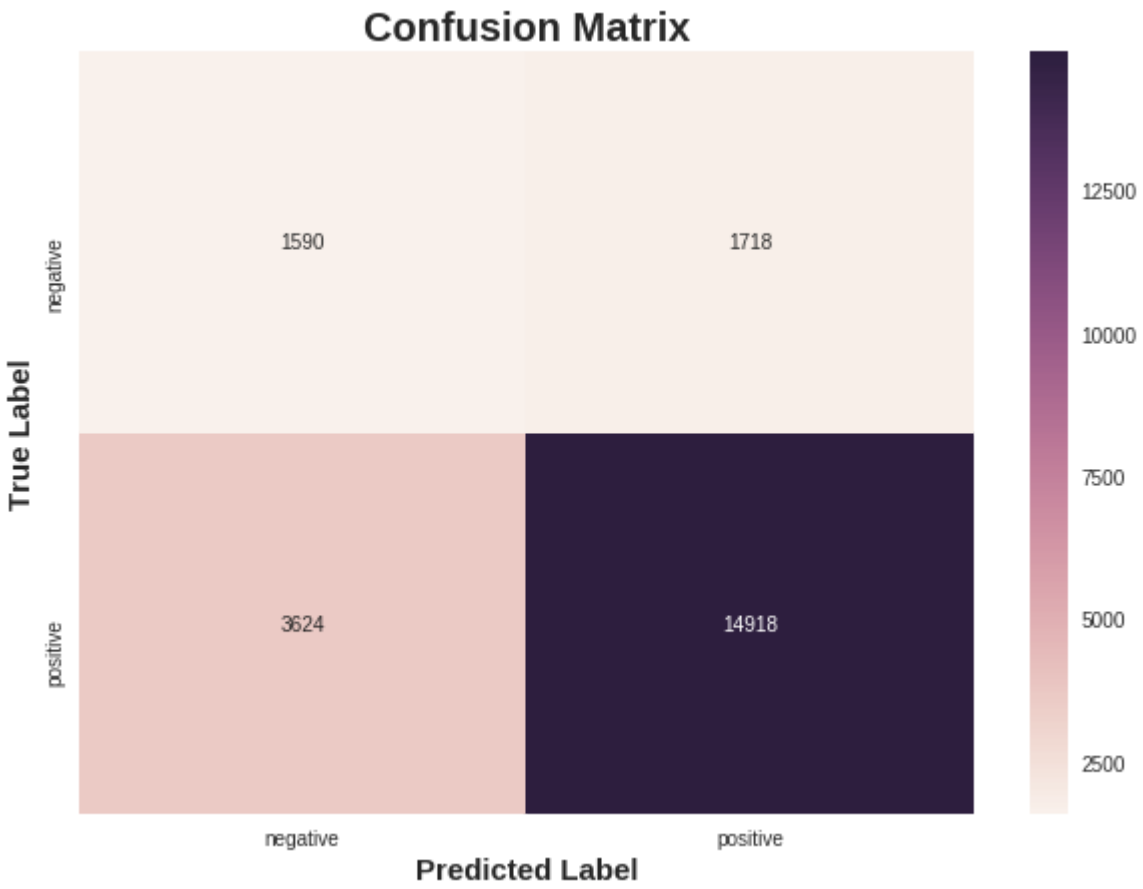
In [69]:
```
start = time.time()
error_plot(X_tr,X_te,y_train,y_test)
end = time.time()
minutes = float((end - start)/60)
print("execution time in minutes:",minutes)
print("execution time in hours:",int(minutes/60))
```




```
==========================================================================================
====================
execution time in minutes: 0.05978336334228516
execution time in hours: 0
```

confusion matrix for test data

In [72]:
```
labels = ["negative","positive"]
get_confusion_matrix_values(np.array(y_test), y_pred)
```




```
True Positives : 14918
False Positives : 1718
True Negatives : 1590
False Negatives : 3624
```

In [73]:
```
# Printing roc auc score
y_pred = model.decision_function(X_te)
roc_auc_score(y_true=y_test, y_score=y_pred)
```

Out[73]: 0.7169041016329867

Summary

In [86]:
```
from prettytable import PrettyTable
import sys
sys.stdout.write("\033[1;30m")

x = PrettyTable()
x.field_names = ["Vectorizer", "C","Penalty", "AUC"]

x.add_row(["BOW", 0.01, 'l2', 0.7289])
x.add_row(["TFIDF", 0.001, 'l2', 0.7234])
x.add_row(["AVGW2V", 0.0001, 'l1', 0.6654])
x.add_row(["TFIDFW2V", 0.001, 'l2', 0.6698])
x.add_row(["Task2", 0.00001, 'l1', 0.7169])

print(x)
```

```
+------------+--------+---------+--------+
| Vectorizer |   C    | Penalty |  AUC   |
+------------+--------+---------+--------+
|    BOW     |  0.01  |    l2   | 0.7289 |
|   TFIDF    | 0.001  |    l2   | 0.7234 |
|   AVGW2V   | 0.0001 |    l1   | 0.6654 |
|  TFIDFW2V  | 0.001  |    l2   | 0.6698 |
|   Task2    | 1e-05  |    l1   | 0.7169 |
+------------+--------+---------+--------+
```