In [0]:
```
from google.colab import drive
drive.mount('/content/drive')
```

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

In [0]:
```
%cd drive/My Drive
```

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

In [0]:
```
%cd Assignments_DonorsChoose_2018
```

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

In [0]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

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

In [0]:
```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

In [0]:
```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_data.columns)]

#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
project_data.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://stackoverflo
w.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-fr
om-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-strin
g-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 & Scienc
e", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on
space "Math & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to
replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(emp
ty) ex:"Math & Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the tra
iling 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://stackoverflo
w.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-fr
om-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-strin
g-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 & Scienc
e", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on
space "Math & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to
replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(emp
ty) ex:"Math & Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the tra
iling 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/2289859
5/4084039
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', 'it
self', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 't
hat', "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', 'becau
se', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',
'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on',
'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'a
ll', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'tha
n', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "shoul
d've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn',
"didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'm
a', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shoul
dn'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'].ap
ply(lambda x: find_num(x))
```

In [0]:
```
project_data['project_grade_category']=project_data['project_grade_category'].
str.replace(' ','_')
project_data['project_grade_category']=project_data['project_grade_category'].
str.replace('-','to')
set(project_data['project_grade_category'])
```

Out[0]:
```
{'Grades_3to5', 'Grades_6to8', 'Grades_9to12', 'Grades_PreKto2'}
```

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

In [0]:
```
#project_data = project_data[:5000]
```

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

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

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].valu
es.reshape(-1,1))
  print(standardized_data_train.shape)

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

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

  return standardized_data_train, standardized_data_traincv, standardized_data
_test
```

In [0]:
```
from sklearn.feature_extraction.text import CountVectorizer
def vectorized_data(df_train,df_cv,df_test,column_name,vocab=False):
  if(vocab):
    vectorizer = CountVectorizer(vocabulary=list(vocab.keys()), lowercase=Fals
e, binary=True)
  else:
    vectorizer = CountVectorizer(lowercase=False, binary=True)

  categories_one_hot_tr = vectorizer.fit_transform(df_train[column_name].value
s)
  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,vo
cab_list
```

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

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

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

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

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

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

In [0]:
```
price_standardized_tr, price_standardized_val, price_standardized_te = X_train
['price'].values.reshape(-1,1),X_val['price'].values.reshape(-1,1),X_test['pri
ce'].values.reshape(-1,1)
#standardize_data(X_train,X_val, X_test,'price')
print()
project_standardized_tr, project_standardized_val, project_standardized_te = X
_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1),\
X_val['teacher_number_of_previously_posted_projects'].values.reshape(-1,1),X_t
est['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)
#standardize_data(X_train,X_val, X_test,'teacher_number_of_previously_posted_p
rojects')
```

In [0]:
```
cat_one_hot_tr,cat_one_hot_val,cat_one_hot_te,cat_one_hot_list = vectorized_da
ta(X_train,X_val,X_test,'clean_categories',vocab=sorted_cat_dict)
cat_sub_one_hot_tr,cat_sub_one_hot_val,cat_sub_one_hot_te,cat_sub_one_hot_list
= vectorized_data(X_train,X_val,X_test,'clean_subcategories',vocab=sorted_cat_
dict)
```

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

In [0]:
```
school_state_dict = create_dict(X_train,'school_state')
teacher_prefix_dict = create_dict(X_train,'teacher_prefix')

state_one_hot_tr,state_one_hot_val,state_one_hot_te,state_one_hot_list = vecto
rized_data(X_train,X_val,X_test,'school_state',vocab=school_state_dict)
teacher_one_hot_tr,teacher_one_hot_val,teacher_one_hot_te,teacher_one_hot_list
= vectorized_data(X_train,X_val,X_test,'teacher_prefix',vocab=teacher_prefix_d
ict)
```

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

In [0]:
```
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 = vecto
rized_data(X_train,X_val,X_test,'project_grade_category',vocab=grade_dict)
```

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

In [0]:
```
one_hot_num_dig_tr = num_hot_encode(X_train,'numerical_digits')
one_hot_num_dig_te = num_hot_encode(X_test,'numerical_digits')
one_hot_num_dig_val = num_hot_encode(X_val,'numerical_digits')
```

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

[Task-1] Apply Decision Tree Classifier(DecisionTreeClassifier) 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_ho
t_tr,cat_sub_one_hot_tr,teacher_one_hot_tr,price_standardized_tr,project_stand
ardized_tr))
f_cr = hstack((one_hot_num_dig_val,grade_one_hot_val,state_one_hot_val,cat_one
_hot_val,cat_sub_one_hot_val,teacher_one_hot_val,price_standardized_val,projec
t_standardized_val))
f_te = hstack((one_hot_num_dig_te,grade_one_hot_te,state_one_hot_te,cat_one_ho
t_te,cat_sub_one_hot_te,teacher_one_hot_te,price_standardized_te,project_stand
ardized_te))

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

In [0]:
```
feature_list_x = ['dig_0','dig_1']+grade_one_hot_list+state_one_hot_list+cat_o
ne_hot_list+cat_sub_one_hot_list+teacher_one_hot_list+['price','teacher_number
_of_previously_posted_projects']
len(feature_list_x)
```

Out[0]: 82

In [0]:
```
from sklearn.metrics import roc_auc_score
from sklearn.tree import DecisionTreeClassifier
import time

def optimal_hyp(X_tr,y_train,X_cr,y_val):
  global df1
  depth =  [1, 5, 10, 50, 100, 500, 100]
  min_samples_split_list = [5, 10, 100, 500]
  cols = ['depth', 'min_samples_split', 'Train_AUC_Score', 'CV_AUC_Score']
  lst = []
  start = time.time()
  for d in depth:
    for l in min_samples_split_list:
        clf =  DecisionTreeClassifier(max_depth = d, min_samples_split = l,ran
dom_state = 0)
        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_val), y_score=clf.predict_p
roba(X_cr)[:,1])
        #print(tr_score)
        lst.append([d,l,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-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)
    #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://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix
import seaborn as sns
def error_plot(X_train,X_te,y_train,y_test):
  global model
  model =  DecisionTreeClassifier(max_depth = int(best_d), min_samples_split = int(sample_split), random_state = 0)
  model.fit(X_train, y_train)
  #pred = model.predict(X_train)
  y_train_pred = model.predict_proba(X_train)[:,1]

  #pred = model.predict(X_te)
  y_test_pred = model.predict_proba(X_te)[:,1]

  #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("hyperparameters(depth:{},min sample split:{})")
  plt.ylabel("AUC")
  plt.title("ERROR PLOTS")
  plt.grid()
  plt.show()
  print("="*100)
```

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 hi
gh

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshol
d", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions'''
```

Out[0]:
```
'# we are writing our own function for predict, with defined thresould\n# we
will pick a threshold that will give the least fpr\ndef predict(proba, thresh
ould, fpr, tpr):\n    \n    t = threshould[np.argmax(fpr*(1-tpr))]\n    \n
# (tpr*(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
threshold", np.round(t,3))\n    predictions = []\n    for i in proba:\n
if i>=t:\n            predictions.append(1)\n        else:\n            predi
ctions.append(0)\n    return predictions'
```

In [0]:
```
def plot_auc(df1):
  df = df1[['Train_AUC_Score','CV_AUC_Score']]
  df1['depth'] = df1['depth'].astype(str)
  df1['min_samples_split'] = df1['min_samples_split'].astype(str)

  ax = df.plot(xticks=df.index, rot=55,figsize=(15,5))
  ax.set_ylabel("AUC Score",fontsize=20)
  ax.set_xlabel("Hyperparameters: (Depth,Min samples split)",fontsize=20)

  ax.set_xticklabels('('+df1['depth'] +',' + df1['min_samples_split']+')',font
size=15)
  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('Depth',weight='bold')
  ax.set_ylabel('Min sample split',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()
```

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

1.5.1 Vectorizing Categorical data

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

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

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

In [0]:
```
tfidf_vec_essay_tr,tfidf_vec_essay_val,tfidf_vec_essay_te,tfidf_vec_essay_list
= tfidf_vec(textpreprocessed(X_train,'essay'),textpreprocessed(X_val,'essay'),
textpreprocessed(X_test,'essay'))
tfidf_vec_titles_tr,tfidf_vec_titles_val,tfidf_vec_titles_te,tfidf_vec_titles_
list = tfidf_vec(textpreprocessed(X_train,'project_title'),textpreprocessed(X_
val,'project_title'), textpreprocessed(X_test,'project_title'))
tfidf_vec_resource_tr,tfidf_vec_resource_val,tfidf_vec_resource_te,tfidf_vec_r
esource_list = tfidf_vec(textpreprocessed(X_train,'project_resource_summary'),
textpreprocessed(X_val,'project_resource_summary'), textpreprocessed(X_test,'p
roject_resource_summary'))
```

```
100%|██████████| 69918/69918 [00:42<00:00, 1628.88it/s]
100%|██████████| 17480/17480 [00:10<00:00, 1591.61it/s]
100%|██████████| 21850/21850 [00:13<00:00, 1618.69it/s]

Shape of matrix after one hot encodig  (69918, 13872)
Shape of matrix after one hot encodig  (17480, 13872)

  5%|▌         | 3547/69918 [00:00<00:01, 35467.16it/s]

Shape of matrix after one hot encodig  (21850, 13872)

100%|██████████| 69918/69918 [00:02<00:00, 34514.67it/s]
100%|██████████| 17480/17480 [00:00<00:00, 34245.07it/s]
100%|██████████| 21850/21850 [00:00<00:00, 34209.54it/s]

Shape of matrix after one hot encodig  (69918, 2460)
Shape of matrix after one hot encodig  (17480, 2460)

  2%|▏          | 1543/69918 [00:00<00:04, 15425.64it/s]

Shape of matrix after one hot encodig  (21850, 2460)

100%|██████████| 69918/69918 [00:04<00:00, 15587.33it/s]
100%|██████████| 17480/17480 [00:01<00:00, 15410.87it/s]
100%|██████████| 21850/21850 [00:01<00:00, 15337.46it/s]

Shape of matrix after one hot encodig  (69918, 4634)
Shape of matrix after one hot encodig  (17480, 4634)
Shape of matrix after one hot encodig  (21850, 4634)
```

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)
print(tfidf_vec_resource_tr.shape)
```

```
(69918, 2)
(69918, 4)
(69918, 51)
(69918, 9)
(69918, 9)
(69918, 5)
(69918, 1)
(69918, 1)
(69918, 2460)
(69918, 13872)
(69918, 4634)
```

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

```
21048
```

In [0]:
```
start_essay = len([*feature_list , *tfidf_vec_titles_list])
end_essay = len([*feature_list , *tfidf_vec_titles_list, *tfidf_vec_essay_list
])-1
```

In [0]:
```
X_tr, X_cr, X_te = hstack_data(tfidf_vec_titles_tr, tfidf_vec_titles_val, tfid
f_vec_titles_te, \
                    tfidf_vec_essay_tr, tfidf_vec_essay_val, tfidf_vec_essay_te
,\
                    tfidf_vec_resource_tr, tfidf_vec_resource_val, tfidf_vec_re
source_te)
```

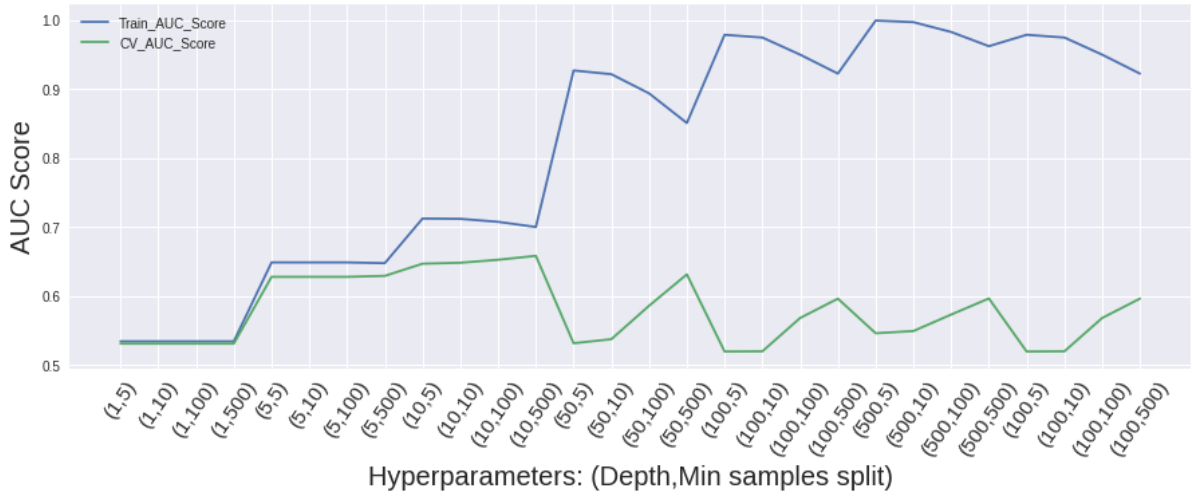
In [0]:
```
X_tr.shape,X_cr.shape,X_te.shape
```

Out[0]:
```
((69918, 21048), (17480, 21048), (21850, 21048))
```

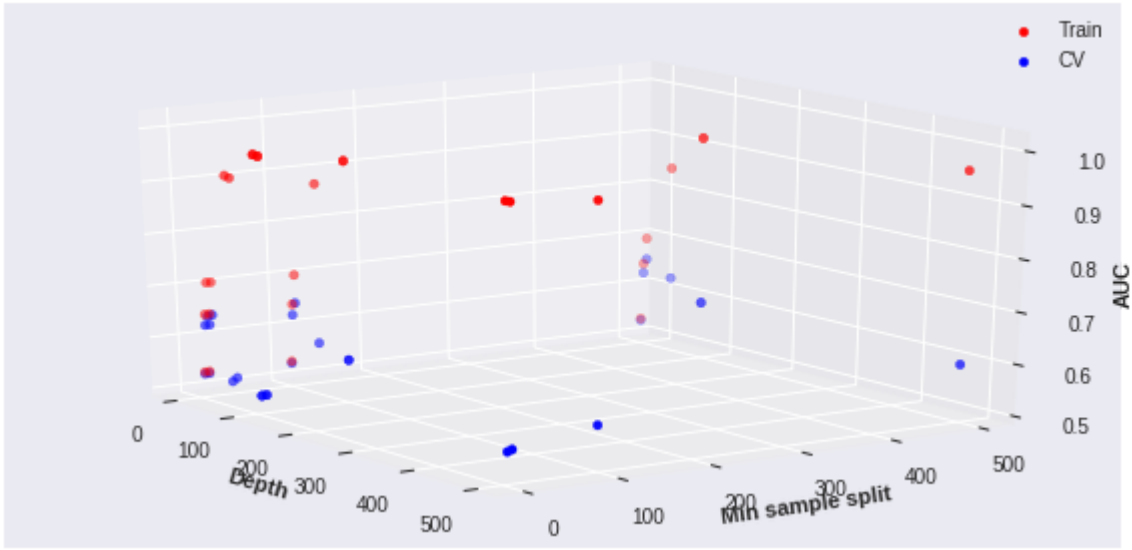
In [0]:
```
optimal_hyp(X_tr,y_train,X_cr,y_val)
```

```
execution time in minutes: 60.505533579985304
execution time in hours: 1
```

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



In [0]:
```
x = df1['depth'].astype(float).values
y = df1['min_samples_split'].astype(float).values
z1 = df1['Train_AUC_Score'].astype(float).values
z2 = df1['CV_AUC_Score'].astype(float).values
plot_3D(x,y,z1,z2)
```



In [0]:
```
best_d,sample_split,tr_auc_score,cv_auc_score = df1.iloc[df1.CV_AUC_Score.argmax()]
print("optimal depth:",best_d,"\nMin sample split:",sample_split,"\nCV AUC score:", cv_auc_score)
```

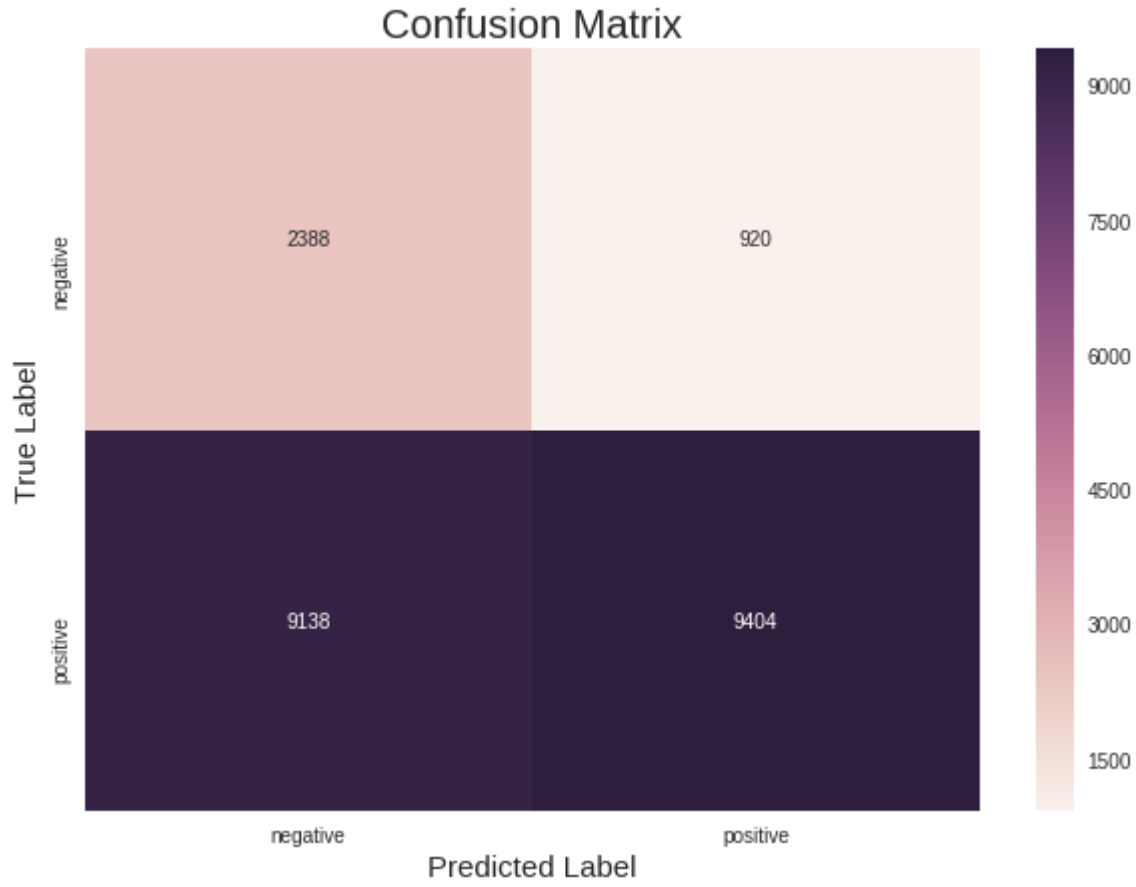
```
optimal depth: 10
Min sample split: 500
CV AUC score: 0.6584277885786018
```

In [0]:
```
model = DecisionTreeClassifier(max_depth = int(best_d), min_samples_split = in
t(sample_split), class_weight = 'balanced',random_state = 0)
model.fit(X_tr, y_train)
```

Out[0]:
```
DecisionTreeClassifier(class_weight='balanced', criterion='gini',
            max_depth=10, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=500,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')
```

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

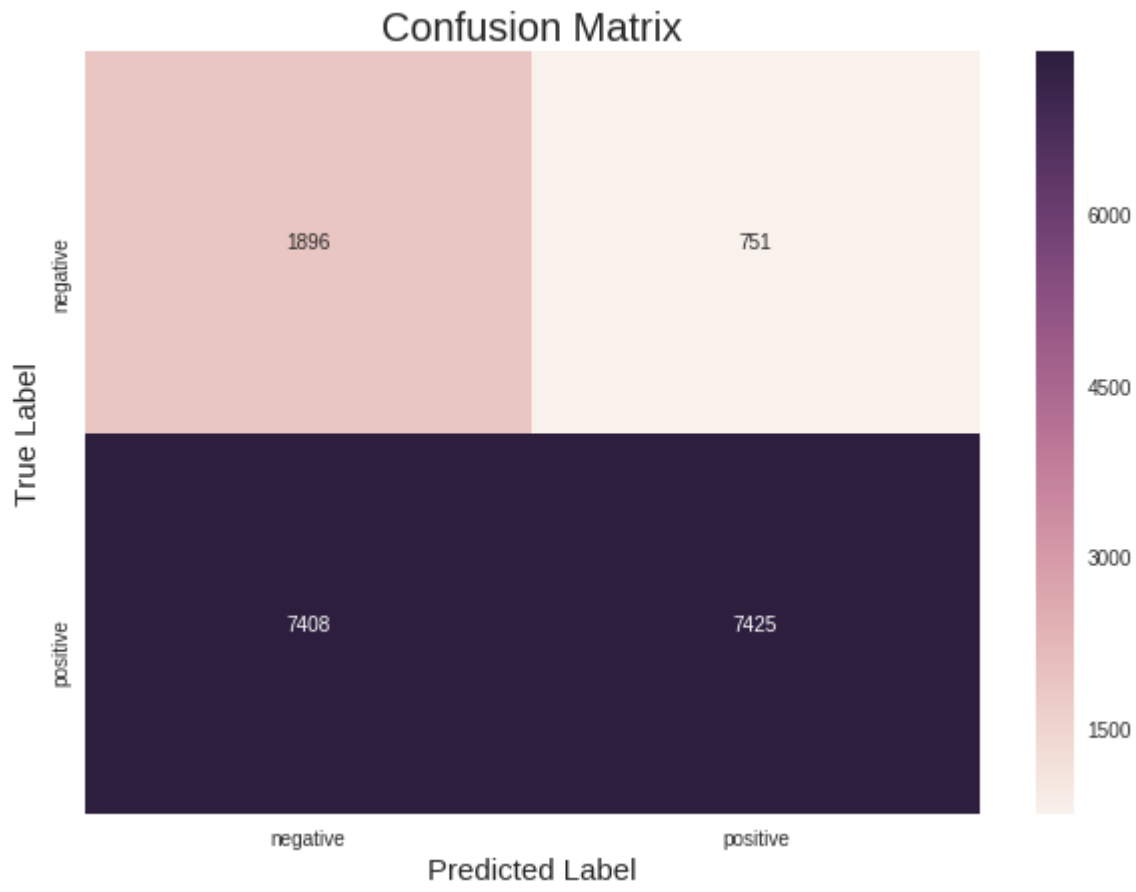
In [0]:
```
labels = ["negative","positive"]
get_confusion_matrix_values(np.array(y_test), y_pred_te)
```




```
True Positives : 9404
False Positives : 920
True Negatives : 2388
False Negatives : 9138
```

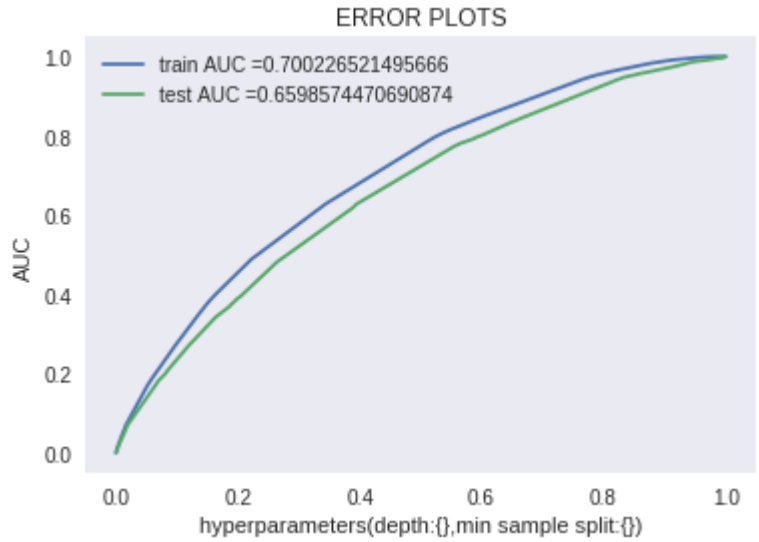
In [0]:
```
y_pred_val = model.predict(X_cr)
```

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




```
True Positives : 7425
False Positives : 751
True Negatives : 1896
False Negatives : 7408
```

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: 0.3238296945889791
execution time in hours: 0
```

In [0]:
```
# Printing roc auc score
y_pred = model.predict_proba(X_te)[:,1]
roc_auc_score(y_true=y_test, y_score=y_pred)
```

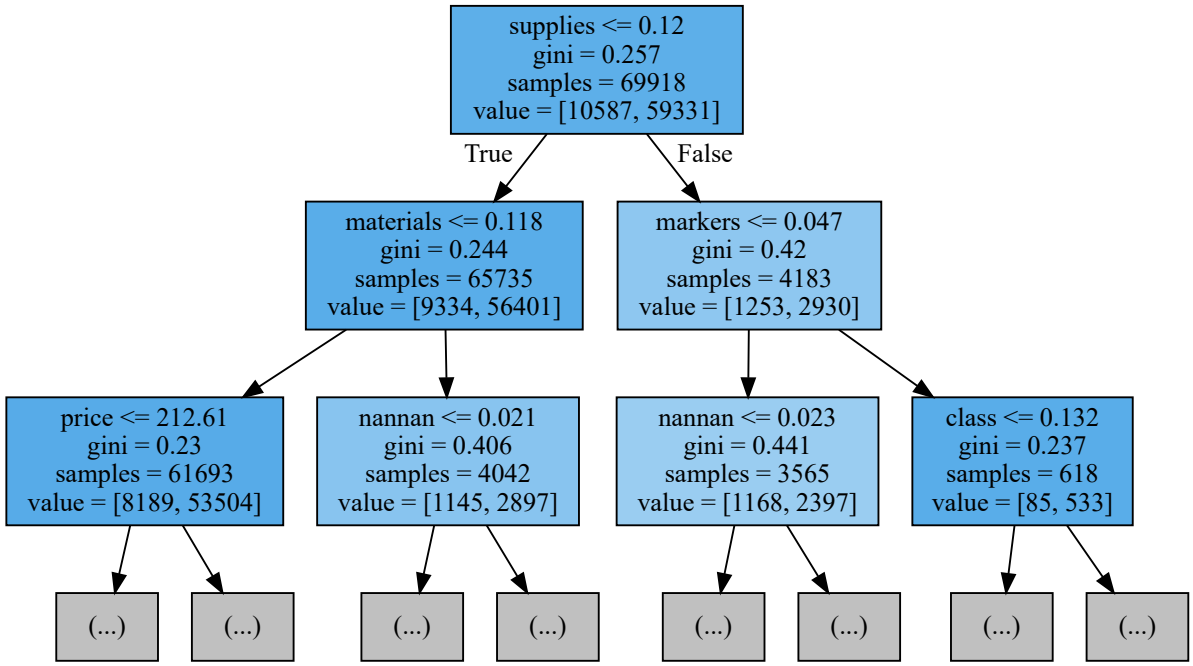
Out[0]: 0.6598574470690874

In [0]:
```
%cd -
#%cd drive/My Drive
```

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

In [0]:
```
from sklearn.tree import export_graphviz
import graphviz

export_graphviz(model, out_file="dt_tfidf.dot", feature_names = feature_names,
max_depth = 2, filled = True)
with open("dt_tfidf.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```

Out[0]:



In [0]:
```
ind_bool_fp = [i==0 and j==1 for i, j in zip(y_val, y_pred_val)]
ind_fp =list(np.where(ind_bool_fp)[0])
ind_bool_fn = [i==1 and j==0 for i, j in zip(y_val, y_pred_val)]
ind_fn =list(np.where(ind_bool_fn)[0])
ind_bool_tp = [i==1 and j==1 for i, j in zip(y_val, y_pred_val)]
ind_tp =list(np.where(ind_bool_tp)[0])
ind_bool_tn = [i==0 and j==0 for i, j in zip(y_val, y_pred_val)]
ind_tn =list(np.where(ind_bool_tn)[0])
```

In [0]:
```
print("False Positive:",len(ind_fp))
print("True Positive:",len(ind_tp))
print("False Negative:",len(ind_fn))
print("True Negative:",len(ind_tn))
```

```
False Positive: 751
True Positive: 7425
False Negative: 7408
True Negative: 1896
```

In [0]:
```
df = X_cr.toarray()
```

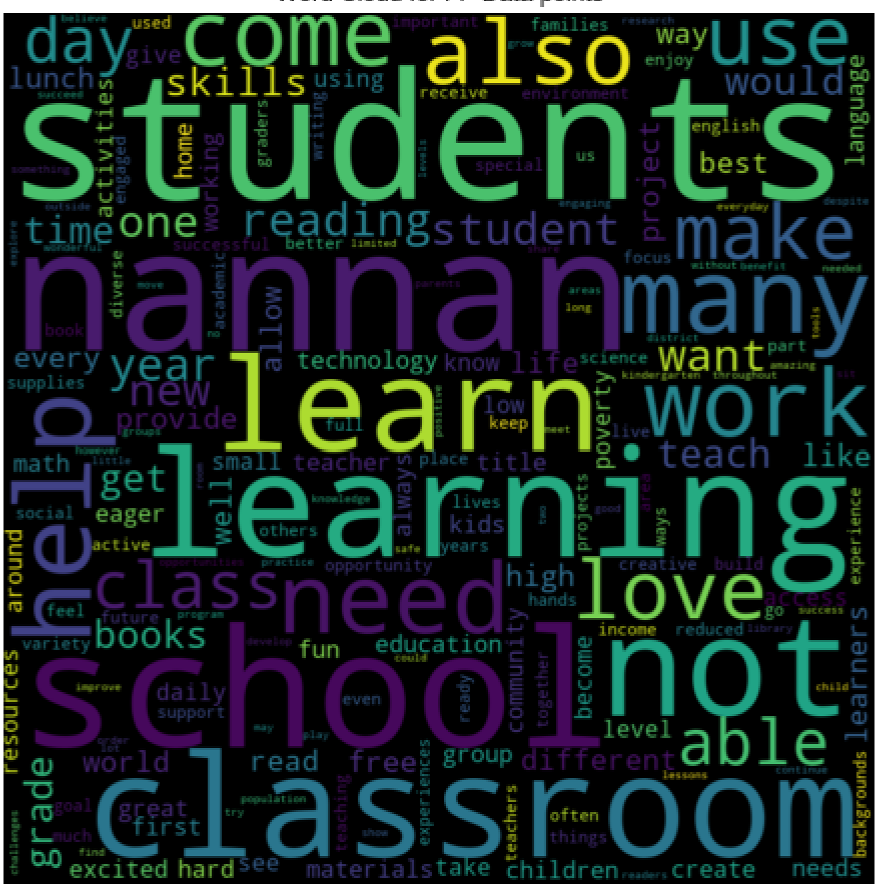
In [0]:
```
df_fp = df[ind_fp,:]
df_fn = df[ind_fn,:]
df_tp = df[ind_tp,:]
df_tn = df[ind_tn,:]
```

In [0]:
```
df_fp_words = df_fp[:,start_essay:end_essay+1]
df_fp_words =pd.DataFrame(df_fp_words)
df_fp_words.columns = tfidf_vec_essay_list
x=df_fp_words[df_fp_words > 0].count()
```

Plotting the WordCloud

In [0]:
```
import matplotlib.pyplot as plt
from wordcloud import WordCloud

wordcloud = WordCloud(width = 800, height = 800,)
wordcloud.generate_from_frequencies(frequencies=x)
plt.figure(figsize = (8, 8), facecolor = None)
plt.title("Word Cloud for FP Data points")
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

Word Cloud for FP Data points



These are the top 100 words which corresponds to FP datapoints

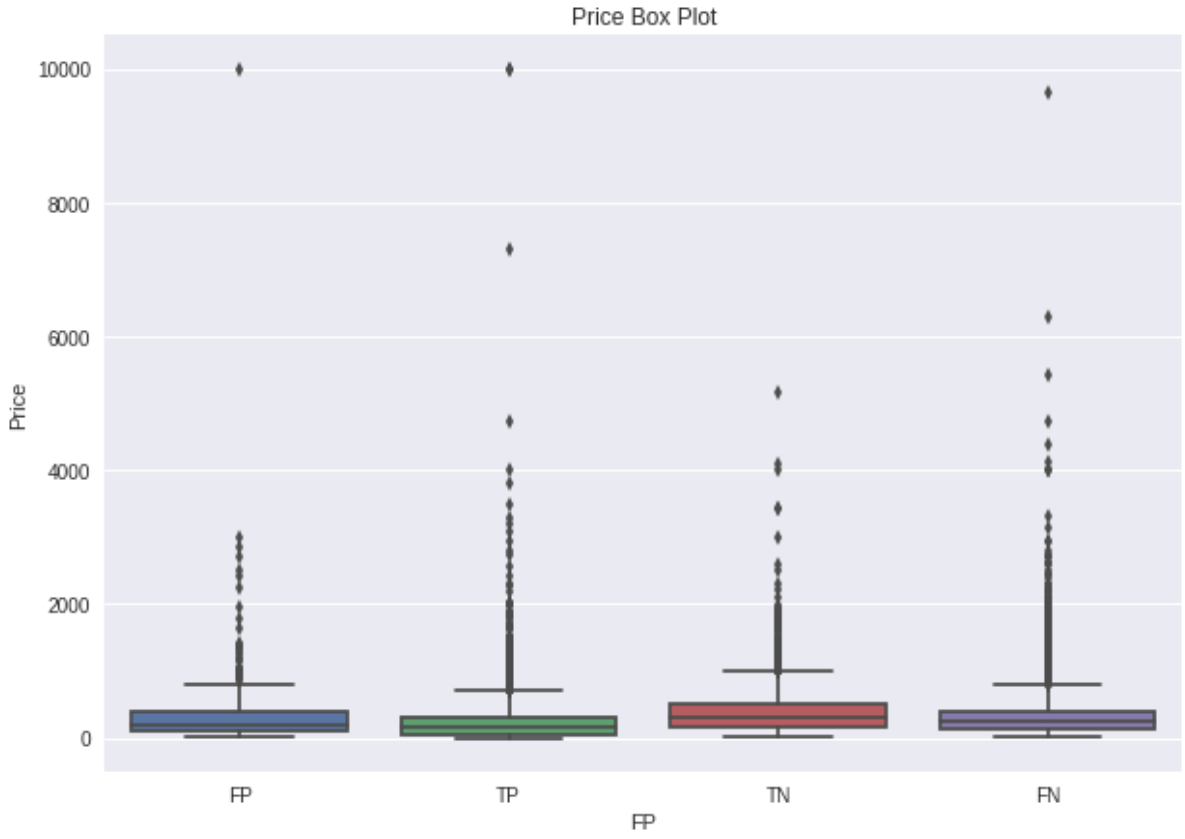
In [0]:
```
ind_price = feature_names.index('price')
ind_prev_proj = feature_names.index('teacher_number_of_previously_posted_proje
cts')
```

In [0]:
```
price_fp = df_fp[:,ind_price]
prev_proj_fp = df_fp[:,ind_prev_proj]

price_fn = df_fn[:,ind_price]
prev_proj_fn = df_fn[:,ind_prev_proj]

price_tp = df_tp[:,ind_price]
prev_proj_tp = df_tp[:,ind_prev_proj]

price_tn = df_tn[:,ind_price]
prev_proj_tn = df_tn[:,ind_prev_proj]
```

Plotting Box Plot of Price of false positive data points

In [0]:
```
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

# Make boxplot for one group only
plt.figure(figsize=(10,7))
ax = sns.boxplot(data=[price_fp,price_tp,price_tn,price_fn])
ax.set_xticklabels(['FP','TP','TN','FN'])
ax.set_title("Price Box Plot")
ax.set_ylabel('Price')
ax.set_xlabel('FP')
#sns.plt.show()
```

Out[0]: Text(0.5, 0, 'FP')



Since we are analysing FP which are misclassified negative datapoints we will look at TN and TP box plots.
Observations:

By Comparing FP and TN we will see what price values would have given us correct answer.

- We can see if the price was more than 386 It would have given us correct class as 75 th precentile of FP is 386 and 75th percentile of TN is 484 and also the median of TN is close to the 75th percentile.
- 25 th percentile of the FP is 88 and 25th percentile of TN is 164 .If the price is below 88 chances of misclassification is more.

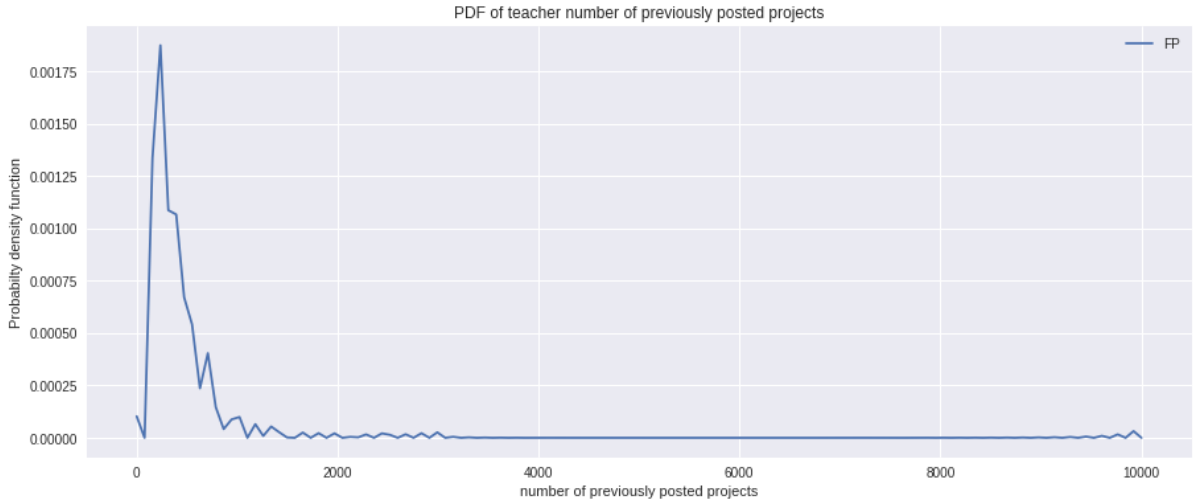
In [0]:
```
p25 = [np.percentile(label, 25) for label in [price_fp, price_tp, price_tn, pr
ice_fn]] # return 25th percentile.
p50 = [np.percentile(label, 50) for label in [price_fp, price_tp, price_tn, pr
ice_fn]] # return 50th percentile.
p75 = [np.percentile(label, 75) for label in [price_fp, price_tp, price_tn, pr
ice_fn]] # return 75th percentile.
per = pd.DataFrame([p25,p50,p75],columns=['FP','TP','TN','FN'],index=['25th Pe
rcentile','50th Percentile','75th Percentile'])
per
```

Out[0]:

	FP	TP	TN	FN
25th Percentile	88.04	57.04	164.135	137.8950
50th Percentile	199.73	164.44	297.160	244.8650
75th Percentile	386.81	315.37	494.895	404.9075

Plotting the pdf with the teacher number of previously posted projects of false positive data points

In [0]:
```
plt.figure(figsize=(15,6))
ax = sns.kdeplot(price_fp, label="FP", bw=0.5)
#ax = sns.kdeplot(price_tp, label="TP", bw=0.5)
#ax = sns.kdeplot(price_fn, label="FN", bw=0.5)
#ax = sns.kdeplot(price_tn, label="TN", bw=0.5)

ax.set_title("PDF of teacher number of previously posted projects")
ax.set_ylabel('Probabilty density function')
ax.set_xlabel('number of previously posted projects')
plt.legend()
plt.show()
```



Observation

For FP datapoint around 150-500 no. of previously posted projects is found to be more.

Task 2

In [0]:
```
pd_df = dict(zip(feature_names, model.feature_importances_))
feature_df=pd.DataFrame(list(pd_df.items()), columns=['features', 'coef'])
```

In [0]:
```
feature_df = feature_df.sort_values("coef", ascending = False)[:5000]
indices = list(feature_df.index.values)
```

In [0]:
```
x_tr = X_tr[:,indices]
x_te = X_te[:,indices]
x_cr= X_cr[:,indices]
```

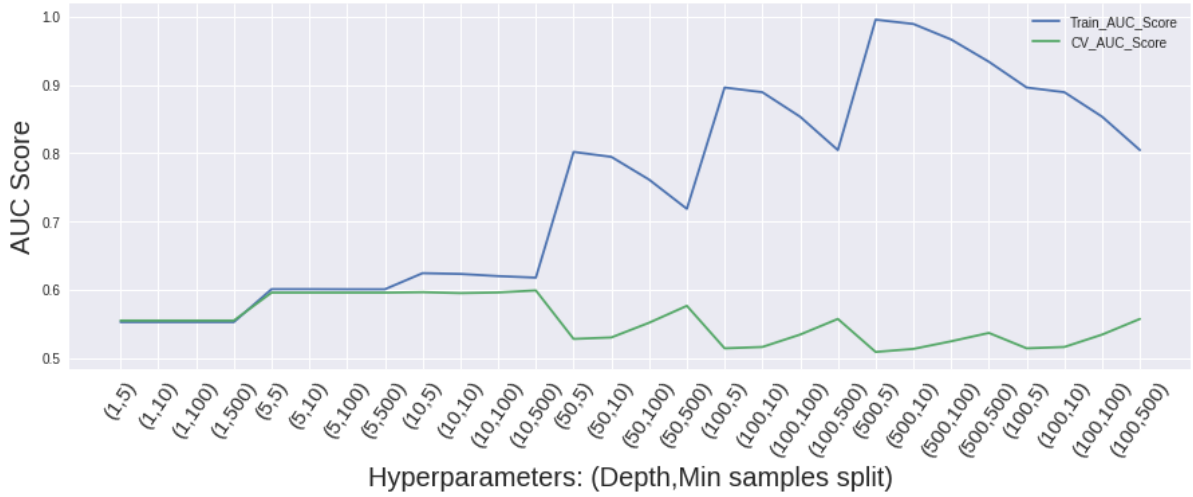
In [0]:
```
x_tr.shape, x_cr.shape, x_te.shape
```

Out[0]:
```
((69918, 5000), (17480, 5000), (21850, 5000))
```

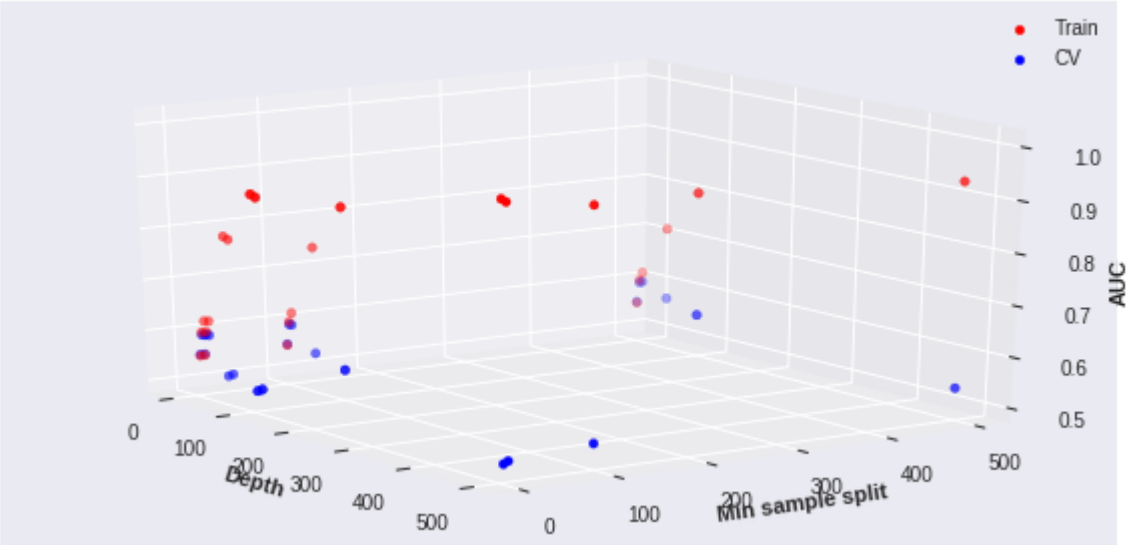
In [0]:
```
optimal_hyp(x_tr,y_train,x_cr,y_val)
```

```
execution time in minutes: 21.813022514184315
execution time in hours: 0
```

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



In [0]:
```
x = df1['depth'].astype(float).values
y = df1['min_samples_split'].astype(float).values
z1 = df1['Train_AUC_Score'].astype(float).values
z2 = df1['CV_AUC_Score'].astype(float).values
plot_3D(x,y,z1,z2)
```



In [0]:
```
best_d,sample_split,tr_auc_score,cv_auc_score = df1.iloc[df1.CV_AUC_Score.argmax()]
print("optimal depth:",best_d,"\nMin sample split:",sample_split,"\nCV AUC score:", cv_auc_score)
```

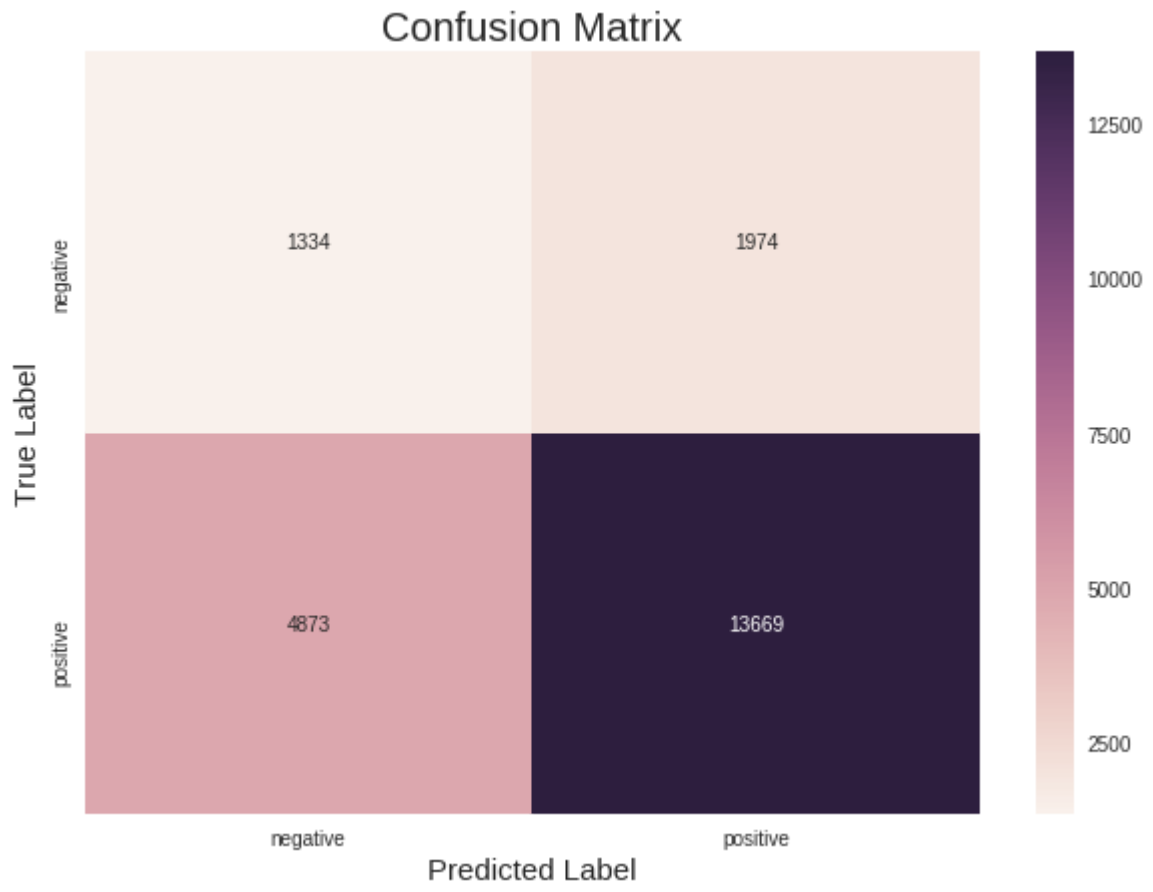
```
optimal depth: 10
Min sample split: 500
CV AUC score: 0.598847969934812
```

In [0]:
```
model = DecisionTreeClassifier(max_depth = int(best_d), min_samples_split = int(sample_split), class_weight = 'balanced',random_state = 0)
model.fit(x_tr, y_train)
```

Out[0]:
```
DecisionTreeClassifier(class_weight='balanced', criterion='gini',
            max_depth=10, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=500,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')
```

In [0]:
```
y_pred_te = model.predict(x_te)
```

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




```
True Positives : 13669
False Positives : 1974
True Negatives : 1334
False Negatives : 4873
```

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: 0.09674640496571858
execution time in hours: 0
```

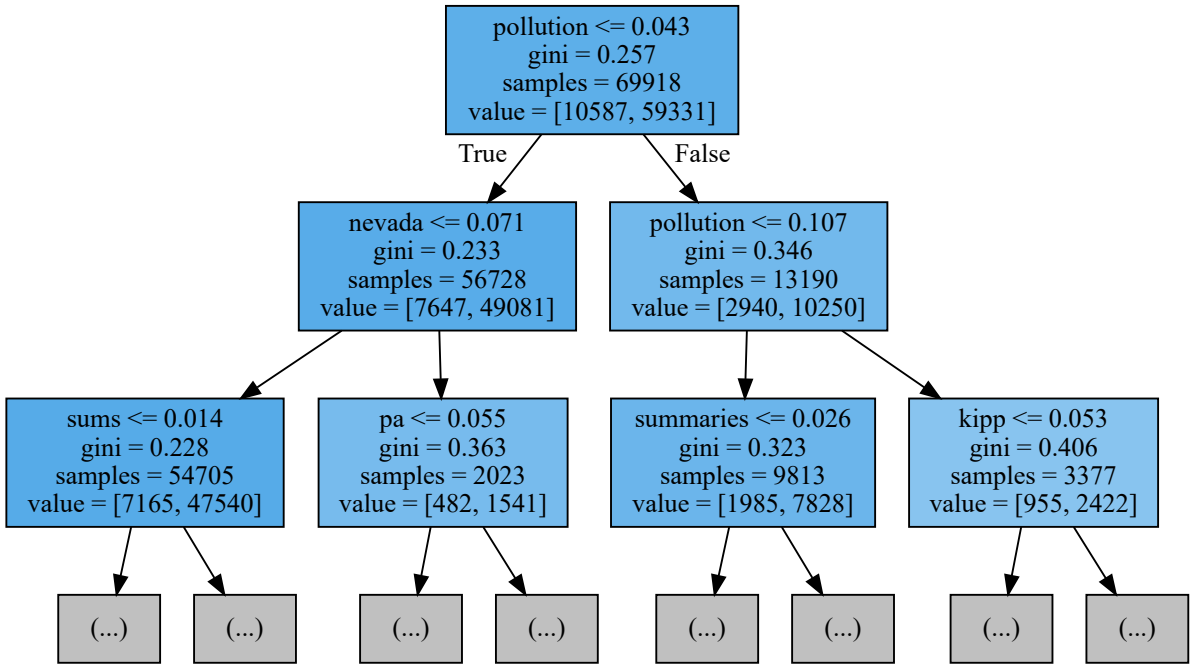
In [0]:
```
# Printing roc auc score
y_pred = model.predict_proba(x_te)[:,1]
roc_auc_score(y_true=y_test, y_score=y_pred)
```

Out[0]: 0.5882814394250147

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

In [0]:
```
from sklearn.tree import export_graphviz
import graphviz

export_graphviz(model, out_file="dt_tfidf2.dot", feature_names = feature_df.fe
atures.values, max_depth = 2, filled = True)
with open("dt_tfidf2.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```

Out[0]:



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

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

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

In [0]:
```
bow_vec_essay_tr,bow_vec_essay_val,bow_vec_essay_te,bow_vec_essay_list = bow_v
ec(textpreprocessed(X_train,'essay'),textpreprocessed(X_val,'essay'), textprep
rocessed(X_test,'essay'))
bow_vec_titles_tr,bow_vec_titles_val,bow_vec_titles_te,bow_vec_titles_list = b
ow_vec(textpreprocessed(X_train,'project_title'),textpreprocessed(X_val,'proje
ct_title'), textpreprocessed(X_test,'project_title'))
bow_vec_resource_tr,bow_vec_resource_val,bow_vec_resource_te,bow_vec_resource_
list = bow_vec(textpreprocessed(X_train,'project_resource_summary'),textprepro
cessed(X_val,'project_resource_summary'), textpreprocessed(X_test,'project_res
ource_summary'))
```

```
100%|██████████| 69918/69918 [00:42<00:00, 1636.98it/s]
100%|██████████| 17480/17480 [00:10<00:00, 1641.36it/s]
100%|██████████| 21850/21850 [00:13<00:00, 1640.22it/s]

Shape of matrix after one hot encodig  (69918, 13860)
Shape of matrix after one hot encodig  (17480, 13860)

  5%|▌         | 3506/69918 [00:00<00:01, 35055.44it/s]

Shape of matrix after one hot encodig  (21850, 13860)

100%|██████████| 69918/69918 [00:01<00:00, 35453.11it/s]
100%|██████████| 17480/17480 [00:00<00:00, 34268.87it/s]
100%|██████████| 21850/21850 [00:00<00:00, 35190.84it/s]

Shape of matrix after one hot encodig  (69918, 2476)
Shape of matrix after one hot encodig  (17480, 2476)

  2%|▏          | 1564/69918 [00:00<00:04, 15633.75it/s]

Shape of matrix after one hot encodig  (21850, 2476)

100%|██████████| 69918/69918 [00:04<00:00, 15578.43it/s]
100%|██████████| 17480/17480 [00:01<00:00, 15516.07it/s]
100%|██████████| 21850/21850 [00:01<00:00, 15411.23it/s]

Shape of matrix after one hot encodig  (69918, 4657)
Shape of matrix after one hot encodig  (17480, 4657)
Shape of matrix after one hot encodig  (21850, 4657)
```

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

```
21075
```

In [0]:
```
X_tr, X_cr, X_te = hstack_data(bow_vec_titles_tr, bow_vec_titles_val, bow_vec_
titles_te, \
                   bow_vec_essay_tr, bow_vec_essay_val, bow_vec_essay_te,\
                   bow_vec_resource_tr, bow_vec_resource_val, bow_vec_resource
_te)
```

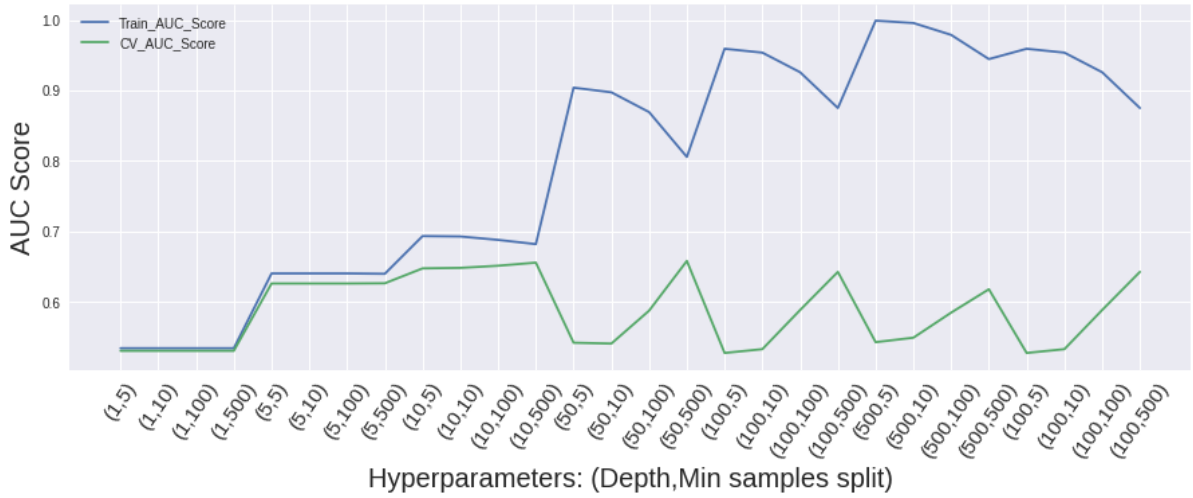
In [0]:
```
X_tr.shape,X_cr.shape,X_te.shape
```

Out[0]:
```
((69918, 21075), (17480, 21075), (21850, 21075))
```

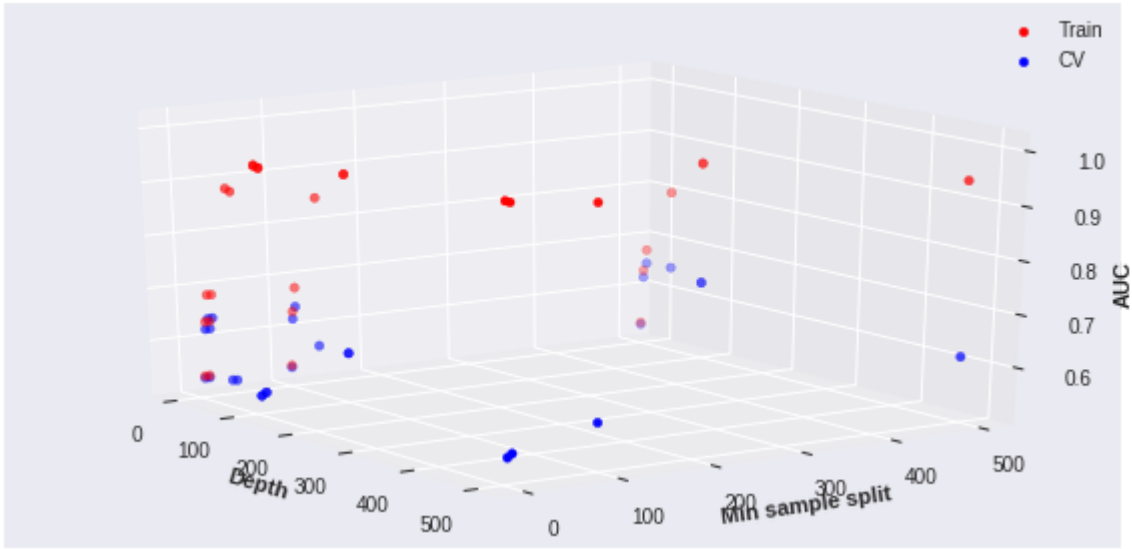
In [0]:
```
optimal_hyp(X_tr,y_train,X_cr,y_val)
```

```
execution time in minutes: 50.46486276785533
execution time in hours: 0
```

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



In [0]:
```
x = df1['depth'].astype(float).values
y = df1['min_samples_split'].astype(float).values
z1 = df1['Train_AUC_Score'].astype(float).values
z2 = df1['CV_AUC_Score'].astype(float).values
plot_3D(x,y,z1,z2)
```



In [0]:
```
best_d,sample_split,tr_auc_score,cv_auc_score = df1.iloc[df1.CV_AUC_Score.argmax()]
print("optimal depth:",best_d,"\nMin sample split:",sample_split,"\nCV AUC score:", cv_auc_score)
```

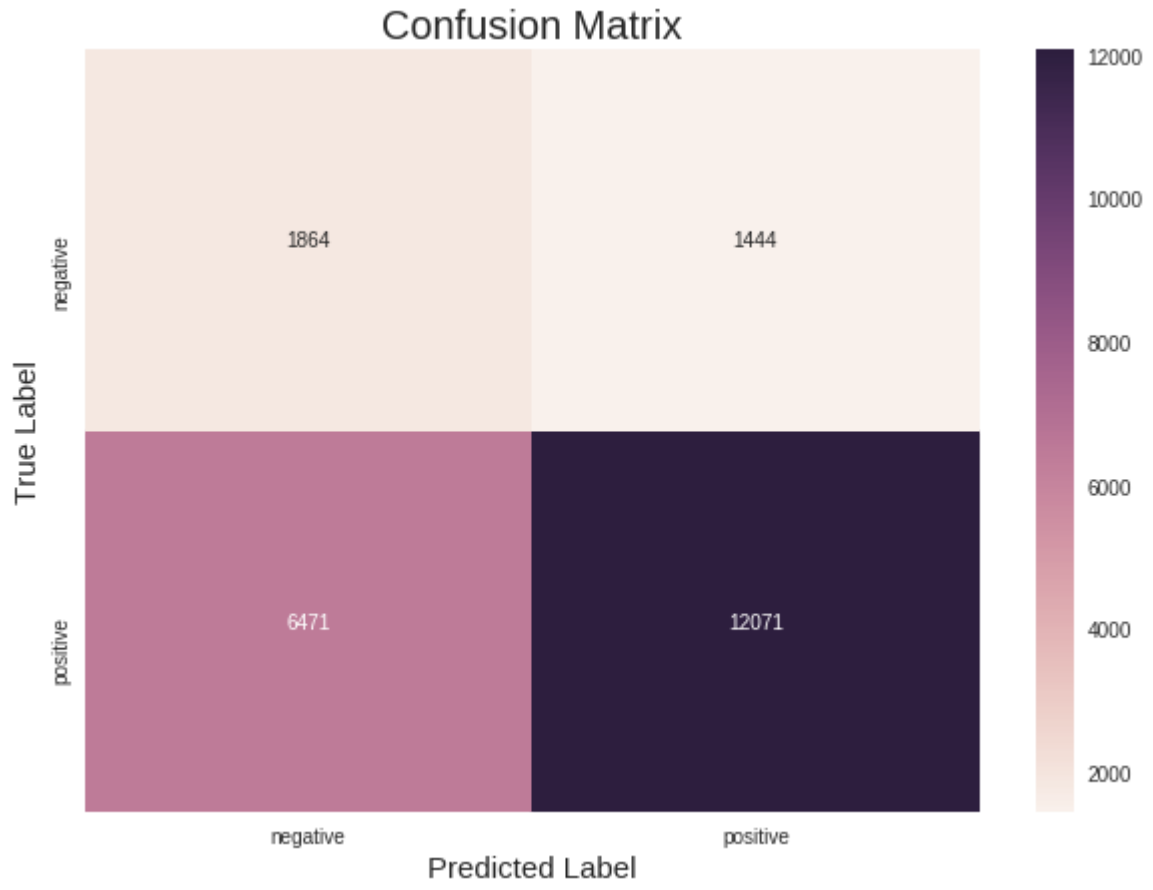
```
optimal depth: 50
Min sample split: 500
CV AUC score: 0.6579035284433918
```

In [0]:
```
model = DecisionTreeClassifier(max_depth = int(best_d), min_samples_split = in
t(sample_split), class_weight = 'balanced',random_state = 0)
model.fit(X_tr, y_train)
```

Out[0]:
```
DecisionTreeClassifier(class_weight='balanced', criterion='gini',
            max_depth=50, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=500,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')
```

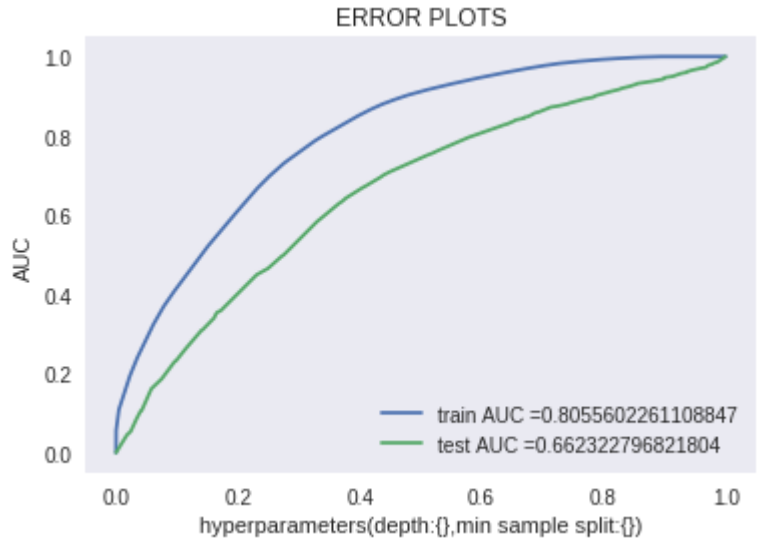
In [0]:
```
y_pred_te = model.predict(X_te)
```

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




```
True Positives : 12071
False Positives : 1444
True Negatives : 1864
False Negatives : 6471
```

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: 1.222061848640442
execution time in hours: 0
```

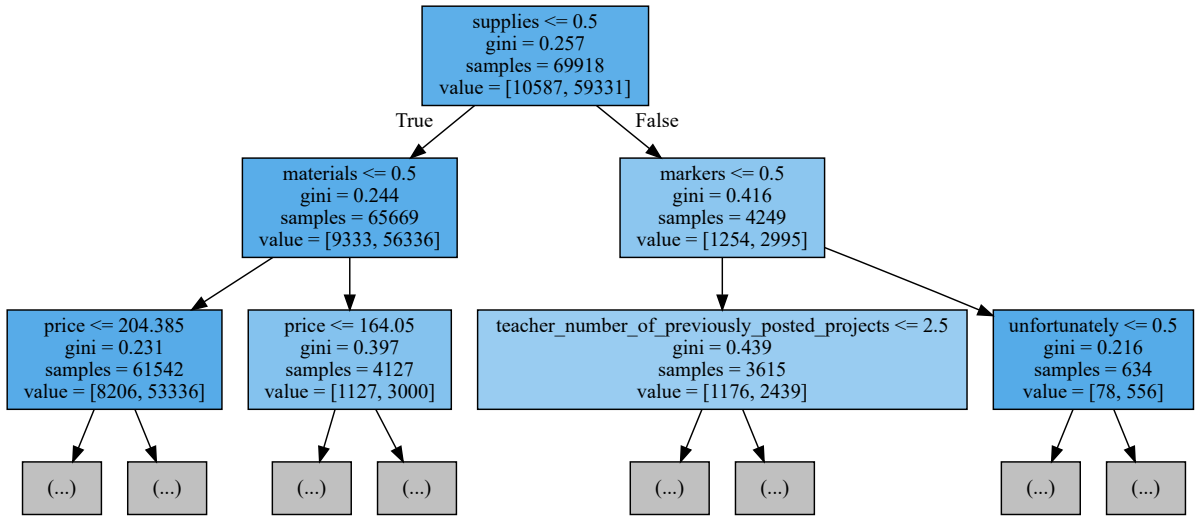
In [0]:
```
# Printing roc auc score
y_pred = model.predict_proba(X_te)[:,1]
roc_auc_score(y_true=y_test, y_score=y_pred)
```

Out[0]: 0.662322796821804

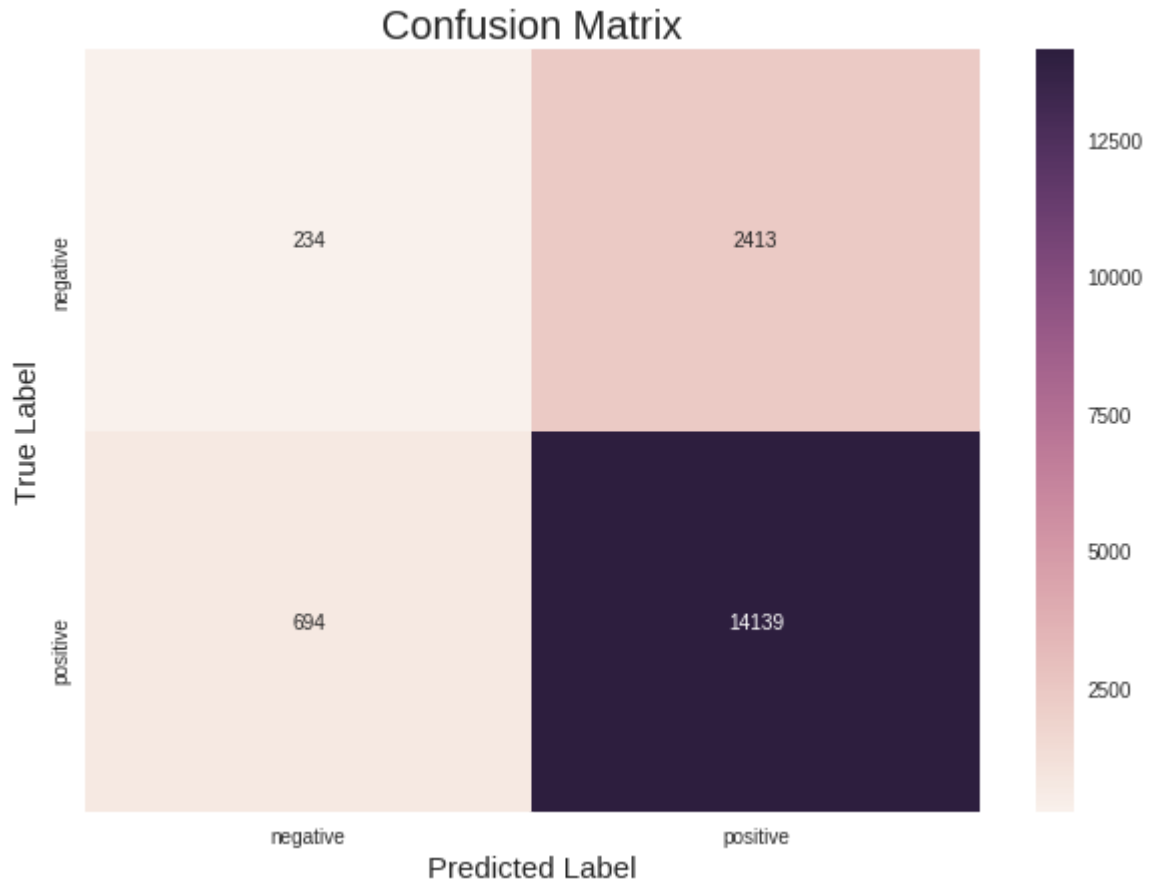
In [0]:
```
%cd -
from sklearn.tree import export_graphviz
import graphviz

export_graphviz(model, out_file="dt_bow.dot", feature_names = feature_names, m
ax_depth = 2, filled = True)
with open("dt_bow.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```

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

Out[0]:



In [0]:
```
y_pred_val = model.predict(X_cr)
labels = ["negative","positive"]
get_confusion_matrix_values(np.array(y_val), y_pred_val)
```




```
True Positives : 14139
False Positives : 2413
True Negatives : 234
False Negatives : 694
```

In [0]:
```
ind_bool_fp = [i==0 and j==1 for i, j in zip(y_val, y_pred_val)]
ind_fp =list(np.where(ind_bool_fp)[0])
ind_bool_fn = [i==1 and j==0 for i, j in zip(y_val, y_pred_val)]
ind_fn =list(np.where(ind_bool_fn)[0])
ind_bool_tp = [i==1 and j==1 for i, j in zip(y_val, y_pred_val)]
ind_tp =list(np.where(ind_bool_tp)[0])
ind_bool_tn = [i==0 and j==0 for i, j in zip(y_val, y_pred_val)]
ind_tn =list(np.where(ind_bool_tn)[0])
```

In [0]:
```
print("False Positive:",len(ind_fp))
print("True Positive:",len(ind_tp))
print("False Negative:",len(ind_fn))
print("True Negative:",len(ind_tn))
```

```
False Positive: 2413
True Positive: 14139
False Negative: 694
True Negative: 234
```

In [0]:
```
df = X_cr.toarray()
```

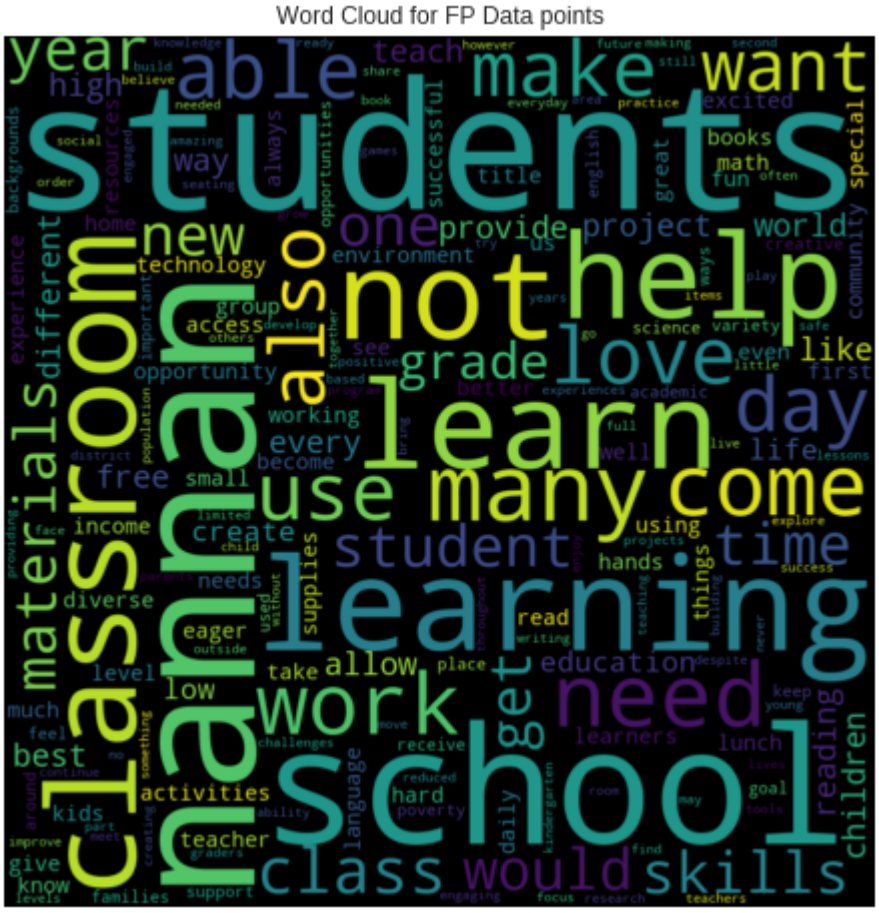
In [0]:
```
df_fp = df[ind_fp,:]
df_fn = df[ind_fn,:]
df_tp = df[ind_tp,:]
df_tn = df[ind_tn,:]
```

In [0]:
```
start_essay = len([*feature_list , *bow_vec_titles_list])
end_essay = len([*feature_list , *bow_vec_titles_list, *bow_vec_essay_list])-1
```

In [0]:
```
df_fp_words = df_fp[:,start_essay:end_essay+1]
df_fp_words =pd.DataFrame(df_fp_words)
df_fp_words.columns = bow_vec_essay_list
x=df_fp_words[df_fp_words > 0].count()
```

Plotting the WordCloud

In [0]:
```
import matplotlib.pyplot as plt
from wordcloud import WordCloud

wordcloud = WordCloud(width = 800, height = 800,)
wordcloud.generate_from_frequencies(frequencies=x)
plt.figure(figsize = (8, 8), facecolor = None)
plt.title("Word Cloud for FP Data points")
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```



These are the top 100 words which corresponds to FP datapoints

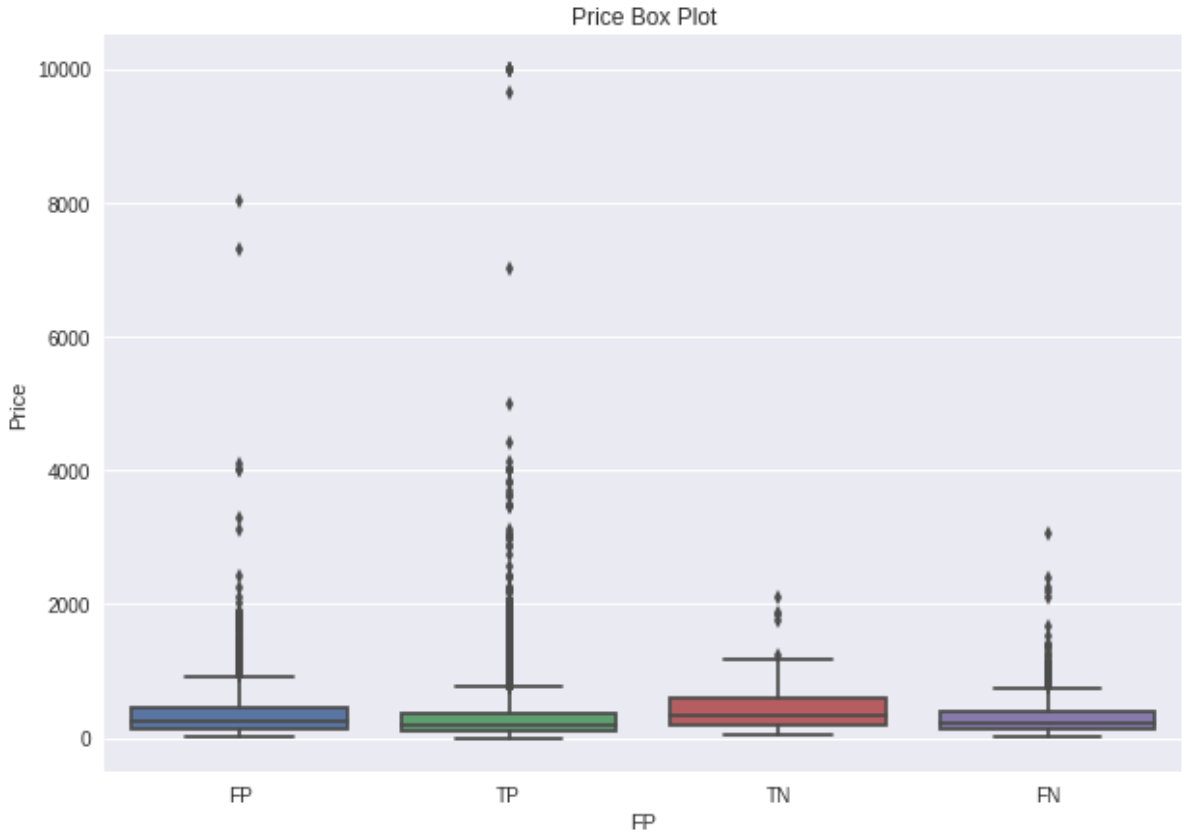
In [0]:
```
ind_price = feature_names.index('price')
ind_prev_proj = feature_names.index('teacher_number_of_previously_posted_projects')
```

In [0]:
```
price_fp = df_fp[:,ind_price]
prev_proj_fp = df_fp[:,ind_prev_proj]

price_fn = df_fn[:,ind_price]
prev_proj_fn = df_fn[:,ind_prev_proj]

price_tp = df_tp[:,ind_price]
prev_proj_tp = df_tp[:,ind_prev_proj]

price_tn = df_tn[:,ind_price]
prev_proj_tn = df_tn[:,ind_prev_proj]
```

Plotting Box Plot of Price of false positive data points

In [0]:
```
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

# Make boxplot for one group only
plt.figure(figsize=(10,7))
ax = sns.boxplot(data=[price_fp,price_tp,price_tn,price_fn])
ax.set_xticklabels(['FP','TP','TN','FN'])
ax.set_title("Price Box Plot")
ax.set_ylabel('Price')
ax.set_xlabel('FP')
#sns.plt.show()
```

Out[0]: Text(0.5, 0, 'FP')



Observation: By looking at 75th percentile of FP and TP we can conclude if the price is more than 368 It will correspond to the FP.

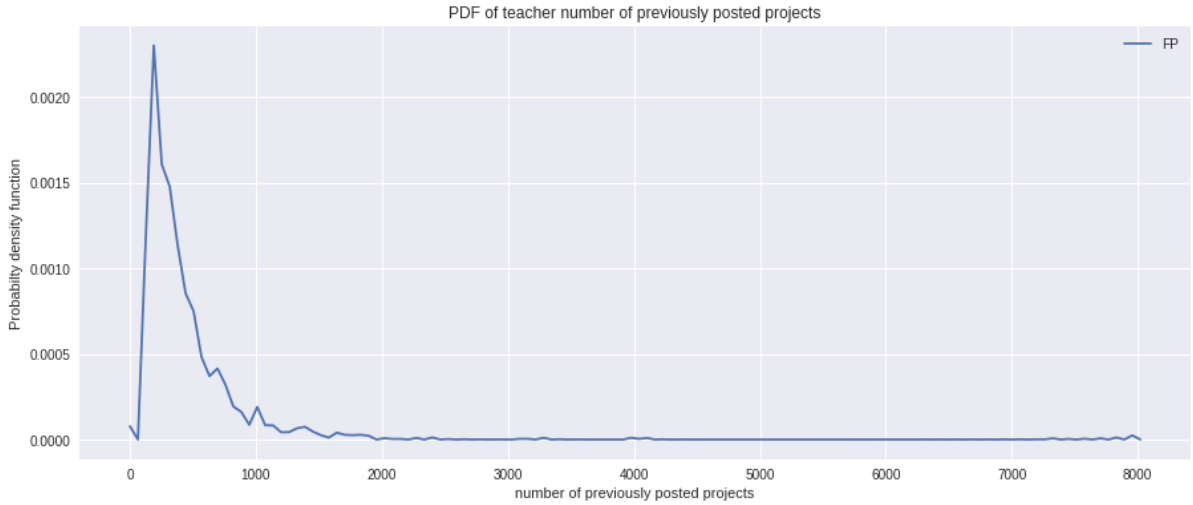
In [0]:
```
p25 = [np.percentile(label, 25) for label in [price_fp, price_tp, price_tn, pr
ice_fn]] # return 25th percentile.
p50 = [np.percentile(label, 50) for label in [price_fp, price_tp, price_tn, pr
ice_fn]] # return 50th percentile.
p75 = [np.percentile(label, 75) for label in [price_fp, price_tp, price_tn, pr
ice_fn]] # return 75th percentile.
per = pd.DataFrame([p25,p50,p75],columns=['FP','TP','TN','FN'],index=['25th Pe
rcentile','50th Percentile','75th Percentile'])
per
```

Out[0]:

	FP	TP	TN	FN
25th Percentile	132.95	99.50	192.075	129.965
50th Percentile	254.97	198.95	338.975	223.740
75th Percentile	452.99	368.00	588.150	379.555

Plotting the pdf with the teacher number of previously posted projects of false positive data points

In [0]:
```
plt.figure(figsize=(15,6))
ax = sns.kdeplot(price_fp, label="FP", bw=0.5)
#ax = sns.kdeplot(price_tp, label="TP", bw=0.5)
#ax = sns.kdeplot(price_fn, label="FN", bw=0.5)
#ax = sns.kdeplot(price_tn, label="TN", bw=0.5)

ax.set_title("PDF of teacher number of previously posted projects")
ax.set_ylabel('Probabilty density function')
ax.set_xlabel('number of previously posted projects')
plt.legend()
plt.show()
```



Observation: The occurence of 200-500 no.of previously posted project is more in FP.

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

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

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

In [0]:
```
#%cd Assignments_DonorsChoose_2018
# stronging variables into pickle files python: http://www.jessicayung.com/how
-to-use-pickle-to-save-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 [0]:
```
avgw2v_vec_essay_tr = avg_w2v(textpreprocessed(X_train,'essay'))
avgw2v_vec_essay_te = avg_w2v(textpreprocessed(X_test,'essay'))
avgw2v_vec_essay_val = avg_w2v(textpreprocessed(X_val,'essay'))
```

```
100%|██████████| 69918/69918 [00:43<00:00, 1622.35it/s]
100%|██████████| 69918/69918 [00:20<00:00, 3457.16it/s]
  1%|          | 163/21850 [00:00<00:13, 1628.69it/s]

69918
30

100%|██████████| 21850/21850 [00:13<00:00, 1633.87it/s]
100%|██████████| 21850/21850 [00:06<00:00, 3428.52it/s]
  1%|          | 166/17480 [00:00<00:10, 1654.86it/s]

21850
30

100%|██████████| 17480/17480 [00:10<00:00, 1626.80it/s]
100%|██████████| 17480/17480 [00:05<00:00, 3417.67it/s]

17480
30
```

In [0]:
```
avgw2v_vec_titles_tr = avg_w2v(textpreprocessed(X_train,'project_title'))
avgw2v_vec_titles_te = avg_w2v(textpreprocessed(X_test,'project_title'))
avgw2v_vec_titles_val = avg_w2v(textpreprocessed(X_val,'project_title'))
```

```
100%|██████████| 69918/69918 [00:02<00:00, 34036.46it/s]
100%|██████████| 69918/69918 [00:00<00:00, 81549.40it/s]
 16%|██        | 3511/21850 [00:00<00:00, 35101.41it/s]

69918
30

100%|██████████| 21850/21850 [00:00<00:00, 34984.13it/s]
100%|██████████| 21850/21850 [00:00<00:00, 79858.58it/s]
 20%|██        | 3507/17480 [00:00<00:00, 35063.01it/s]

21850
30

100%|██████████| 17480/17480 [00:00<00:00, 35121.71it/s]
100%|██████████| 17480/17480 [00:00<00:00, 81232.10it/s]

17480
30
```

In [0]:
```
avgw2v_vec_resource_tr = avg_w2v(textpreprocessed(X_train,'project_resource_summary'))
avgw2v_vec_resource_te = avg_w2v(textpreprocessed(X_test,'project_resource_summary'))
avgw2v_vec_resource_val = avg_w2v(textpreprocessed(X_val,'project_resource_summary'))
```

```
100%|██████████| 69918/69918 [00:04<00:00, 15430.06it/s]
100%|██████████| 69918/69918 [00:02<00:00, 33523.42it/s]
  7%|▊         | 1561/21850 [00:00<00:01, 15609.90it/s]

69918
30

100%|██████████| 21850/21850 [00:01<00:00, 15279.90it/s]
100%|██████████| 21850/21850 [00:00<00:00, 32556.24it/s]
  9%|▉         | 1534/17480 [00:00<00:01, 15339.94it/s]

21850
30

100%|██████████| 17480/17480 [00:01<00:00, 15364.59it/s]
100%|██████████| 17480/17480 [00:00<00:00, 32656.97it/s]

17480
30
```

In [0]:
```
X_tr, X_cr, X_te = hstack_data(avgw2v_vec_titles_tr, avgw2v_vec_titles_val, avgw2v_vec_titles_te, \
                    avgw2v_vec_essay_tr, avgw2v_vec_essay_val, avgw2v_vec_essay_te,\
                    avgw2v_vec_resource_tr, avgw2v_vec_resource_val, avgw2v_vec_resource_te)
```

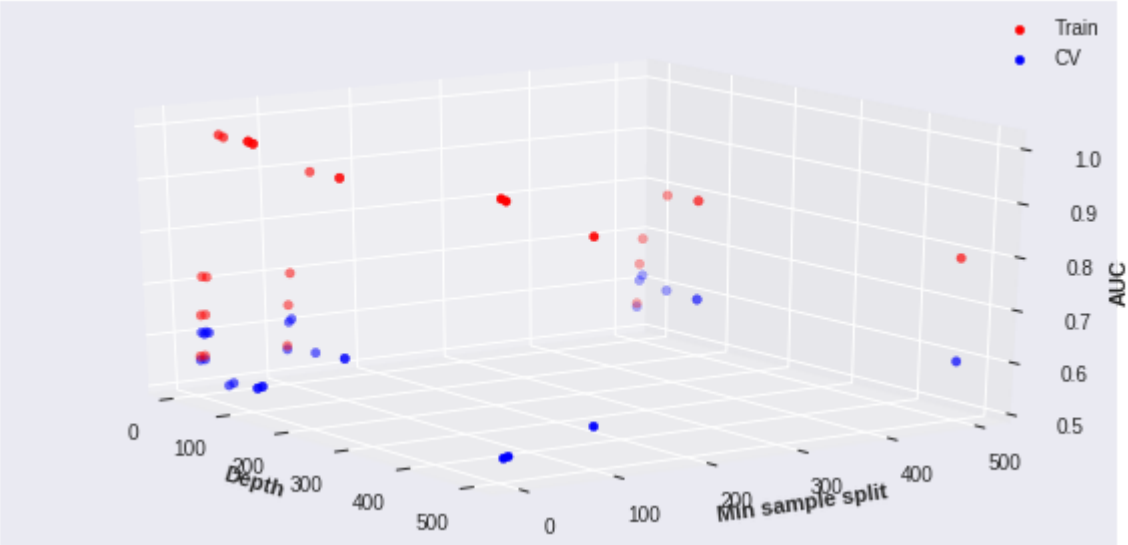
In [0]:
```
X_tr.shape,X_cr.shape,X_te.shape
```

Out[0]:
```
((69918, 172), (17480, 172), (21850, 172))
```

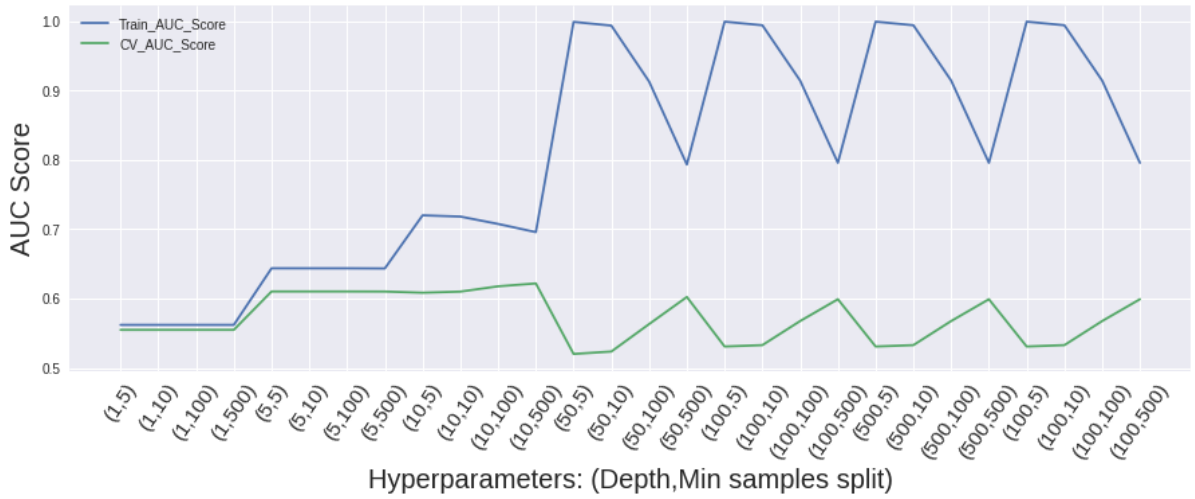
In [0]:
```
optimal_hyp(X_tr,y_train,X_cr,y_val)
```

```
execution time in minutes: 22.421148618062336
execution time in hours: 0
```

In [0]:
```
x = df1['depth'].astype(float).values
y = df1['min_samples_split'].astype(float).values
z1 = df1['Train_AUC_Score'].astype(float).values
z2 = df1['CV_AUC_Score'].astype(float).values
plot_3D(x,y,z1,z2)
```



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



In [0]:
```
best_d,sample_split,tr_auc_score,cv_auc_score = df1.iloc[df1.CV_AUC_Score.argmax()]
print("optimal depth:",best_d,"\nMin sample split:",sample_split,"\nCV AUC score:", cv_auc_score)
```

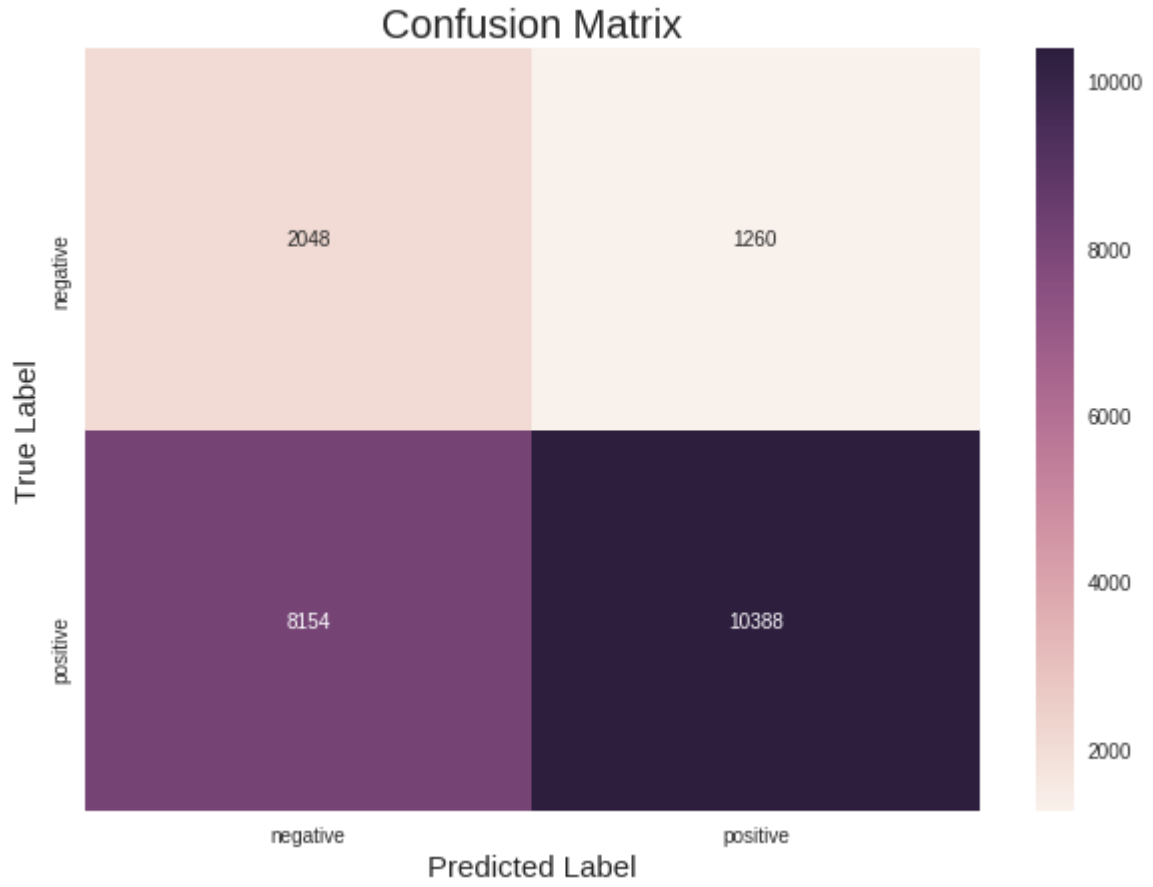
```
optimal depth: 10
Min sample split: 500
CV AUC score: 0.6217583721610738
```

In [0]:
```
model = DecisionTreeClassifier(max_depth = int(best_d), min_samples_split = in
t(sample_split), class_weight = 'balanced',random_state = 0)
model.fit(X_tr, y_train)
```

Out[0]:
```
DecisionTreeClassifier(class_weight='balanced', criterion='gini',
            max_depth=10, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=500,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')
```

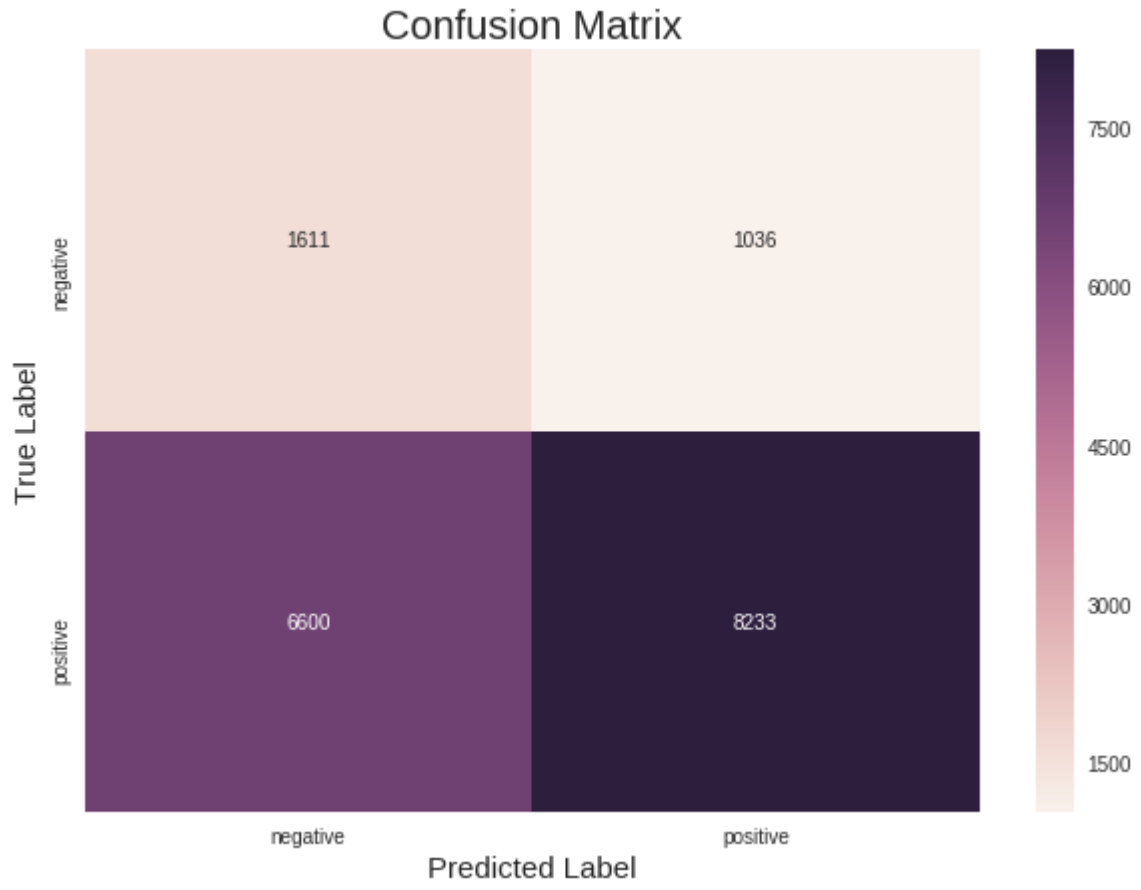
In [0]:
```
y_pred_te = model.predict(X_te)
```

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



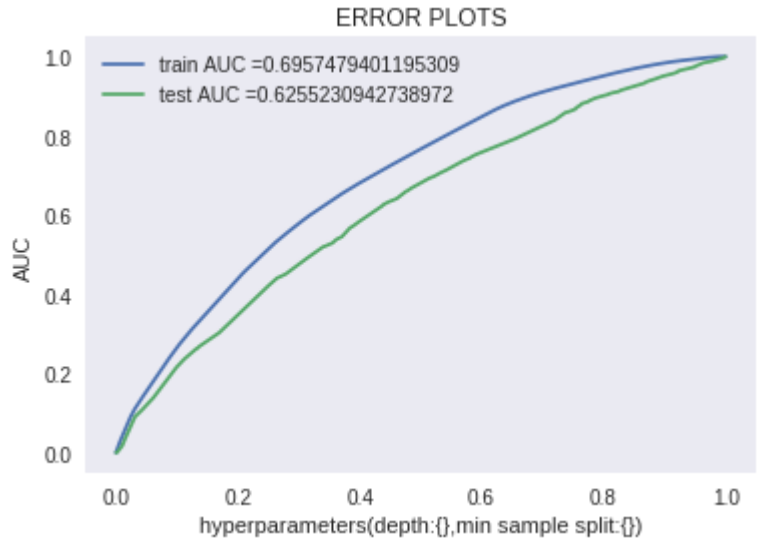

```
True Positives : 10388
False Positives : 1260
True Negatives : 2048
False Negatives : 8154
```

In [0]:
```
y_pred_val = model.predict(X_cr)
labels = ["negative","positive"]
get_confusion_matrix_values(np.array(y_val), y_pred_val)
```




```
True Positives : 8233
False Positives : 1036
True Negatives : 1611
False Negatives : 6600
```

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: 0.27576781511306764
execution time in hours: 0
```

In [0]:
```
# Printing roc auc score
y_pred = model.predict_proba(X_te)[:,1]
roc_auc_score(y_true=y_test, y_score=y_pred)
```

Out[0]: 0.6255230942738972

In [0]:
```
ind_bool_fp = [i==0 and j==1 for i, j in zip(y_val, y_pred_val)]
ind_fp =list(np.where(ind_bool_fp)[0])
ind_bool_fn = [i==1 and j==0 for i, j in zip(y_val, y_pred_val)]
ind_fn =list(np.where(ind_bool_fn)[0])
ind_bool_tp = [i==1 and j==1 for i, j in zip(y_val, y_pred_val)]
ind_tp =list(np.where(ind_bool_tp)[0])
ind_bool_tn = [i==0 and j==0 for i, j in zip(y_val, y_pred_val)]
ind_tn =list(np.where(ind_bool_tn)[0])
```

In [0]:
```
print("False Positive:",len(ind_fp))
print("True Positive:",len(ind_tp))
print("False Negative:",len(ind_fn))
print("True Negative:",len(ind_tn))
```

```
False Positive: 1036
True Positive: 8233
False Negative: 6600
True Negative: 1611
```

In [0]:
```
df = X_cr.toarray()
```

In [0]:
```
df_fp = df[ind_fp,:]
df_fn = df[ind_fn,:]
df_tp = df[ind_tp,:]
df_tn = df[ind_tn,:]
```

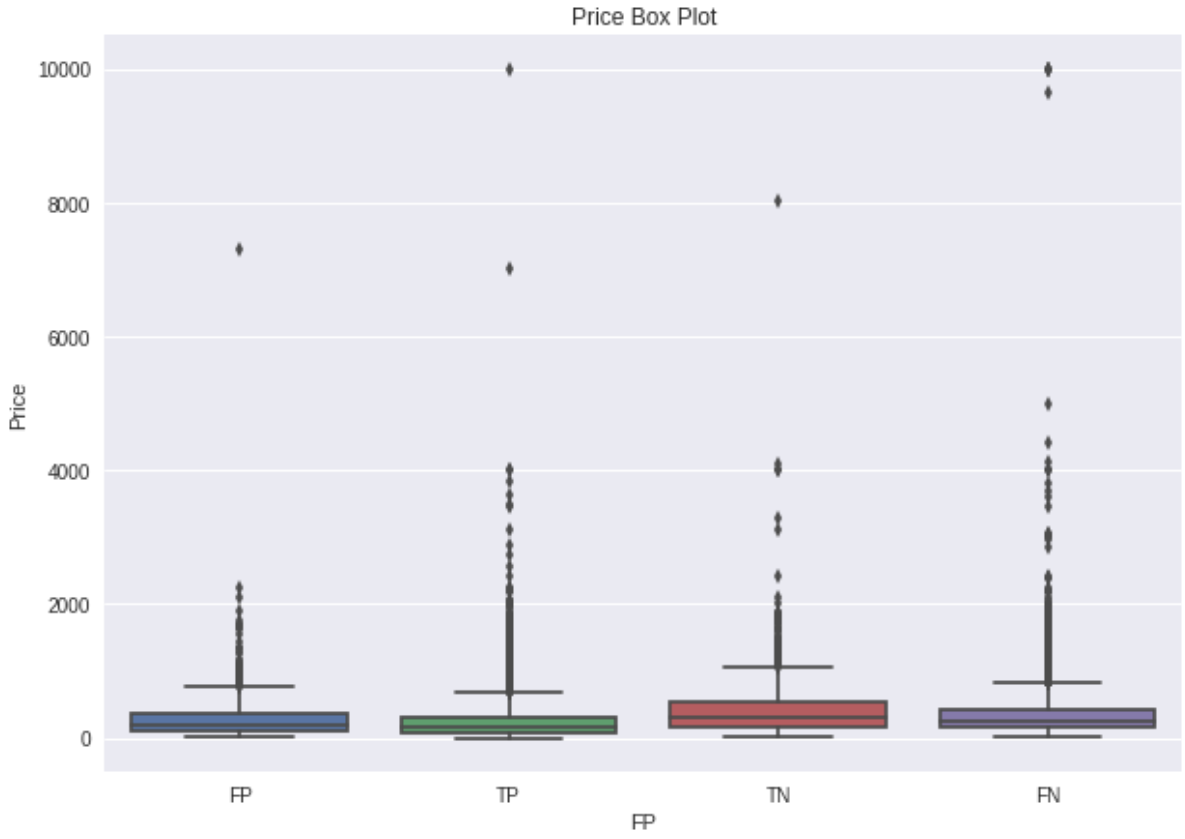
In [0]:
```
ind_price = feature_names.index('price')
ind_prev_proj = feature_names.index('teacher_number_of_previously_posted_proje
cts')
```

In [0]:
```
price_fp = df_fp[:,ind_price]
prev_proj_fp = df_fp[:,ind_prev_proj]

price_fn = df_fn[:,ind_price]
prev_proj_fn = df_fn[:,ind_prev_proj]

price_tp = df_tp[:,ind_price]
prev_proj_tp = df_tp[:,ind_prev_proj]

price_tn = df_tn[:,ind_price]
prev_proj_tn = df_tn[:,ind_prev_proj]
```

Plotting Box Plot of Price of false positive data points

In [0]:
```
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

# Make boxplot for one group only
plt.figure(figsize=(10,7))
ax = sns.boxplot(data=[price_fp,price_tp,price_tn,price_fn])
ax.set_xticklabels(['FP','TP','TN','FN'])
ax.set_title("Price Box Plot")
ax.set_ylabel('Price')
ax.set_xlabel('FP')
#sns.plt.show()
```

Out[0]: Text(0.5, 0, 'FP')



Observation: By looking at 75th percentile of FP and TP we can conclude If the price is above 312 It will correspond to FP.

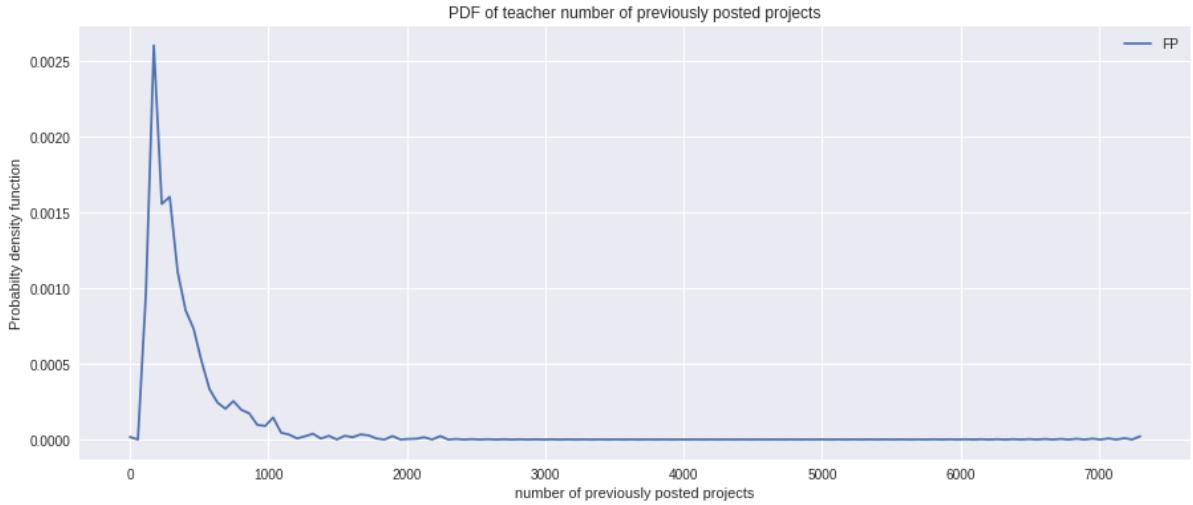
In [0]:
```
p25 = [np.percentile(label, 25) for label in [price_fp, price_tp, price_tn, pr
ice_fn]] # return 25th percentile.
p50 = [np.percentile(label, 50) for label in [price_fp, price_tp, price_tn, pr
ice_fn]] # return 50th percentile.
p75 = [np.percentile(label, 75) for label in [price_fp, price_tp, price_tn, pr
ice_fn]] # return 75th percentile.
per = pd.DataFrame([p25,p50,p75],columns=['FP','TP','TN','FN'],index=['25th Pe
rcentile','50th Percentile','75th Percentile'])
per
```

Out[0]:

	FP	TP	TN	FN
25th Percentile	92.0850	59.99	166.87	146.6175
50th Percentile	199.0000	163.73	302.67	249.0850
75th Percentile	361.9675	312.67	526.52	413.4200

Plotting the pdf with the teacher number of previously posted projects of false positive data points

In [0]:
```
plt.figure(figsize=(15,6))
ax = sns.kdeplot(price_fp, label="FP", bw=0.5)
#ax = sns.kdeplot(price_tp, label="TP", bw=0.5)
#ax = sns.kdeplot(price_fn, label="FN", bw=0.5)
#ax = sns.kdeplot(price_tn, label="TN", bw=0.5)

ax.set_title("PDF of teacher number of previously posted projects")
ax.set_ylabel('Probabilty density function')
ax.set_xlabel('number of previously posted projects')
plt.legend()
plt.show()
```



Observation: 200-300 no of previously posted project has more no of occurence in FP.

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

In [0]:
```
#%cd Assignments_DonorsChoose_2018
# stronging variables into pickle files python: http://www.jessicayung.com/how
-to-use-pickle-to-save-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]:
```
# 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 store
d 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/rev
iew
      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.spl
it())) # getting the tfidf value for each word
              vector += (vec * tf_idf) # calculating tfidf weighted w2v
              tf_idf_weight += tf_idf
      if tf_idf_weight != 0:
          vector /= tf_idf_weight
      tfidf_w2v_vectors_list.append(vector)

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

In [0]:
```
tfidfw2v_vec_essay_tr = tfidf_w2v(textpreprocessed(X_train,'essay'))
tfidfw2v_vec_essay_te = tfidf_w2v(textpreprocessed(X_test,'essay'))
tfidfw2v_vec_essay_val = tfidf_w2v(textpreprocessed(X_val,'essay'))
```

```
100%|██████████| 69918/69918 [00:43<00:00, 1622.40it/s]
100%|██████████| 69918/69918 [02:05<00:00, 559.09it/s]
  1%|          | 168/21850 [00:00<00:12, 1673.26it/s]

69918
30

100%|██████████| 21850/21850 [00:13<00:00, 1635.50it/s]
100%|██████████| 21850/21850 [00:40<00:00, 545.66it/s]
  1%|          | 162/17480 [00:00<00:10, 1618.87it/s]

21850
30

100%|██████████| 17480/17480 [00:10<00:00, 1619.20it/s]
100%|██████████| 17480/17480 [00:32<00:00, 545.14it/s]

17480
30
```

In [0]:
```
tfidfw2v_vec_titles_tr = tfidf_w2v(textpreprocessed(X_train,'project_title'))
tfidfw2v_vec_titles_te = tfidf_w2v(textpreprocessed(X_test,'project_title'))
tfidfw2v_vec_titles_val = tfidf_w2v(textpreprocessed(X_val,'project_title'))
```

```
100%|██████████| 69918/69918 [00:02<00:00, 34490.95it/s]
100%|██████████| 69918/69918 [00:01<00:00, 40790.25it/s]
 15%|██        | 3375/21850 [00:00<00:00, 33741.59it/s]

69918
30

100%|██████████| 21850/21850 [00:00<00:00, 34697.07it/s]
100%|██████████| 21850/21850 [00:00<00:00, 41540.69it/s]
 20%|██        | 3413/17480 [00:00<00:00, 34124.34it/s]

21850
30

100%|██████████| 17480/17480 [00:00<00:00, 34704.25it/s]
100%|██████████| 17480/17480 [00:00<00:00, 41275.29it/s]

17480
30
```

In [0]:
```
tfidfw2v_vec_resource_tr = tfidf_w2v(textpreprocessed(X_train,'project_resource_summary'))
tfidfw2v_vec_resource_te = tfidf_w2v(textpreprocessed(X_test,'project_resource_summary'))
tfidfw2v_vec_resource_val = tfidf_w2v(textpreprocessed(X_val,'project_resource_summary'))
```

```
100%|██████████| 69918/69918 [00:04<00:00, 15369.74it/s]
100%|██████████| 69918/69918 [00:05<00:00, 12747.78it/s]
  7%|▊         | 1503/21850 [00:00<00:01, 15026.65it/s]

69918
30

100%|██████████| 21850/21850 [00:01<00:00, 15342.53it/s]
100%|██████████| 21850/21850 [00:01<00:00, 12651.80it/s]
  9%|▉         | 1488/17480 [00:00<00:01, 14877.18it/s]

21850
30

100%|██████████| 17480/17480 [00:01<00:00, 15315.96it/s]
100%|██████████| 17480/17480 [00:01<00:00, 13065.79it/s]

17480
30
```

In [0]:
```
X_tr, X_cr, X_te = hstack_data(tfidfw2v_vec_titles_tr, tfidfw2v_vec_titles_val, tfidfw2v_vec_titles_te, \
                    tfidfw2v_vec_essay_tr, tfidfw2v_vec_essay_val, tfidfw2v_vec_essay_te,\
                    tfidfw2v_vec_resource_tr, tfidfw2v_vec_resource_val, tfidfw2v_vec_resource_te)
```

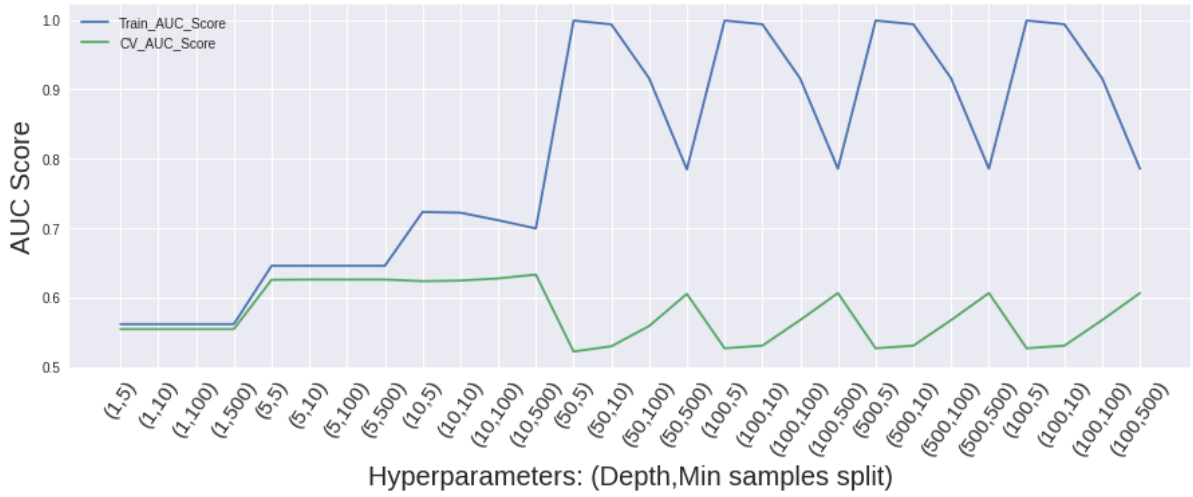
In [0]:
```
X_tr.shape,X_cr.shape,X_te.shape
```

Out[0]:
```
((69918, 172), (17480, 172), (21850, 172))
```

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

```
execution time in minutes: 21.055604934692383
execution time in hours: 0
```

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



In [0]:
```
best_d,sample_split,tr_auc_score,cv_auc_score = df1.iloc[df1.CV_AUC_Score.argmax()]
print("optimal depth:",best_d,"\nMin sample split:",sample_split,"\nCV AUC score:", cv_auc_score)
```

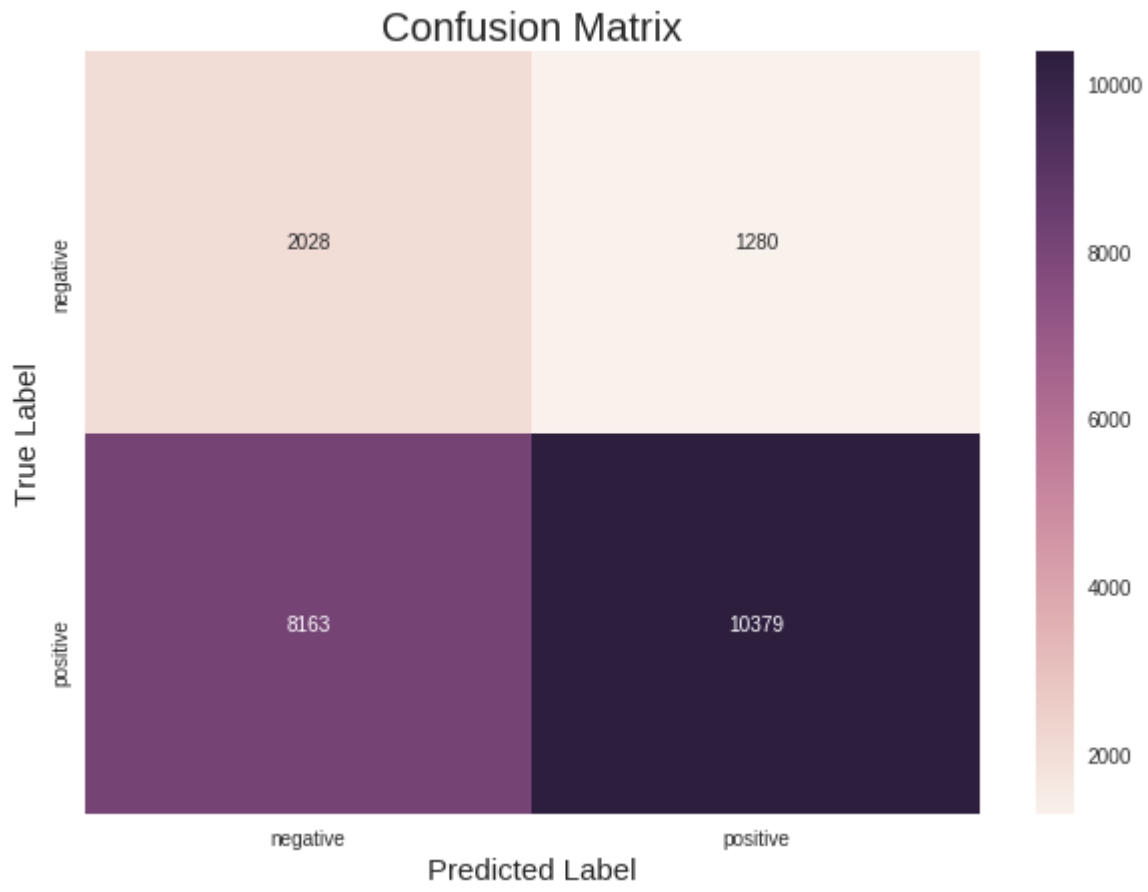
```
optimal depth: 10
Min sample split: 500
CV AUC score: 0.633279780727638
```

In [0]:
```
model = DecisionTreeClassifier(max_depth = int(best_d), min_samples_split = int(sample_split), class_weight = 'balanced',random_state = 0)
model.fit(X_tr, y_train)
```

Out[0]:
```
DecisionTreeClassifier(class_weight='balanced', criterion='gini',
            max_depth=10, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=500,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')
```

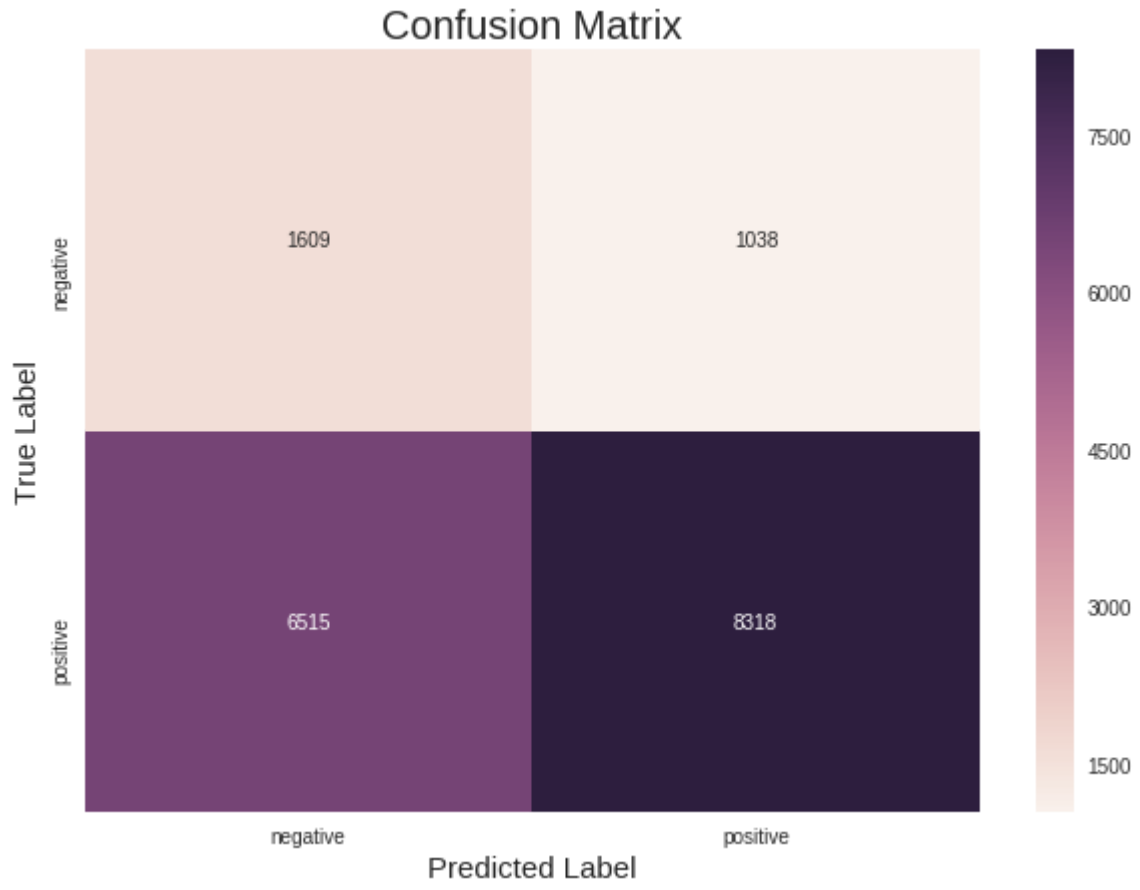
In [0]:
```
y_pred_te = model.predict(X_te)
```

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



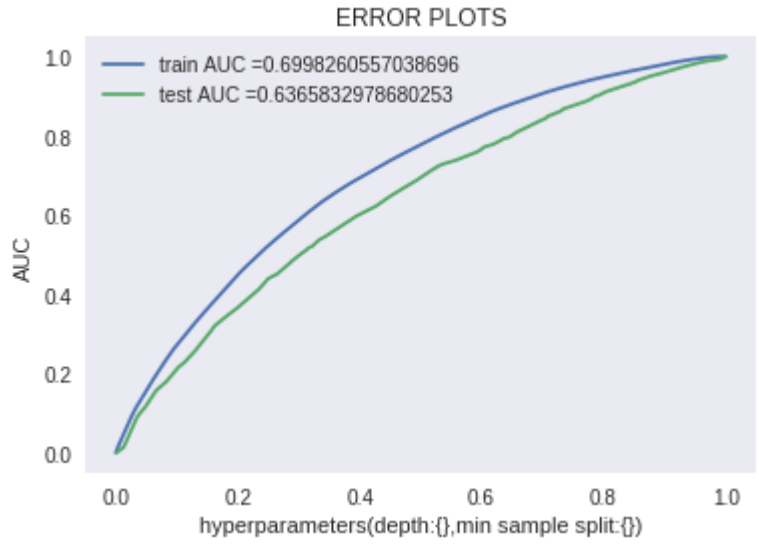

```
True Positives : 10379
False Positives : 1280
True Negatives : 2028
False Negatives : 8163
```

In [0]:
```
y_pred_val = model.predict(X_cr)
labels = ["negative","positive"]
get_confusion_matrix_values(np.array(y_val), y_pred_val)
```




```
True Positives : 8318
False Positives : 1038
True Negatives : 1609
False Negatives : 6515
```

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: 0.2837895671526591
execution time in hours: 0
```

In [0]:
```
# Printing roc auc score
y_pred = model.predict_proba(X_te)[:,1]
roc_auc_score(y_true=y_test, y_score=y_pred)
```

Out[0]: 0.6365832978680253

In [0]:
```
ind_bool_fp = [i==0 and j==1 for i, j in zip(y_val, y_pred_val)]
ind_fp =list(np.where(ind_bool_fp)[0])
ind_bool_fn = [i==1 and j==0 for i, j in zip(y_val, y_pred_val)]
ind_fn =list(np.where(ind_bool_fn)[0])
ind_bool_tp = [i==1 and j==1 for i, j in zip(y_val, y_pred_val)]
ind_tp =list(np.where(ind_bool_tp)[0])
ind_bool_tn = [i==0 and j==0 for i, j in zip(y_val, y_pred_val)]
ind_tn =list(np.where(ind_bool_tn)[0])
```

In [0]:
```
print("False Positive:",len(ind_fp))
print("True Positive:",len(ind_tp))
print("False Negative:",len(ind_fn))
print("True Negative:",len(ind_tn))
```

```
False Positive: 1038
True Positive: 8318
False Negative: 6515
True Negative: 1609
```

In [0]:
```
df = X_cr.toarray()
```

In [0]:
```
df_fp = df[ind_fp,:]
df_fn = df[ind_fn,:]
df_tp = df[ind_tp,:]
df_tn = df[ind_tn,:]
```

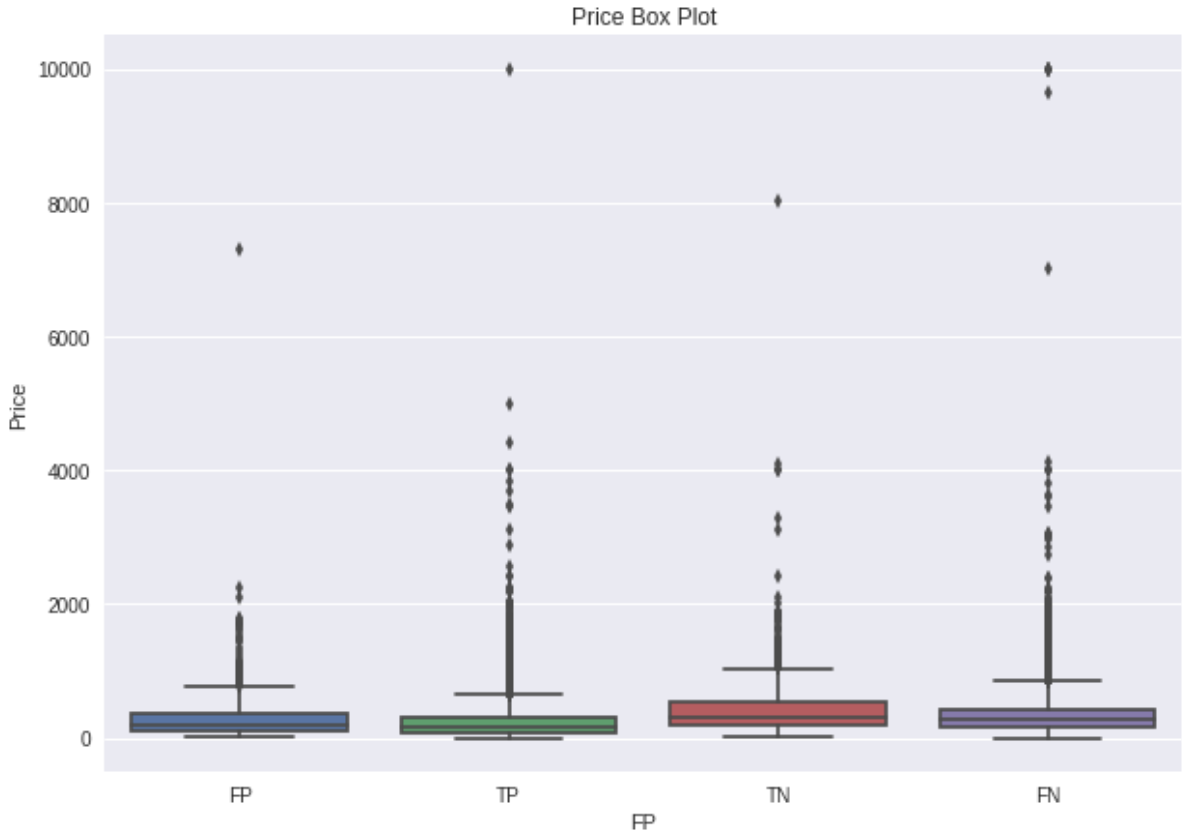
In [0]:
```
ind_price = feature_names.index('price')
ind_prev_proj = feature_names.index('teacher_number_of_previously_posted_proje
cts')
```

In [0]:
```
price_fp = df_fp[:,ind_price]
prev_proj_fp = df_fp[:,ind_prev_proj]

price_fn = df_fn[:,ind_price]
prev_proj_fn = df_fn[:,ind_prev_proj]

price_tp = df_tp[:,ind_price]
prev_proj_tp = df_tp[:,ind_prev_proj]

price_tn = df_tn[:,ind_price]
prev_proj_tn = df_tn[:,ind_prev_proj]
```

Plotting Box Plot of Price of false positive data points

In [0]:
```
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

# Make boxplot for one group only
plt.figure(figsize=(10,7))
ax = sns.boxplot(data=[price_fp,price_tp,price_tn,price_fn])
ax.set_xticklabels(['FP','TP','TN','FN'])
ax.set_title("Price Box Plot")
ax.set_ylabel('Price')
ax.set_xlabel('FP')
#sns.plt.show()
```

Out[0]: Text(0.5, 0, 'FP')



Observation: By comparing 75th percentile of TP and FP if the price is above 300 chances of being FP will be more.

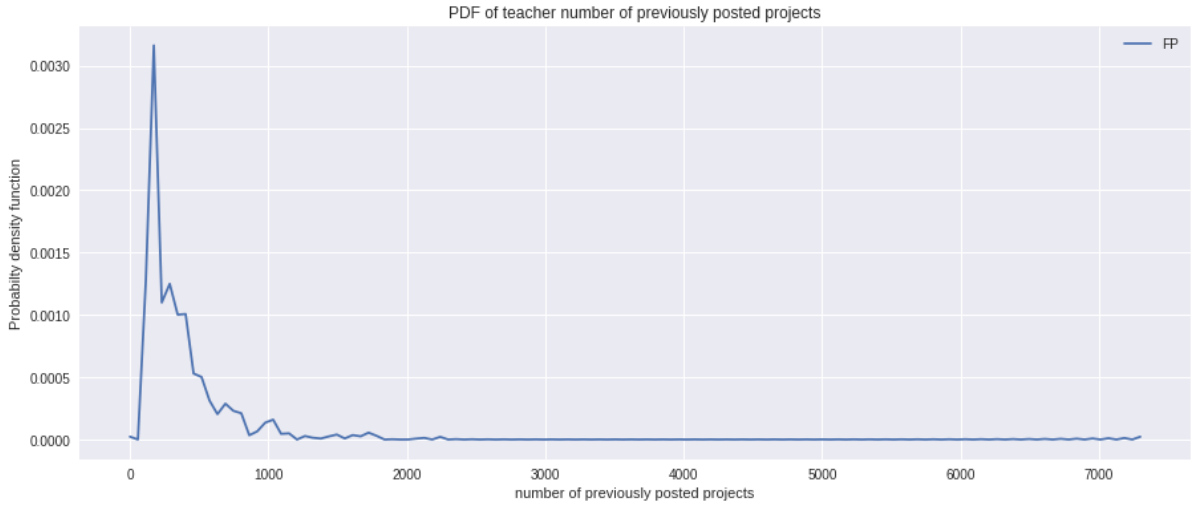
In [0]:
```
p25 = [np.percentile(label, 25) for label in [price_fp, price_tp, price_tn, pr
ice_fn]] # return 25th percentile.
p50 = [np.percentile(label, 50) for label in [price_fp, price_tp, price_tn, pr
ice_fn]] # return 50th percentile.
p75 = [np.percentile(label, 75) for label in [price_fp, price_tp, price_tn, pr
ice_fn]] # return 75th percentile.
per = pd.DataFrame([p25,p50,p75],columns=['FP','TP','TN','FN'],index=['25th Pe
rcentile','50th Percentile','75th Percentile'])
per
```

Out[0]:

	FP	TP	TN	FN
25th Percentile	93.1125	64.9825	180.70	152.945
50th Percentile	178.9950	157.3200	308.05	268.660
75th Percentile	359.9725	299.7350	527.71	429.485

Plotting the pdf with the teacher number of previously posted projects of false positive data points

In [0]:
```
plt.figure(figsize=(15,6))
ax = sns.kdeplot(price_fp, label="FP", bw=0.5)
#ax = sns.kdeplot(price_tp, label="TP", bw=0.5)
#ax = sns.kdeplot(price_fn, label="FN", bw=0.5)
#ax = sns.kdeplot(price_tn, label="TN", bw=0.5)

ax.set_title("PDF of teacher number of previously posted projects")
ax.set_ylabel('Probabilty density function')
ax.set_xlabel('number of previously posted projects')
plt.legend()
plt.show()
```



Observation: For FP, 150-200 no of previously projects submitted has more occurence.

Summary

In [0]:
```
from prettytable import PrettyTable
import sys
sys.stdout.write("\033[1;30m")

x = PrettyTable()
x.field_names = ["Vectorizer", "Optimal Depth","Min sample Split", "AUC"]

x.add_row(["BOW", 50, 500, 0.6623])
x.add_row(["TFIDF", 10, 500, 0.6598])
x.add_row(["AVGW2V", 10, 500, 0.6255])
x.add_row(["TFIDFW2V", 10, 500, 0.6365])
x.add_row(["Task2", 10, 500, 0.5882])
print(x)
```

```
+------------+---------------+------------------+--------+
| Vectorizer | Optimal Depth | Min sample Split |  AUC   |
+------------+---------------+------------------+--------+
|    BOW     |       50      |       500        | 0.6623 |
|   TFIDF    |       10      |       500        | 0.6598 |
|   AVGW2V   |       10      |       500        | 0.6255 |
|  TFIDFW2V  |       10      |       500        | 0.6365 |
|   Task2    |       10      |       500        | 0.5882 |
+------------+---------------+------------------+--------+
```