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
```

```
[Errno 2] No such file or directory: 'drive/My Drive'
/content/drive/My Drive/Assignments_DonorsChoose_2018
```

In [0]:

```
%cd Assignments_DonorsChoose_2018
```

```
[Errno 2] No such file or directory: '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://stackoverflow.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & 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 ''(empty) ex:"Math & Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

project_data['clean_categories'] = cat_list
project_data.drop(['project_subject_categories'], axis=1, inplace=True)

from collections import Counter
my_counter = Counter()
for word in project_data['clean_categories'].values:
    my_counter.update(word.split())

cat_dict = dict(my_counter)
sorted_cat_dict = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))
```

1.3 preprocessing of `project_subject_subcategories`

In [0]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python:
```

```
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039

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

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

project_data['clean_subcategories'] = sub_cat_list
project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
for word in project_data['clean_subcategories'].values:
    my_counter.update(word.split())

sub_cat_dict = dict(my_counter)
sorted_sub_cat_dict = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))
```

1.3 Text preprocessing

In [0]:

```
# merge two column text dataframe:
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(str) + \
                        project_data["project_essay_4"].map(str)
```

In [0]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [0]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [0]:

```
def find_num(text):
    if re.findall(r'\d+', text):
        return 1
    return 0

project_data['numerical_digits'] = project_data['project_resource_summary'].apply(lambda x: find_nu
m(x))
```

In [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[:50000]
```

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
=0.25)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, stratify=y_train, test_size=0.2
5)
```

In [0]:

```
X_train.shape, X_val.shape, X_test.shape
```

Out[0]:

```
((61452, 20), (20484, 20), (27312, 20))
```

In [0]:

```
u=[[],[],[],[],[],[]]
def prob_encode(df):
  for i,col in enumerate(['teacher_prefix', 'school_state', 'clean_categories',
'clean_subcategories', 'project_grade_category','numerical_digits']):
    df[col+str(0)] = 0.5
    df[col+str(1)] = 0.05
    #print(len(u[i]),i)
    if len(u[i])==0:
      u[i]=df.groupby(col).project_is_approved.apply(lambda g: g.value_counts()/len(g)).unstack().s
tack()
    ind=df.columns.get_loc(col)
    for row in df.itertuples():
      try:
        if u[i][row[ind+1]].any():
          df.at[row.Index, col+'0'] = list(u[i][row[ind+1]])[0]
          df.at[row.Index, col+'1'] = list(u[i][row[ind+1]])[1]
      except:
        df.at[row.Index, col+'0'] = 0.5
        df.at[row.Index, col+'1'] = 0.05
    i=i+1
```

In [0]:

```
X_train['project_is_approved'] = y_train
```

In [0]:

```
import time
start = time.time()
prob_encode(X_train)
prob_encode(X_test)
prob_encode(X_val)
X_train.drop(X_train.columns[len(X_train.columns)-1], axis=1, inplace=True)
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: 13.099024506409963
execution time in hours: 0
```

In [0]:

```
from sklearn.preprocessing import StandardScaler
def standardize_data(df_tr,df_cv,df_te,column_name):
  standardized_vec = StandardScaler(with_mean=False)
  # here it will learn mu and sigma
  standardized_vec.fit(df_tr[column_name].values.reshape(-1,1))

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

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

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

In [0]:

```
from sklearn.preprocessing import Normalizer
def normalize_data(df,column_data):
  normalizer = 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]:

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

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

In [0]:

```
from tqdm import tqdm
def textpreprocessed(df,column_name):
  # Combining all the above stundents
  global preprocessed_list

  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]:

```
price_standardized_tr, price_standardized_val, price_standardized_te =
standardize_data(X_train,X_val, X_test,'price')
print()
project_standardized_tr, project_standardized_val, project_standardized_te =
standardize_data(X_train,X_val, X_test,'teacher_number_of_previously_posted_projects')
```

```
(61452, 1)
(20484, 1)
(27312, 1)

(61452, 1)
(20484, 1)
(27312, 1)
```

In [0]:

```
X_train.columns
```

Out[0]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', 'project_title', 'project_essay_1',
       'project_essay_2', 'project_essay_3', 'project_essay_4',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'price', 'quantity',
       'clean_categories', 'clean_subcategories', 'essay', 'numerical_digits',
       'project_is_approved', 'teacher_prefix0', 'teacher_prefix1',
       'school_state0', 'school_state1', 'clean_categories0',
       'clean_categories1', 'clean_subcategories0', 'clean_subcategories1',
       'project_grade_category0', 'project_grade_category1',
       'numerical_digits0'],
      dtype='object')
```

In [0]:

```
X_train.drop(['project_is_approved'], inplace=True, axis=1, errors='ignore')
```

In [0]:

```
X_train.drop(['teacher_prefix', 'school_state', 'clean_categories', 'clean_subcategories',
'project_grade_category', 'numerical_digits','numerical_digits0'], inplace=True, axis=1, errors='ig
nore')
X_test.drop(['teacher_prefix', 'school_state', 'clean_categories', 'clean_subcategories',
'project_grade_category', 'numerical_digits','numerical_digits0'], inplace=True, axis=1, errors='ig
nore')
X_val.drop(['teacher_prefix', 'school_state', 'clean_categories', 'clean_subcategories',
'project_grade_category', 'numerical_digits','numerical_digits0'], inplace=True, axis=1, errors='ig
nore')
```

In [0]:

```
xtr = X_train[['teacher_prefix0', 'teacher_prefix1', 'school_state0', 'school_state1',
'clean_categories0', 'clean_categories1', 'clean_subcategories0', 'clean_subcategories1',
'project_grade_category0', 'project_grade_category1']].T.reset_index().values
xval = X_val[['teacher_prefix0', 'teacher_prefix1', 'school_state0', 'school_state1',
'clean_categories0', 'clean_categories1', 'clean_subcategories0', 'clean_subcategories1',
'project_grade_category0', 'project_grade_category1']].T.reset_index().values
xte = X_test[['teacher_prefix0', 'teacher_prefix1', 'school_state0', 'school_state1',
'clean_categories0', 'clean_categories1', 'clean_subcategories0', 'clean_subcategories1',
'project_grade_category0', 'project_grade_category1']].T.reset_index().values
```

In [0]:

```
teacher_prefix0_tr, teacher_prefix1_tr, school_state0_tr, school_state1_tr, clean_categories0_tr, c
lean_categories1_tr,\
clean_subcategories0_tr, clean_subcategories1_tr, project_grade_category0_tr,
project_grade_category1_tr = [xtr[:,1:][i].reshape(-1,1) for i in range(10)]

teacher_prefix0_val, teacher_prefix1_val, school_state0_val, school_state1_val,
clean_categories0_val, clean_categories1_val,\
clean_subcategories0_val, clean_subcategories1_val, project_grade_category0_val,
project_grade_category1_val = [xval[:,1:][i].reshape(-1,1) for i in range(10)]

teacher_prefix0_te, teacher_prefix1_te, school_state0_te, school_state1_te, clean_categories0_te, c
lean_categories1_te,\
clean_subcategories0_te, clean_subcategories1_te, project_grade_category0_te,
project_grade_category1_te = [xte[:,1:][i].reshape(-1,1) for i in range(10)]
```

[Task-1] Apply both Random Forrest and GBDT on these feature sets

In [0]:

```
from scipy.sparse import hstack

f_tr = np.hstack([teacher_prefix0_tr, teacher_prefix1_tr, school_state0_tr, school_state1_tr, clean
_categories0_tr, clean_categories1_tr,\
clean_subcategories0_tr, clean_subcategories1_tr, project_grade_category0_tr,
project_grade_category1_tr,price_standardized_tr,project_standardized_tr])

f_cr = np.hstack([teacher_prefix0_val, teacher_prefix1_val, school_state0_val, school_state1_val, c
lean_categories0_val, clean_categories1_val,\
clean_subcategories0_val, clean_subcategories1_val, project_grade_category0_val,
project_grade_category1_val,price_standardized_val,project_standardized_val])
f_te = np.hstack([teacher_prefix0_te, teacher_prefix1_te, school_state0_te, school_state1_te, clean
_categories0_te, clean_categories1_te,\
clean_subcategories0_te, clean_subcategories1_te, project_grade_category0_te,
project_grade_category1_te,price_standardized_te,project_standardized_te])

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

In [0]:

```
feature_list_x = ['teacher_prefix0', 'teacher_prefix1', 'school_state0', 'school_state1',
'clean_categories0', 'clean_categories1', 'clean_subcategories0', 'clean_subcategories1',
'project_grade_category0',
'project_grade_category1']+['price','teacher_number_of_previously_posted_projects']
len(feature_list_x)
```

Out[0]:

```
12
```

In [0]:

```
from sklearn.ensemble import RandomForestClassifier
import time

def optimal_hyp(X_tr,y_train,X_cr,y_val):
  global df1
  depth =  [2,4,6,8,10,12]
  n_estimators  = [5, 10, 50,75,100,200]
  cols = ['depth', 'n_estimator', 'Train_AUC_Score', 'CV_AUC_Score']
  lst = []
  start = time.time()
  for d in depth:
    for n in n_estimators:
      clf =  RandomForestClassifier(max_depth = d, n_estimators = n,class_weight='balanced',rando
m_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_proba(X_cr)[:,1])
      #print(tr_score)
      lst.append([d,n,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]:

```
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
```

```
def optimal_hyp_gbdt(X_tr,y_train,X_cr,y_val):
  global df1
  depth =  [2,4,6,8,10,12]
  n_estimators  = [5, 10, 50,75,100]
  cols = ['depth', 'n_estimator', 'Train_AUC_Score', 'CV_AUC_Score']
  lst = []
  start = time.time()
  for d in depth:
    for n in n_estimators:
        clf =  XGBClassifier(max_depth = d, n_estimators = n,class_weight='balanced',random_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_proba(X_cr)[:,1])
        #print(tr_score)
        lst.append([d,n,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-pyth
on
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 ROC_plot(X_train,X_te,y_train,y_test):
  global model
  model =  RandomForestClassifier(max_depth = int(best_d), n_estimators = int(n_est),class_weight='
balanced', 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("FPR")
  plt.ylabel("TPR")
  plt.title("ROC")
  plt.title("ROC")
  plt.grid()
  plt.show()
  print("="*100)
```

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 ROC_plot_gbdt(X_train,X_te,y_train,y_test):
  global model
  model = XGBClassifier(max_depth = int(best_d), n_estimators = int(n_est),class_weight='balanced', 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("FPR")
  plt.ylabel("TPR")
  plt.title("ROC")
  plt.grid()
  plt.show()
  print("="*100)
```

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('n_estimators',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]:

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

    t = threshould[np.argmax(fpr*(1-tpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions'''
```

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, threshould, fpr, tpr):\n    \n    t =
threshould[np.argmax(fpr*(1-tpr))]\n    \n    # (tpr*(1-fpr)) will be maximum if your fpr is very l
ow 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            predictions.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['n_estimator'] = df1['n_estimator'].astype(str)

  ax = df.plot(xticks=df.index, rot=55,figsize=(15,5))
  ax.set_ylabel("AUC Score")
  ax.set_xlabel("Hyperparameters: (depth, n_estimator)")
  ax.set_xticklabels(df1[['depth', 'n_estimator']].apply(lambda x: ','.join(x), axis=1))
  ax
```

Set 2: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF) + preprocessed_eassay (TFIDF)

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_te
st,'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'), textp
reprocessed(X_test,'project_title'))
tfidf_vec_resource_tr,tfidf_vec_resource_val,tfidf_vec_resource_te,tfidf_vec_resource_list =
tfidf_vec(textpreprocessed(X_train,'project_resource_summary'),textpreprocessed(X_val,'project_reso
urce_summary'), textpreprocessed(X_test,'project_resource_summary'))
```

```
100%|██████████| 61452/61452 [01:07<00:00, 908.08it/s]
100%|██████████| 20484/20484 [00:22<00:00, 913.88it/s]
100%|██████████| 27312/27312 [00:29<00:00, 914.80it/s]
```

```
Shape of matrix after one hot encodig  (61452, 13147)
Shape of matrix after one hot encodig  (20484, 13147)
```

```
  3%|▎         | 2121/61452 [00:00<00:02, 21208.71it/s]
```

```
Shape of matrix after one hot encodig  (27312, 13147)
```

```
100%|██████████| 61452/61452 [00:02<00:00, 20982.64it/s]
100%|██████████| 20484/20484 [00:00<00:00, 21148.85it/s]
100%|██████████| 27312/27312 [00:01<00:00, 21099.06it/s]
```

```
Shape of matrix after one hot encodig  (61452, 2289)
Shape of matrix after one hot encodig  (20484, 2289)
```

```
  1%|▏         | 874/61452 [00:00<00:06, 8735.28it/s]
```

```
Shape of matrix after one hot encodig  (27312, 2289)
```

```
100%|██████████| 61452/61452 [00:07<00:00, 8702.09it/s]
100%|██████████| 20484/20484 [00:02<00:00, 8675.37it/s]
100%|██████████| 27312/27312 [00:03<00:00, 8649.86it/s]
```

```
Shape of matrix after one hot encodig  (61452, 4381)
Shape of matrix after one hot encodig  (20484, 4381)
Shape of matrix after one hot encodig  (27312, 4381)
```

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))
```

```
19829
```

In [0]:

```
X_tr, X_cr, X_te = hstack_data(tfidf_vec_titles_tr, tfidf_vec_titles_val, tfidf_vec_titles_te, \
                   tfidf_vec_essay_tr, tfidf_vec_essay_val, tfidf_vec_essay_te,\
                   tfidf_vec_resource_tr, tfidf_vec_resource_val, tfidf_vec_resource_te)
```

In [0]:

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

Out[0]:

```
((61452, 19829), (20484, 19829), (27312, 19829))
```

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")
  ax.set_xlabel("Hyperparameters: (depth, min_samples_split)")
  ax.set_xticklabels(df1[['depth', 'min_samples_split']].apply(lambda x: ','.join(x), axis=1))
  ax"""
```

2.1 Applying Random Forests on TFIDF, SET 2

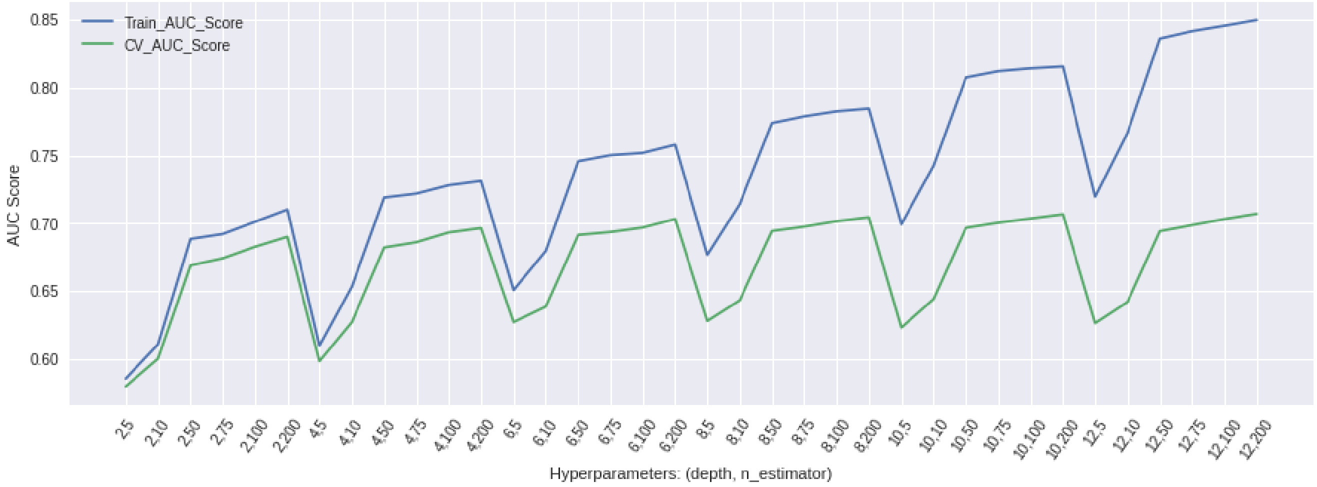
In [0]:

```
optimal_hyp(X_tr,y_train,X_cr,y_val)
```

```
execution time in minutes: 6.646934680143992
execution time in hours: 0
```

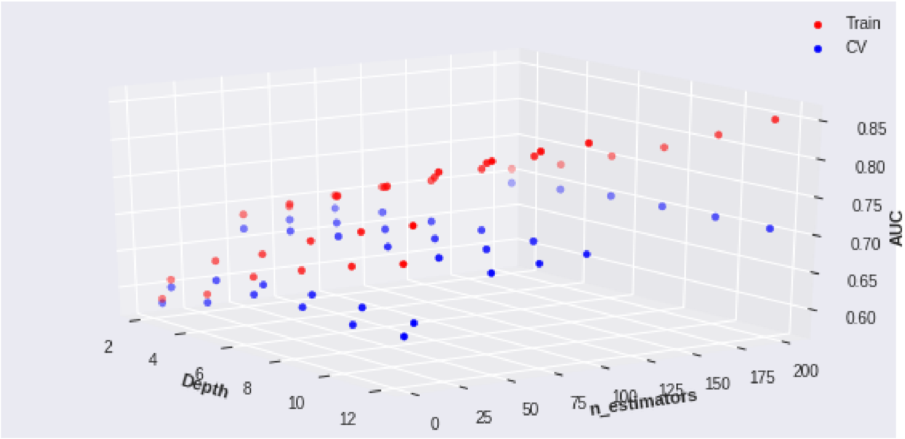
In [0]:

```
plot_auc(df1)
```



In [0]:

```
x = df1['depth'].astype(float).values
y = df1['n_estimator'].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,n_est,tr_auc_score,cv_auc_score = df1.iloc[df1.CV_AUC_Score.argmax()]
print("optimal depth:",best_d,"\nn_estimator:",n_est,"\nCV AUC score:", cv_auc_score)
```

```
optimal depth: 12
n_estimator: 200
CV AUC score: 0.7069679973821456
```

In [0]:

```
model = RandomForestClassifier(max_depth = int(best_d), n_estimators =
int(n_est),class_weight='balanced', random_state = 0)
model.fit(X_tr, y_train)
```

Out[0]:

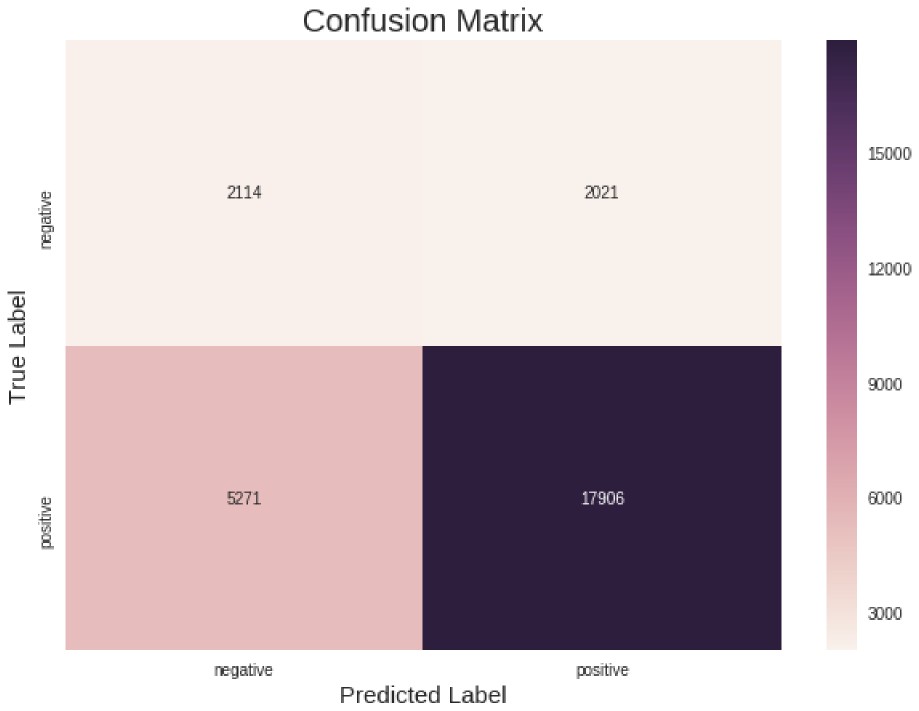
```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=12, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=200, n_jobs=None, oob_score=False, random_state=0,
            verbose=0, warm_start=False)
```

In [0]:

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

In [0]:

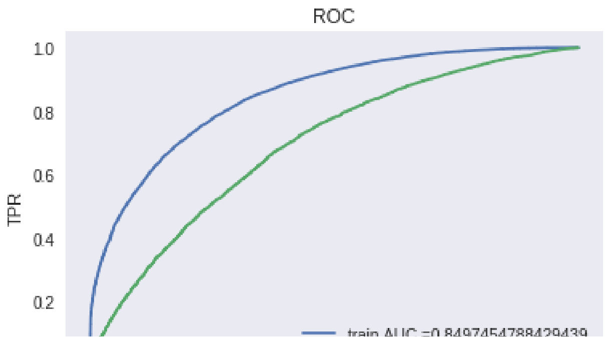
```
labels = ["negative","positive"]
get_confusion_matrix_values(np.array(y_test), y_pred_te)
```



```
True Positives : 17906
False Positives : 2021
True Negatives : 2114
False Negatives : 5271
```

In [0]:

```
start = time.time()
ROC_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: 0.8037584066390991
execution time in hours: 0
```

In [0]:

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

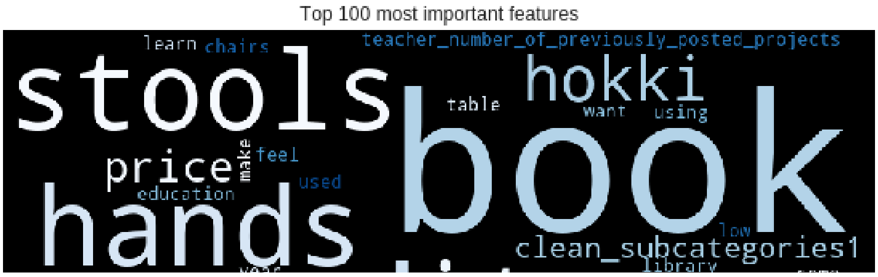
Out[0]:

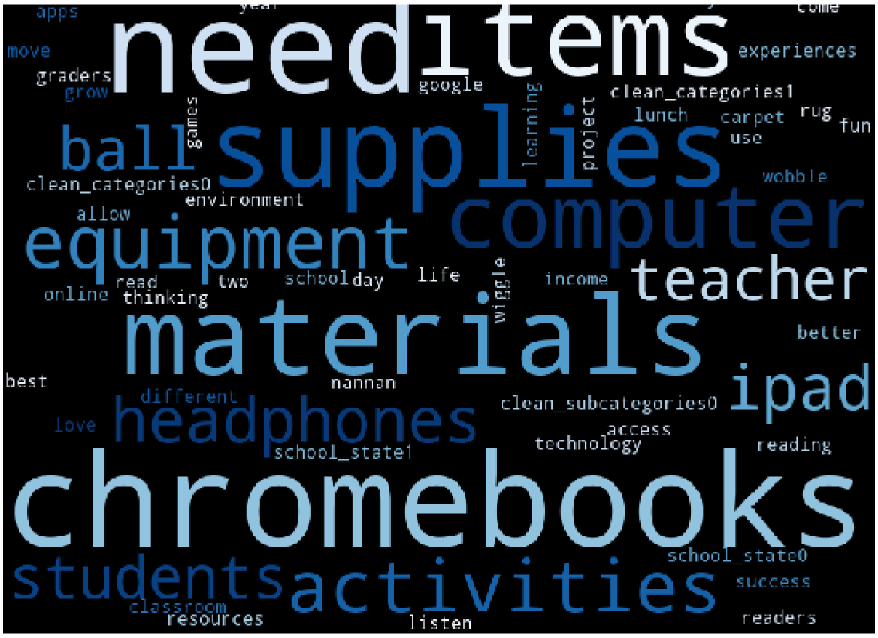
```
0.6960855785238034
```

Top 100 Imp. features

In [0]:

```python
#https://www.geeksforgeeks.org/generating-word-cloud-python/
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import pandas as pd
coef = model.feature_importances_
coef_df = pd.DataFrame({'word': feature_names, 'coeficient': coef})
df = coef_df.sort_values("coeficient", ascending = False)[:100]
#print(df)
# iterate through the csv file
words_str = ' '
stopwords = set(STOPWORDS)
for val in df.word:
    #print(val)
    # typecaste each val to string
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
      words_str = words_str + words + ' '

wordcloud = WordCloud(width = 800, height = 800,
                colormap="Blues",
                stopwords = stopwords,
                min_font_size = 10).generate(words_str)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.title("Top 100 most important features")
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```





2.2 Applying GBDT on TFIDF, SET 2

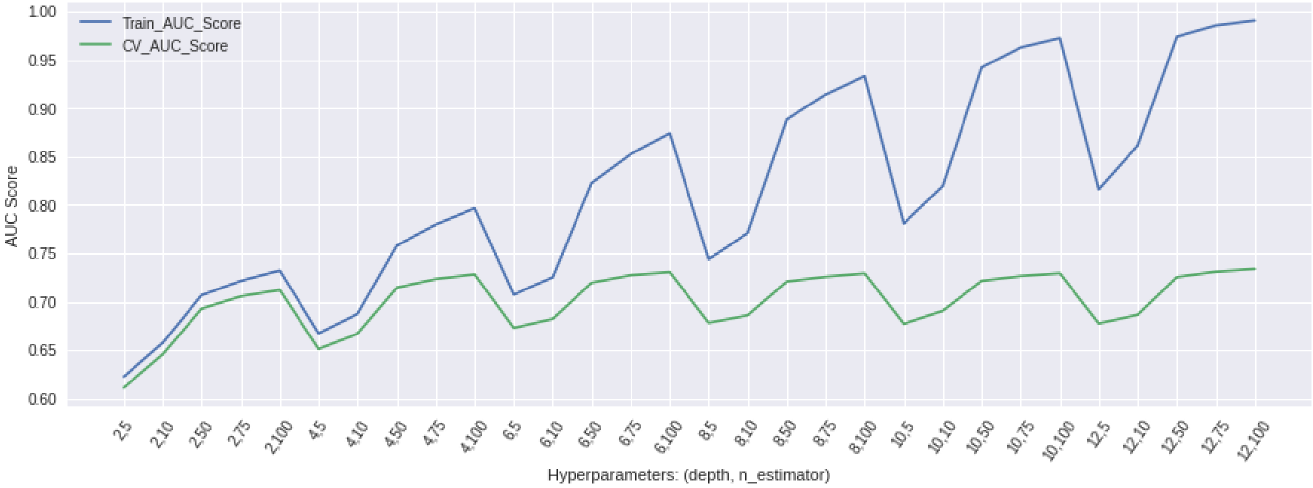
In [0]:

```
optimal_hyp_gbdt(X_tr,y_train,X_cr,y_val)
```

```
execution time in minutes: 73.79893393913905
execution time in hours: 1
```

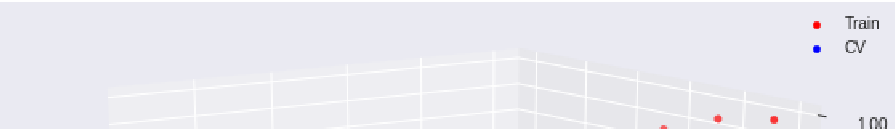
In [0]:

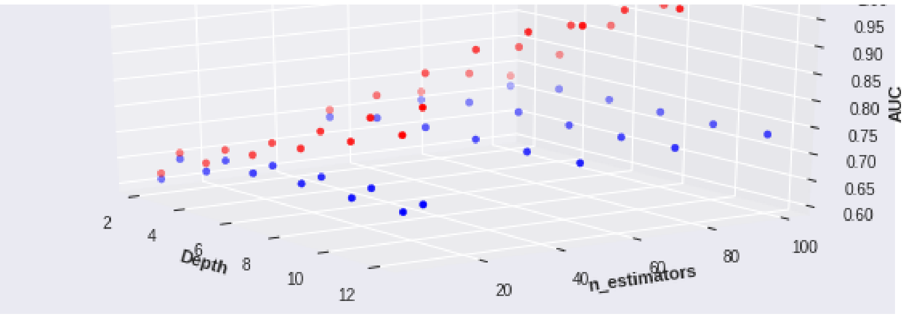
```
plot_auc(df1)
```



In [0]:

```
x = df1['depth'].astype(float).values
y = df1['n_estimator'].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,n_est,tr_auc_score,cv_auc_score = df1.iloc[df1.CV_AUC_Score.argmax()]
print("optimal depth:",best_d,"\nn_estimator:",n_est,"\nCV AUC score:", cv_auc_score)
```

```
optimal depth: 12
n_estimator: 100
CV AUC score: 0.7339309134352062
```

In [0]:

```
model = XGBClassifier(max_depth = int(best_d), n_estimators = int(n_est),class_weight='balanced', r
andom_state = 0)
model.fit(X_tr, y_train)
```

Out[0]:

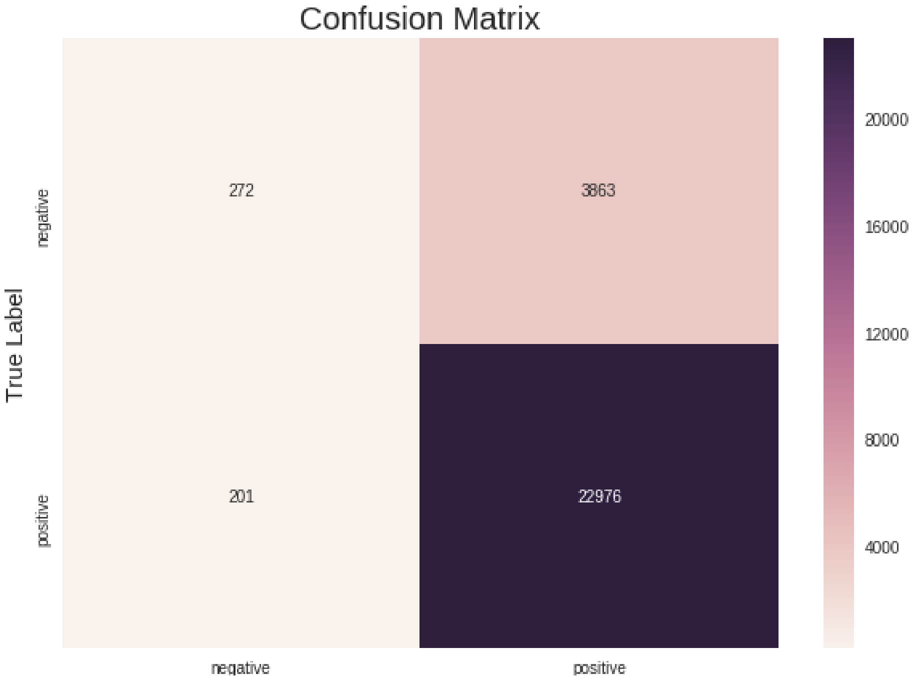
```
XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced',
       colsample_bylevel=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=12, min_child_weight=1, missing=None,
       n_estimators=100, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,
       subsample=1)
```

In [0]:

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

In [0]:

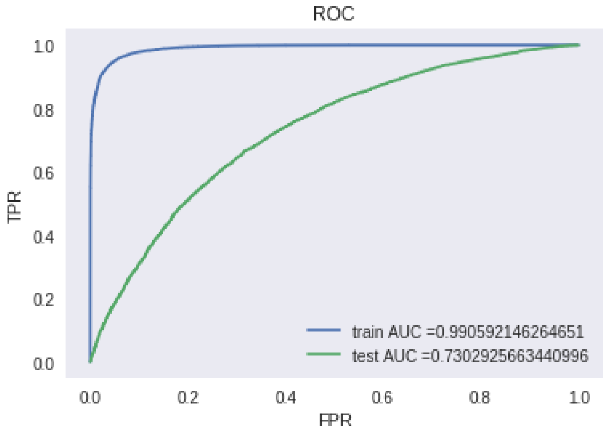
```
labels = ["negative","positive"]
get_confusion_matrix_values(np.array(y_test), y_pred_te)
```



```
True Positives : 22976
False Positives : 3863
True Negatives : 272
False Negatives : 201
```

In [0]:

```
start = time.time()
ROC_plot_gbdt(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: 4.625129687786102
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)
```

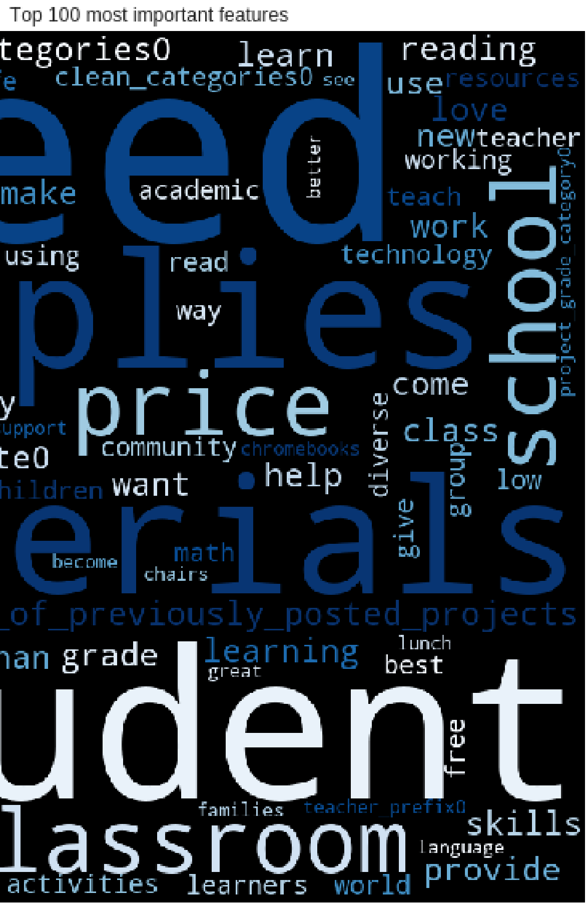
Out[0]:

```
0.7302925663440996
```

Top 100 Imp. features

In [0]:

```
coef = model.feature_importances_
coef_df = pd.DataFrame({'word': feature_names, 'coeficient': coef})
df = coef_df.sort_values("coeficient", ascending = False)[:100]
#print(df)
# iterate through the csv file
words_str = ' '
stopwords = set(STOPWORDS)
for val in df.word:
    #print(val)
    # typecaste each val to string
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()
    for words in tokens:
      words_str = words_str + words + ' '

wordcloud = WordCloud(width = 800, height = 800,
                colormap="Blues",
                stopwords = stopwords,
                min_font_size = 10).generate(words_str)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.title("Top 100 most important features")
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```



Top 100 most important features

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_vec(textpreprocessed(
X_train,'essay'),textpreprocessed(X_val,'essay'), textpreprocessed(X_test,'essay'))
```

```
bow_vec_titles_tr,bow_vec_titles_val,bow_vec_titles_te,bow_vec_titles_list =
bow_vec(textpreprocessed(X_train,'project_title'),textpreprocessed(X_val,'project_title'),
textpreprocessed(X_test,'project_title'))
bow_vec_resource_tr,bow_vec_resource_val,bow_vec_resource_te,bow_vec_resource_list =
bow_vec(textpreprocessed(X_train,'project_resource_summary'),textpreprocessed(X_val,'project_resour
ce_summary'), textpreprocessed(X_test,'project_resource_summary'))
```

```
100%|██████████| 61452/61452 [00:12<00:00, 5087.55it/s]
100%|██████████| 20484/20484 [00:03<00:00, 5152.12it/s]
100%|██████████| 27312/27312 [00:05<00:00, 5222.13it/s]
```

```
Shape of matrix after one hot encodig  (61452, 13145)
Shape of matrix after one hot encodig  (20484, 13145)
```

```
  8%|▊         | 4674/61452 [00:00<00:01, 46731.47it/s]
```

```
Shape of matrix after one hot encodig  (27312, 13145)
```

```
100%|██████████| 61452/61452 [00:01<00:00, 50471.19it/s]
100%|██████████| 20484/20484 [00:00<00:00, 49885.02it/s]
100%|██████████| 27312/27312 [00:00<00:00, 52421.17it/s]
```

```
Shape of matrix after one hot encodig  (61452, 2284)
Shape of matrix after one hot encodig  (20484, 2284)
```

```
  4%|▍         | 2655/61452 [00:00<00:02, 26543.95it/s]
```

```
Shape of matrix after one hot encodig  (27312, 2284)
```

```
100%|██████████| 61452/61452 [00:01<00:00, 32320.61it/s]
100%|██████████| 20484/20484 [00:00<00:00, 33352.42it/s]
100%|██████████| 27312/27312 [00:00<00:00, 33205.33it/s]
```

```
Shape of matrix after one hot encodig  (61452, 4371)
Shape of matrix after one hot encodig  (20484, 4371)
Shape of matrix after one hot encodig  (27312, 4371)
```

In [0]:

```
feature_list = feature_list_x.copy()
feature_names = [*feature_list , *bow_vec_titles_list, *bow_vec_essay_list, *bow_vec_resource_list]
print(len(feature_names))
```

```
19812
```

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]:

```
((61452, 19812), (20484, 19812), (27312, 19812))
```

1.1 Applying Random Forest on BOW, SET 1

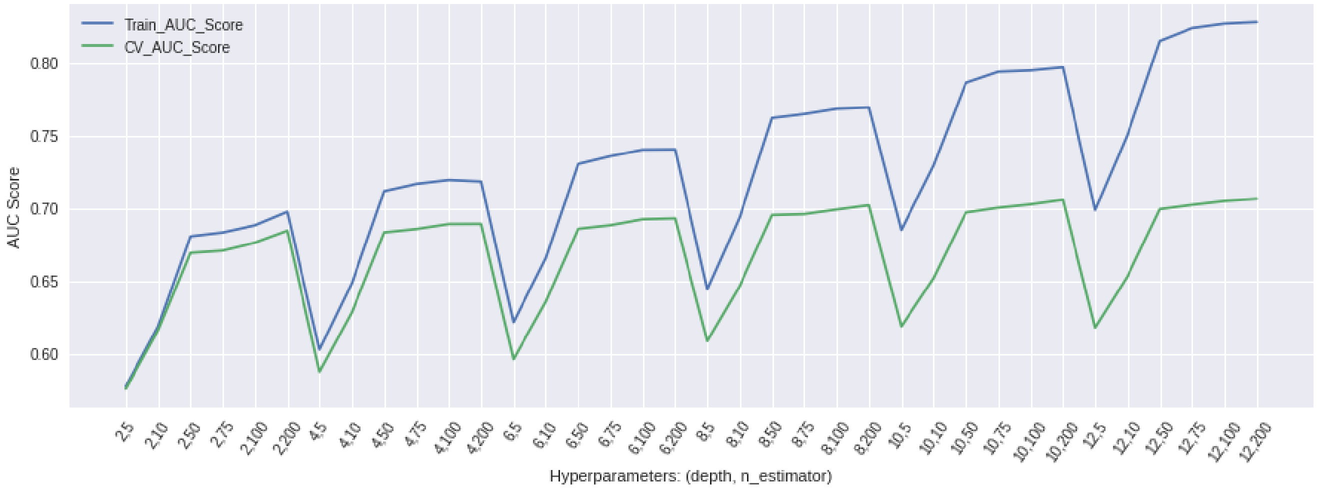
In [0]:

```
optimal_hyp(X_tr,y_train,X_cr,y_val)
```

```
execution time in minutes: 4.3013646880785625
execution time in hours: 0
```

In [0]:

```
plot_auc(df1)
```



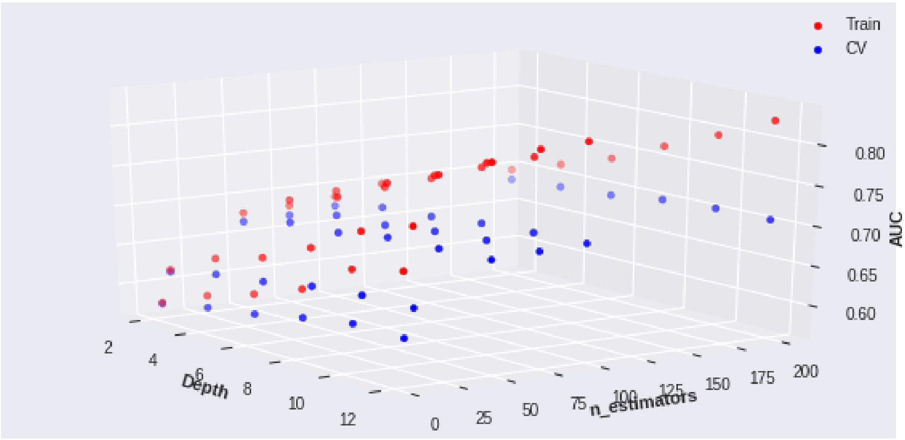
In [0]:

```
best_d,n_est,tr_auc_score,cv_auc_score = df1.iloc[df1.CV_AUC_Score.argmax()]
print("optimal depth:",best_d,"\nn_estimator:",n_est,"\nCV AUC score:", cv_auc_score)
```

```
optimal depth: 12
n_estimator: 200
CV AUC score: 0.7065557583784436
```

In [0]:

```
x = df1['depth'].astype(float).values
y = df1['n_estimator'].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]:

```
model = RandomForestClassifier(max_depth = int(best_d), n_estimators =
int(n_est),class_weight='balanced', random_state = 0)
model.fit(X_tr, y_train)
```

Out[0]:

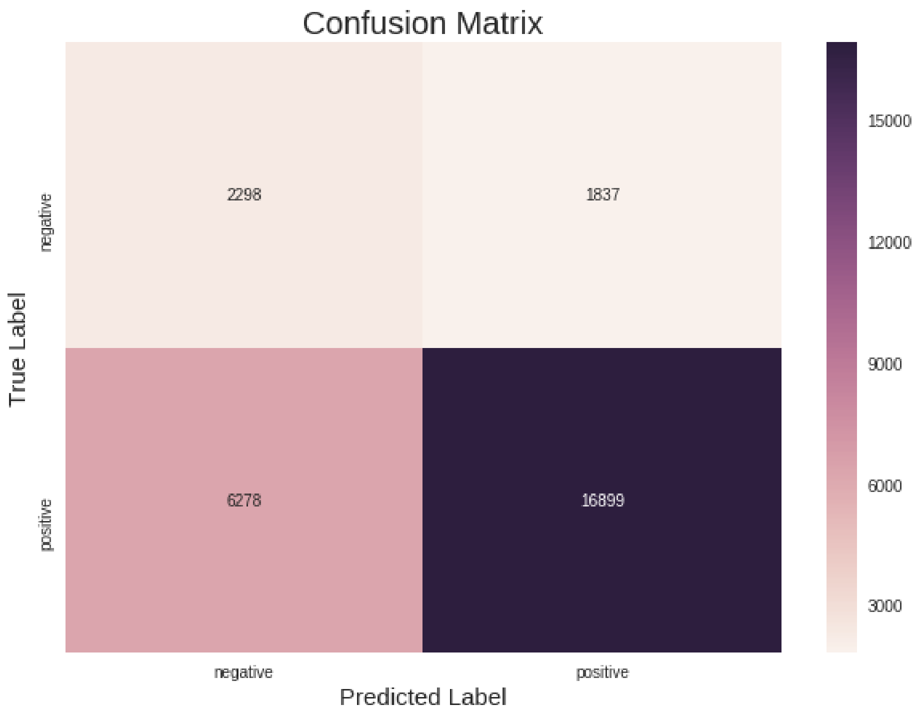
```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=12, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=200, n_jobs=None, oob_score=False, random_state=0,
            verbose=0, warm_start=False)
```

In [0]:

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

In [0]:

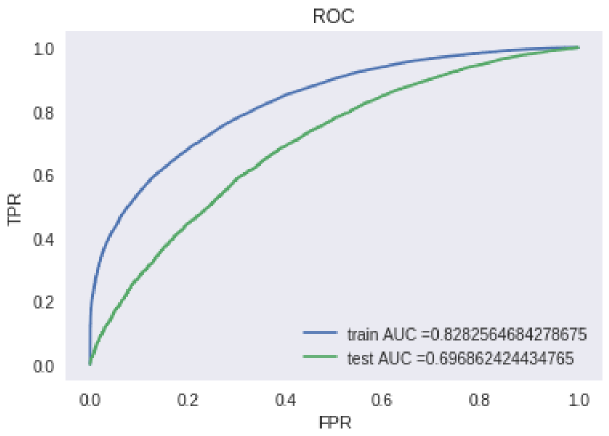
```
labels = ["negative","positive"]
get_confusion_matrix_values(np.array(y_test), y_pred_te)
```



```
True Positives : 16899
False Positives : 1837
True Negatives : 2298
False Negatives : 6278
```

In [0]:

```
start = time.time()
ROC_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: 0.6085020899772644
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)
```

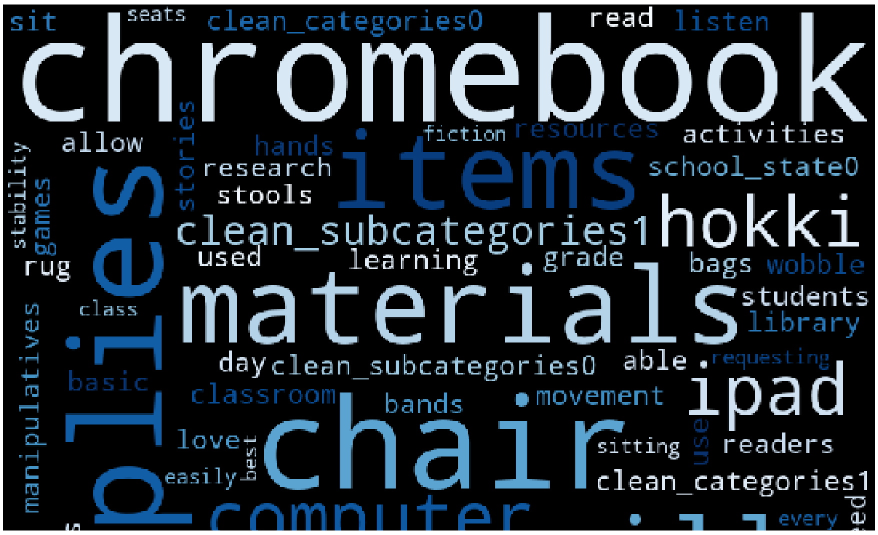
Out[0]:

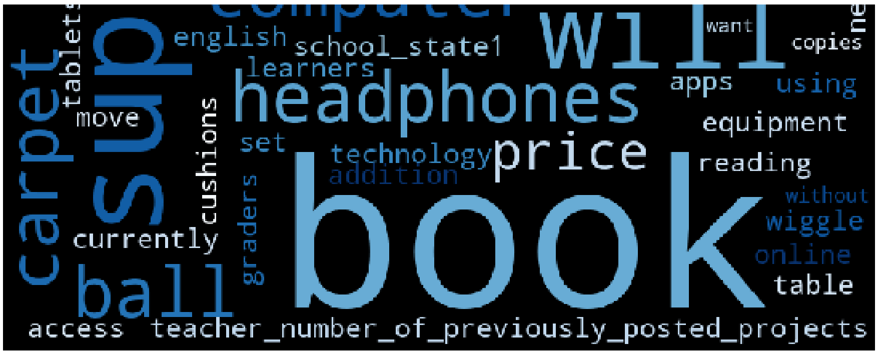
```
0.696862424434765
```

In [0]:

```
coef = model.feature_importances_
coef_df = pd.DataFrame({'word': feature_names, 'coeficient': coef})
df = coef_df.sort_values("coeficient", ascending = False)[:100]
#print(df)
# iterate through the csv file
words_str = ' '
stopwords = set(STOPWORDS)
for val in df.word:
    #print(val)
    # typecaste each val to string
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
      words_str = words_str + words + ' '

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='black',
                colormap="Blues",
                stopwords = stopwords,
                min_font_size = 10).generate(words_str)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.title("Top 100 most important features")
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```

Top 100 most important features





1.2 Applying GBDT on BOW, SET 1

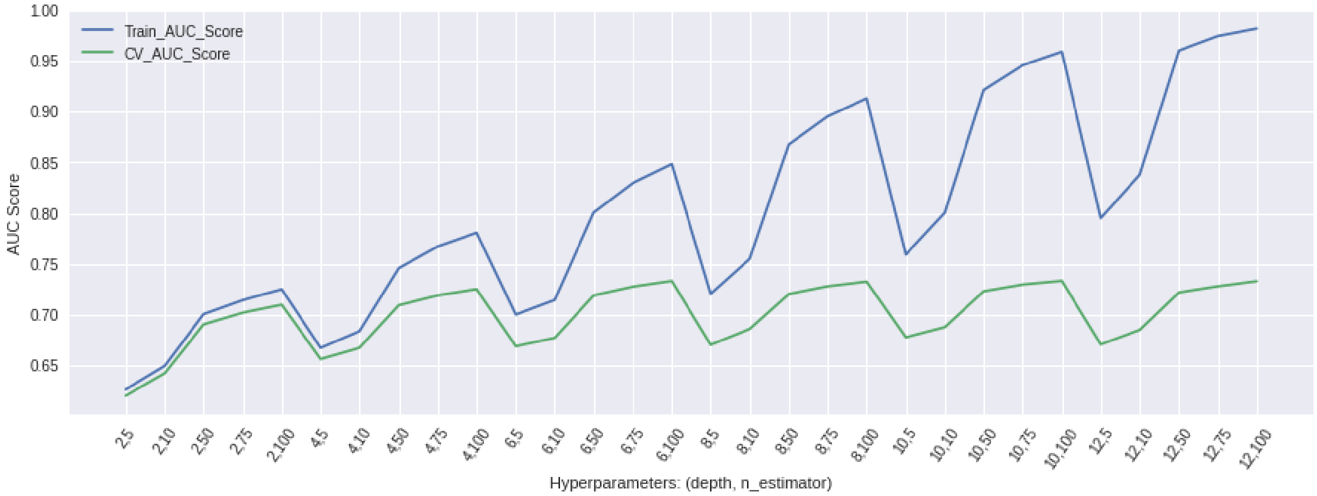
In [0]:

```
optimal_hyp_gbdt(X_tr,y_train,X_cr,y_val)
```

```
execution time in minutes: 31.82730416059494
execution time in hours: 0
```

In [0]:

```
plot_auc(df1)
```



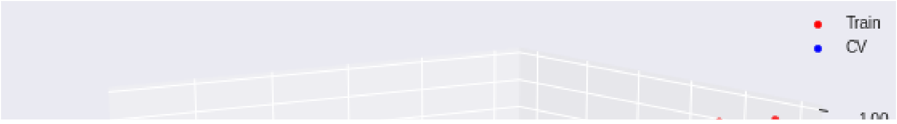
In [0]:

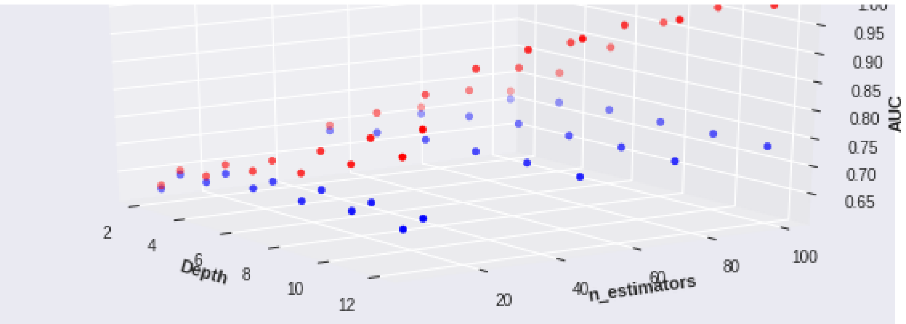
```
best_d,n_est,tr_auc_score,cv_auc_score = df1.iloc[df1.CV_AUC_Score.argmax()]
print("optimal depth:",best_d,"\nn_estimator:",n_est,"\nCV AUC score:", cv_auc_score)
```

```
optimal depth: 10
n_estimator: 100
CV AUC score: 0.7331131121139494
```

In [0]:

```
x = df1['depth'].astype(float).values
y = df1['n_estimator'].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]:

```
model = XGBClassifier(max_depth = int(best_d), n_estimators = int(n_est),class_weight='balanced', r
andom_state = 0)
model.fit(X_tr, y_train)
```

Out[0]:

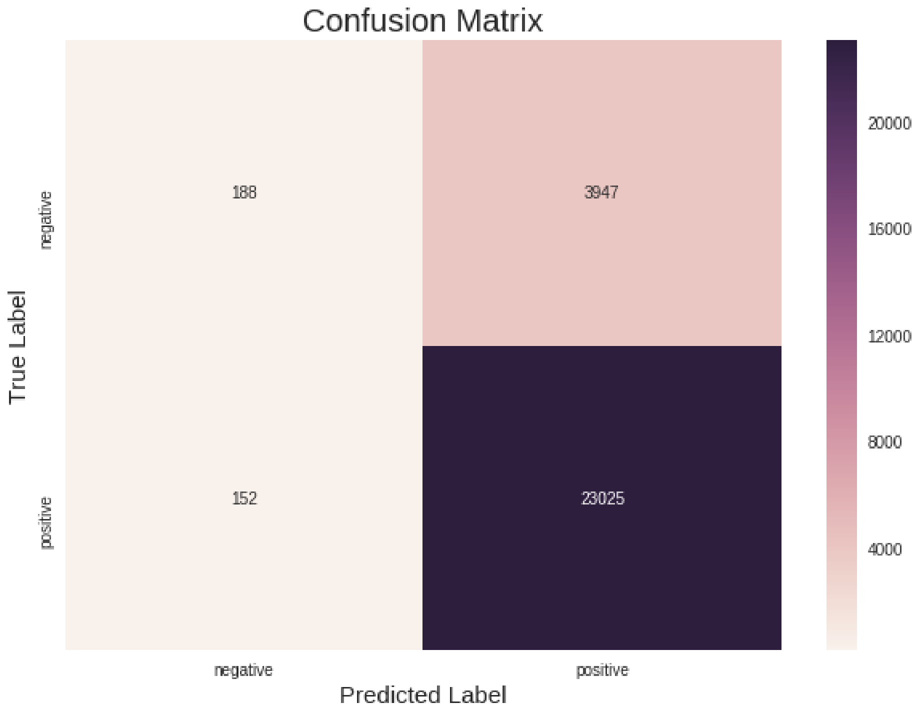
```
XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced',
       colsample_bylevel=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=10, min_child_weight=1, missing=None,
       n_estimators=100, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,
       subsample=1)
```

In [0]:

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

In [0]:

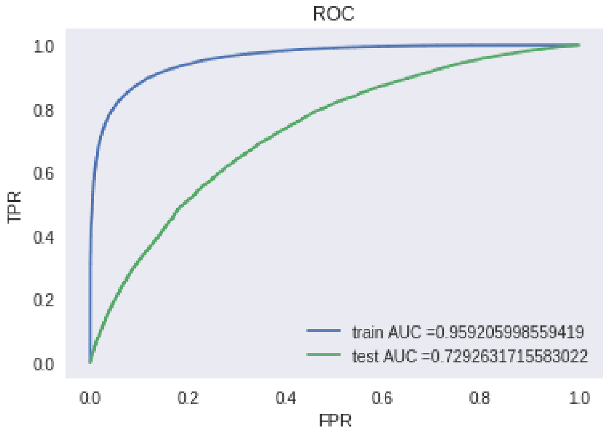
```
labels = ["negative","positive"]
get_confusion_matrix_values(np.array(y_test), y_pred_te)
```



```
True Positives : 23025
False Positives : 3947
True Negatives : 188
False Negatives : 152
```

In [0]:

```
start = time.time()
ROC_plot_gbdt(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: 2.050114325682322
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]:

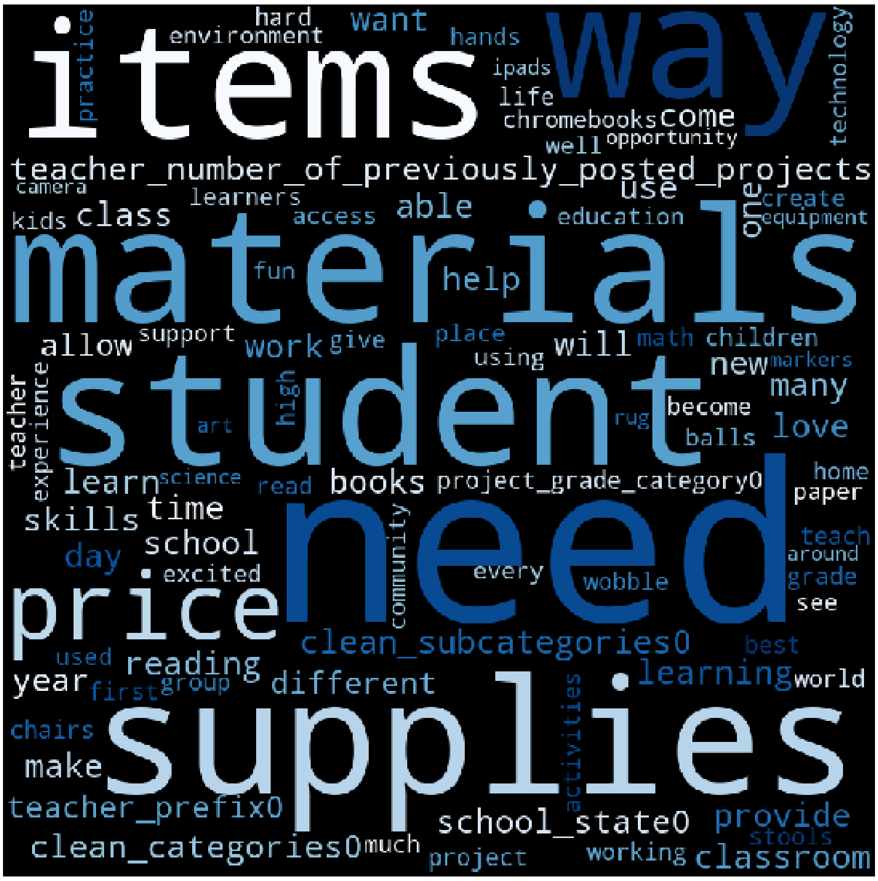
```
0.7292631715583022
```

Top 100 Imp. features

In [0]:

```
coef = model.feature_importances_
coef_df = pd.DataFrame({'word': feature_names, 'coeficient': coef})
df = coef_df.sort_values("coeficient", ascending = False)[:100]
#print(df)
# iterate through the csv file
words_str = ' '
stopwords = set(STOPWORDS)
for val in df.word:
    #print(val)
    # typecaste each val to string
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
      words_str = words_str + words + ' '

wordcloud = WordCloud(width = 800, height = 800,
                colormap="Blues",
                stopwords = stopwords,
                min_font_size = 10).generate(words_str)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.title("Top 100 most important features")
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```

Top 100 most important features



Set 3: categorical(instead of one hot encoding, try response coding: use probability values), 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-use-pickle-to-sa
ve-and-load-variables-in-python/
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [0]:

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

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

In [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%|██████████| 61452/61452 [00:11<00:00, 5138.61it/s]
100%|██████████| 61452/61452 [00:15<00:00, 3840.77it/s]
  2%|▏         | 518/27312 [00:00<00:05, 5172.31it/s]
```

```
61452
30
```

```
100%|██████████| 27312/27312 [00:05<00:00, 5196.33it/s]
100%|██████████| 27312/27312 [00:07<00:00, 3897.92it/s]
  2%|▏         | 506/20484 [00:00<00:03, 5056.32it/s]
```

```
27312
30
```

```
100%|██████████| 20484/20484 [00:03<00:00, 5173.12it/s]
100%|██████████| 20484/20484 [00:05<00:00, 3822.52it/s]
```

```
20484
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%|██████████| 61452/61452 [00:01<00:00, 53112.16it/s]
100%|██████████| 61452/61452 [00:00<00:00, 84982.59it/s]
 19%|█▉        | 5283/27312 [00:00<00:00, 52822.37it/s]
```

```
61452
30
```

```
100%|██████████| 27312/27312 [00:00<00:00, 52867.62it/s]
100%|██████████| 27312/27312 [00:00<00:00, 86352.40it/s]
 25%|██▌       | 5164/20484 [00:00<00:00, 51639.19it/s]
```

```
27312
30
```

```
100%|██████████| 20484/20484 [00:00<00:00, 52039.88it/s]
100%|██████████| 20484/20484 [00:00<00:00, 86828.78it/s]
```

```
20484
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%|██████████| 61452/61452 [00:01<00:00, 33774.37it/s]
100%|██████████| 61452/61452 [00:01<00:00, 35939.94it/s]
 12%|█         | 3373/27312 [00:00<00:00, 33725.13it/s]
```

```
61452
30
```

```
100%|██████████| 27312/27312 [00:00<00:00, 34562.10it/s]
100%|██████████| 27312/27312 [00:00<00:00, 35966.30it/s]
 16%|█▌        | 3307/20484 [00:00<00:00, 33061.20it/s]
```

```
27312
30
```

```
100%|██████████| 20484/20484 [00:00<00:00, 33381.78it/s]
100%|██████████| 20484/20484 [00:00<00:00, 35538.23it/s]
```

```
20484
30
```

In [0]:

```
from scipy.sparse import coo_matrix
X_tr, X_cr, X_te = hstack_data(coo_matrix(np.asarray(avgw2v_vec_titles_tr)), coo_matrix(np.asarray
(avgw2v_vec_titles_val)), coo_matrix(np.asarray(avgw2v_vec_titles_te)), \
                   coo_matrix(np.asarray(avgw2v_vec_essay_tr)),
coo_matrix(np.asarray(avgw2v_vec_essay_val)), coo_matrix(np.asarray(avgw2v_vec_essay_te)),\
                   coo_matrix(np.asarray(avgw2v_vec_resource_tr)),
coo_matrix(np.asarray(avgw2v_vec_resource_val)), coo_matrix(np.asarray(avgw2v_vec_resource_te)))
```

In [0]:

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

Out[0]:

```
((61452, 102), (20484, 102), (27312, 102))
```

3.1 Applying Random Forest on AVG W2V, SET 3

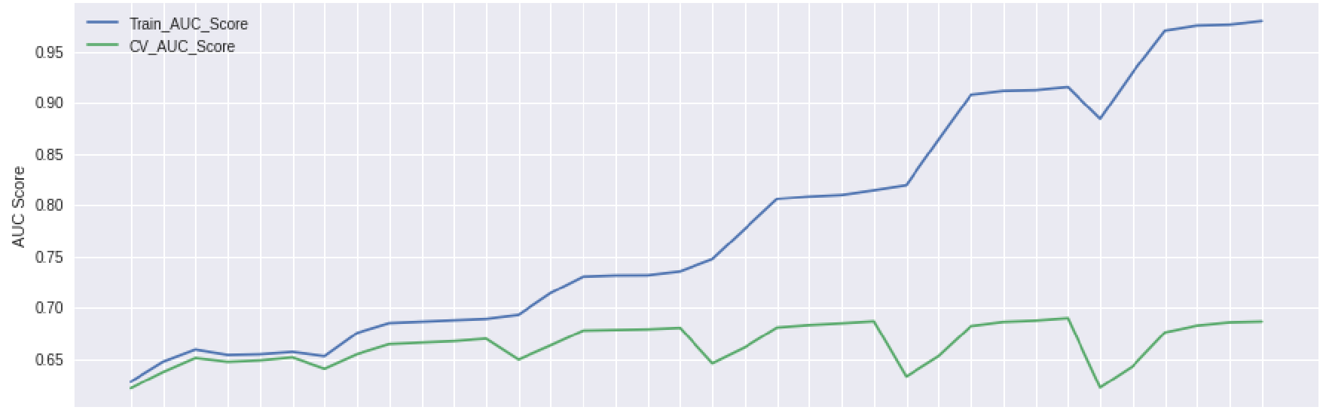
In [0]:

```
optimal_hyp(X_tr,y_train,X_cr,y_val)
```

```
execution time in minutes: 38.61274026632309
execution time in hours: 0
```

In [0]:

```
plot_auc(df1)
```



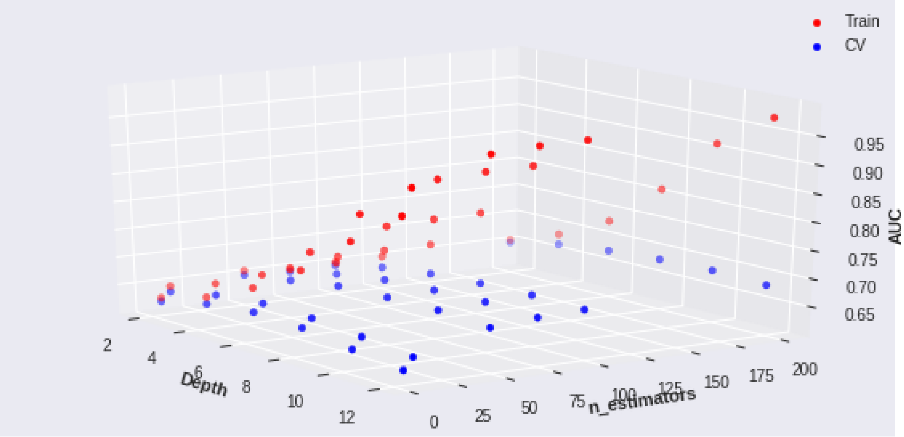
In [0]:

```
best_d,n_est,tr_auc_score,cv_auc_score = df1.iloc[df1.CV_AUC_Score.argmax()]
print("optimal depth:",best_d,"\nn_estimator:",n_est,"\nCV AUC score:", cv_auc_score)
```

```
optimal depth: 10
n_estimator: 200
CV AUC score: 0.6897127326111088
```

In [0]:

```
x = df1['depth'].astype(float).values
y = df1['n_estimator'].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]:

```
model = RandomForestClassifier(max_depth = int(best_d), n_estimators =
int(n_est),class_weight='balanced', random_state = 0)
model.fit(X_tr, y_train)
```

Out[0]:

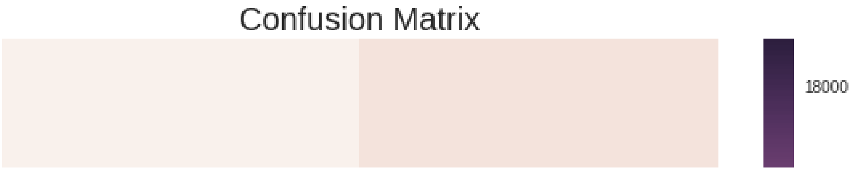
```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=10, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=200, n_jobs=None, oob_score=False, random_state=0,
            verbose=0, warm_start=False)
```

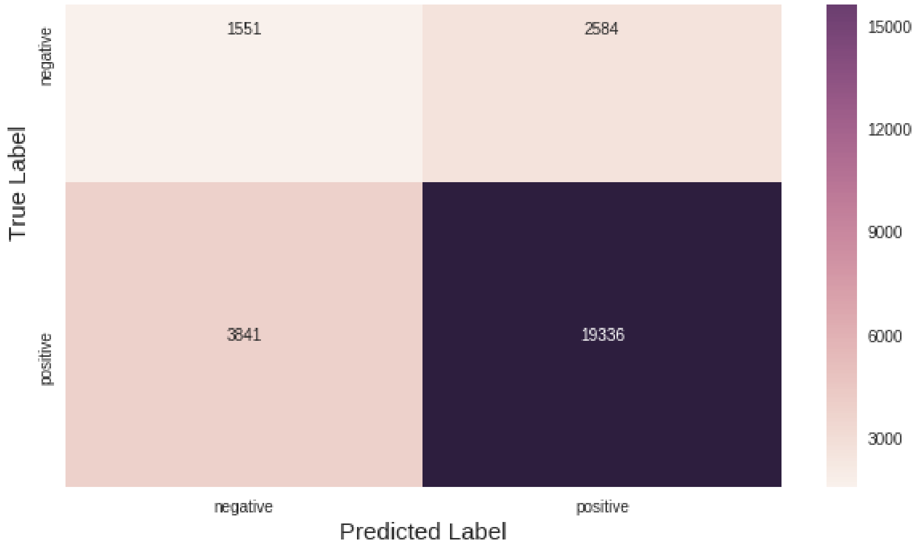
In [0]:

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

In [0]:

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

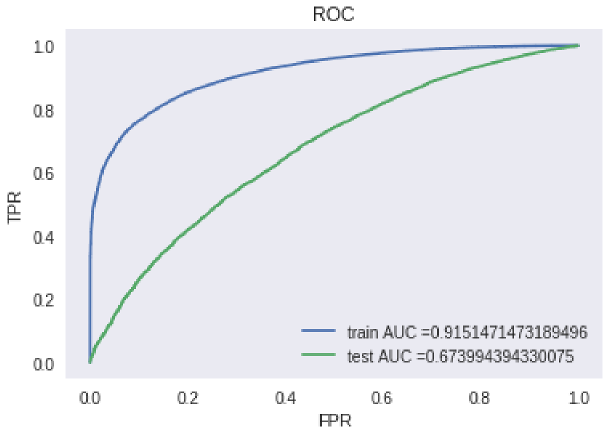




```
True Positives : 19336
False Positives : 2584
True Negatives : 1551
False Negatives : 3841
```

In [0]:

```
start = time.time()
ROC_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: 4.620915853977204
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]:

```
0.673994394330075
```

3.2 Applying GBDT on AVG W2V, SET 3

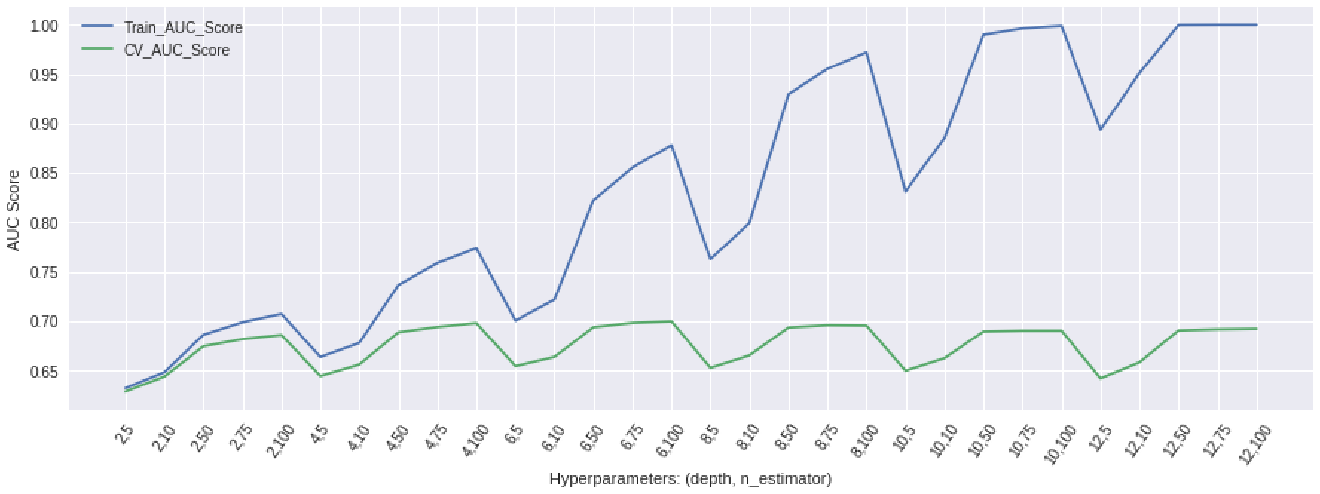
In [0]:

```
optimal_hyp_gbdt(X_tr,y_train,X_cr,y_val)
```

```
execution time in minutes: 24.606902952988943
execution time in hours: 0
```

In [0]:

```
plot_auc(df1)
```



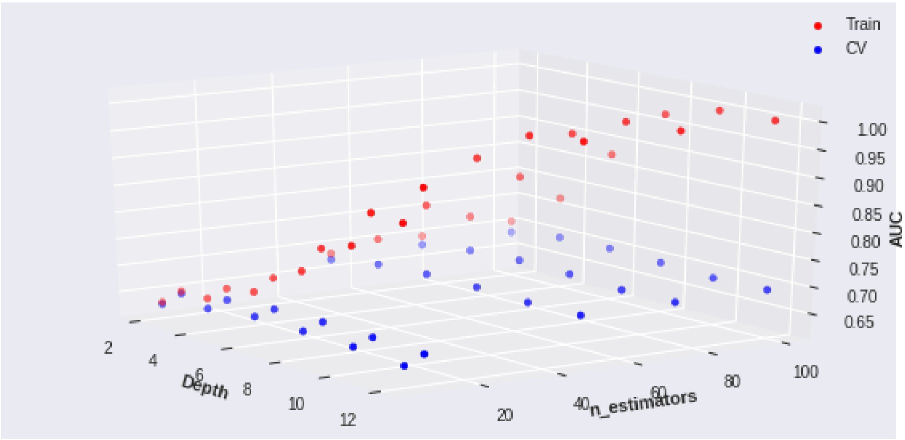
In [0]:

```
best_d,n_est,tr_auc_score,cv_auc_score = df1.iloc[df1.CV_AUC_Score.argmax()]
print("optimal depth:",best_d,"\nn_estimator:",n_est,"\nCV AUC score:", cv_auc_score)
```

```
optimal depth: 6
n_estimator: 100
CV AUC score: 0.6999189005189344
```

In [0]:

```
x = df1['depth'].astype(float).values
y = df1['n_estimator'].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]:

```
model =XGBClassifier(max_depth = int(best_d), n_estimators = int(n_est),class_weight='balanced', ra
ndom_state = 0)
model.fit(X_tr, y_train)
```

Out[0]:

Out[0]:

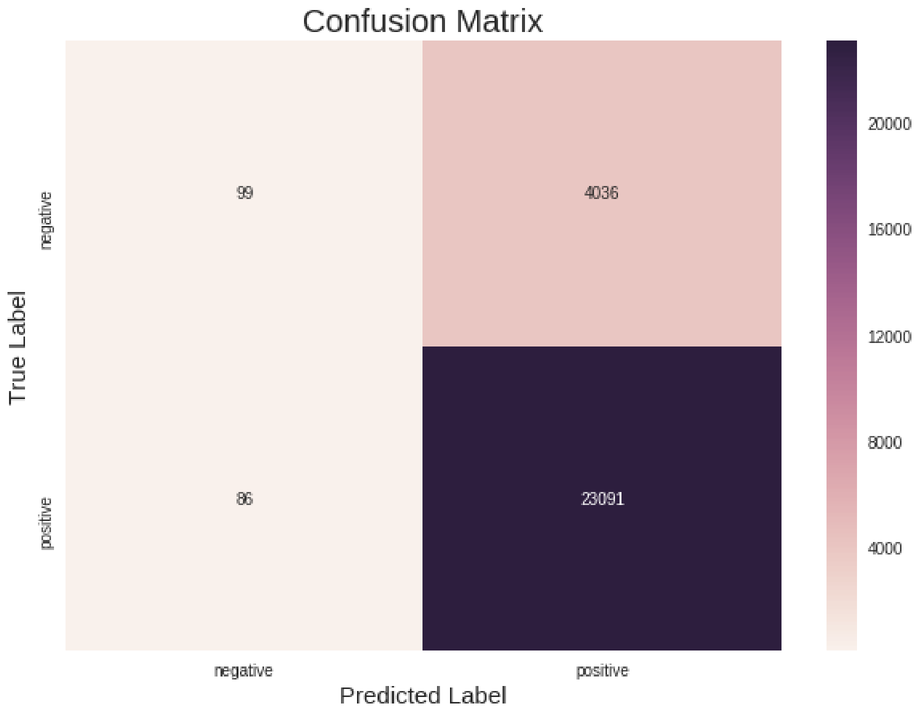
```
XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced',
       colsample_bylevel=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=6, min_child_weight=1, missing=None,
       n_estimators=100, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,
       subsample=1)
```

In [0]:

```
y_pred_te = model.predict(X_te)
labels = ["negative","positive"]
```

In [0]:

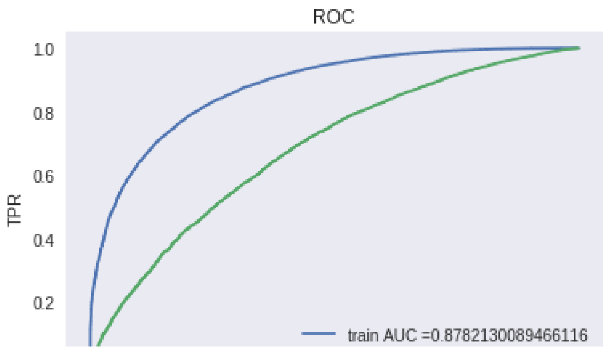
```
get_confusion_matrix_values(np.array(y_test), y_pred_te)
```



```
True Positives : 23091
False Positives : 4036
True Negatives : 99
False Negatives : 86
```

In [0]:

```
start = time.time()
ROC_plot_gbdt(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: 1.344078254699707
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]:

```
0.6836673078776185
```

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-sa
ve-and-load-variables-in-python/
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [0]:

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

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

In [0]:

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

```
tfidfw2v_vec_essay_val = tfidf_w2v(textpreprocessed(X_val,'essay'))
```

```
100%|██████████| 61452/61452 [00:12<00:00, 5079.99it/s]
100%|██████████| 61452/61452 [01:44<00:00, 586.67it/s]
  2%|▏         | 503/27312 [00:00<00:05, 5020.79it/s]
```

```
61452
30
```

```
100%|██████████| 27312/27312 [00:05<00:00, 5063.94it/s]
100%|██████████| 27312/27312 [00:46<00:00, 585.68it/s]
  2%|▏         | 497/20484 [00:00<00:04, 4964.30it/s]
```

```
27312
30
```

```
100%|██████████| 20484/20484 [00:04<00:00, 5108.40it/s]
100%|██████████| 20484/20484 [00:34<00:00, 590.57it/s]
```

```
20484
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%|██████████| 61452/61452 [00:01<00:00, 53827.96it/s]
100%|██████████| 61452/61452 [00:01<00:00, 44901.35it/s]
 19%|█▉        | 5321/27312 [00:00<00:00, 53207.89it/s]
```

```
61452
30
```

```
100%|██████████| 27312/27312 [00:00<00:00, 54036.83it/s]
100%|██████████| 27312/27312 [00:00<00:00, 45235.34it/s]
 27%|██▋       | 5515/20484 [00:00<00:00, 55142.43it/s]
```

```
27312
30
```

```
100%|██████████| 20484/20484 [00:00<00:00, 54667.76it/s]
100%|██████████| 20484/20484 [00:00<00:00, 45641.85it/s]
```

```
20484
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%|██████████| 61452/61452 [00:01<00:00, 33771.55it/s]
100%|██████████| 61452/61452 [00:04<00:00, 14204.48it/s]
 13%|█▎        | 3442/27312 [00:00<00:00, 34413.63it/s]
```

```
61452
30
```

```
100%|██████████| 27312/27312 [00:00<00:00, 33832.45it/s]
100%|██████████| 27312/27312 [00:01<00:00, 14263.86it/s]
 17%|█▋        | 3394/20484 [00:00<00:00, 33925.31it/s]
```

```
27312
```

```
30
```

```
100%|██████████| 20484/20484 [00:00<00:00, 33588.50it/s]
100%|██████████| 20484/20484 [00:01<00:00, 14160.96it/s]
```

```
20484
30
```

In [0]:

```
X_tr, X_cr, X_te = X_tr, X_cr, X_te = hstack_data(coo_matrix(np.asarray(tfidfw2v_vec_titles_tr)), c
oo_matrix(np.asarray(tfidfw2v_vec_titles_val)), coo_matrix(np.asarray(tfidfw2v_vec_titles_te)), \
                   coo_matrix(np.asarray(tfidfw2v_vec_essay_tr)),
coo_matrix(np.asarray(tfidfw2v_vec_essay_val)), coo_matrix(np.asarray(tfidfw2v_vec_essay_te)),\
                   coo_matrix(np.asarray(tfidfw2v_vec_resource_tr)),
coo_matrix(np.asarray(tfidfw2v_vec_resource_val)), coo_matrix(np.asarray(tfidfw2v_vec_resource_te)
))
```

In [0]:

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

Out[0]:

```
((61452, 102), (20484, 102), (27312, 102))
```

4.1 Applying Random Forest on TFIDF W2V, SET 4

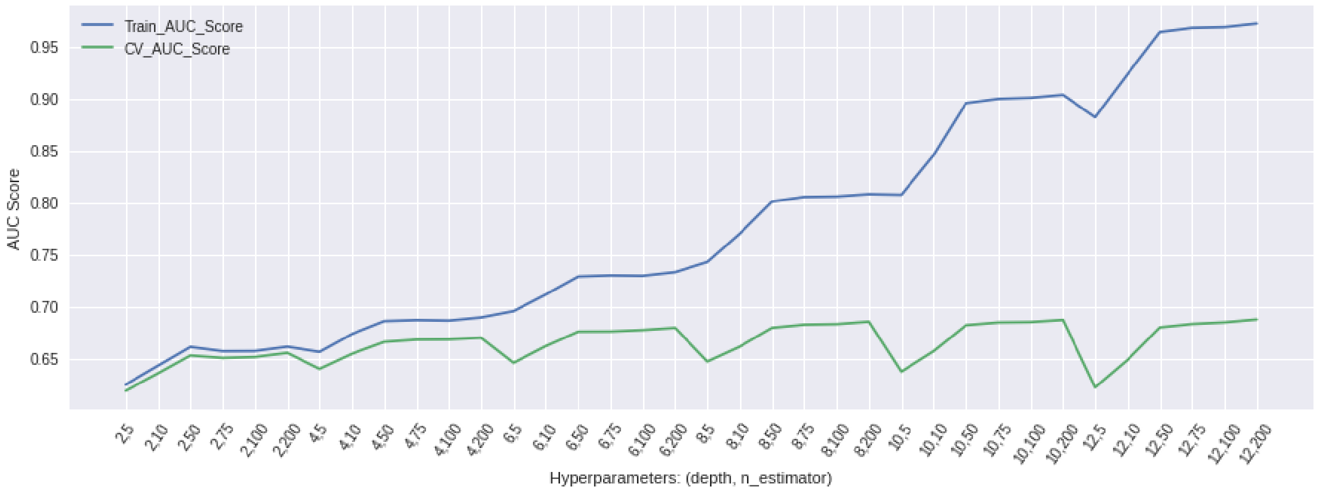
In [0]:

```
optimal_hyp(X_tr,y_train,X_cr,y_val)
```

```
execution time in minutes: 38.001007584730786
execution time in hours: 0
```

In [0]:

```
plot_auc(df1)
```



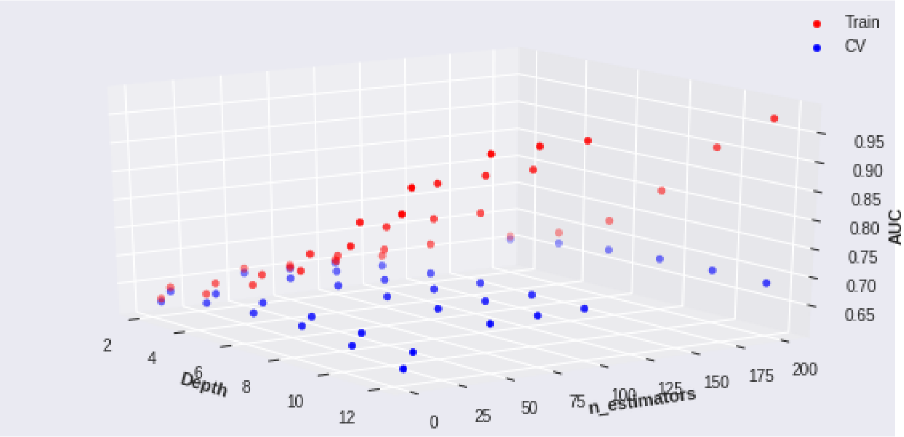
In [0]:

```
best_d,n_est,tr_auc_score,cv_auc_score = df1.iloc[df1.CV_AUC_Score.argmax()]
print("optimal depth:",best_d,"\nn_estimator:",n_est,"\nCV AUC score:", cv_auc_score)
```

```
optimal depth: 12
n_estimator: 200
CV AUC score: 0.6875718902907704
```

In [0]:

```
x = df1['depth'].astype(float).values
y = df1['n_estimator'].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]:

```
model = RandomForestClassifier(max_depth = int(best_d), n_estimators =
int(n_est),class_weight='balanced', random_state = 0)
model.fit(X_tr, y_train)
```

Out[0]:

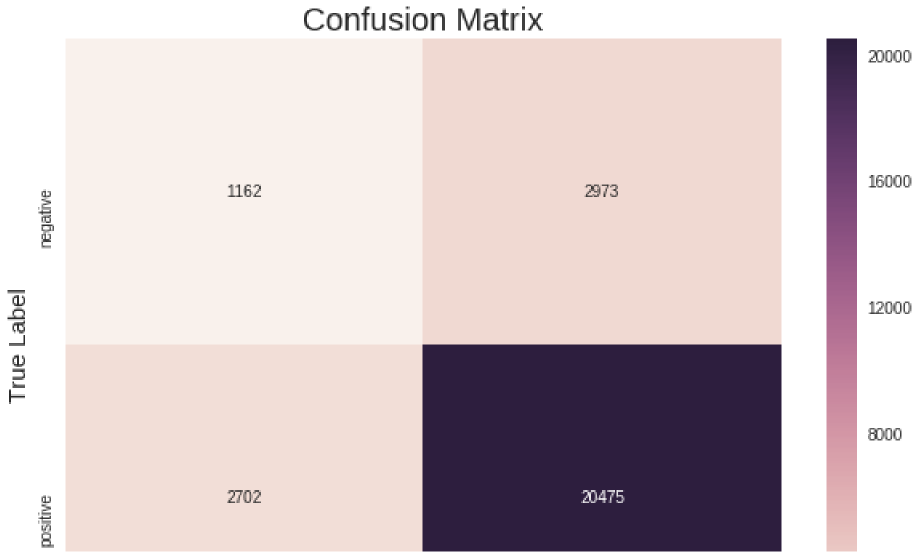
```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=12, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=200, n_jobs=None, oob_score=False, random_state=0,
            verbose=0, warm_start=False)
```

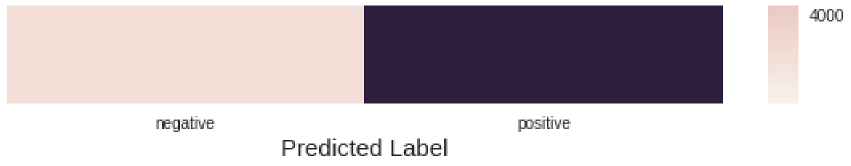
In [0]:

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

In [0]:

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

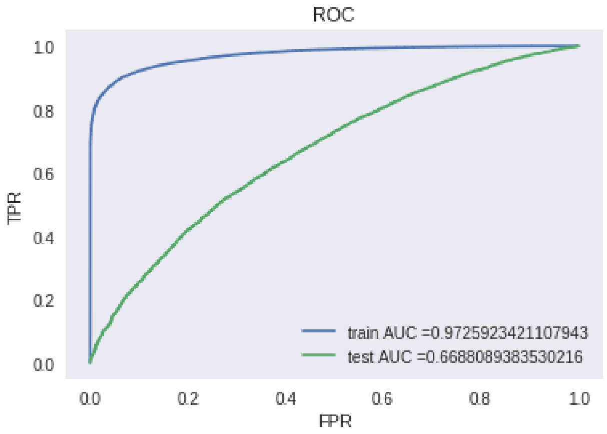




```
True Positives : 20475
False Positives : 2973
True Negatives : 1162
False Negatives : 2702
```

In [0]:

```
start = time.time()
ROC_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: 5.961549035708109
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]:

```
0.6688089383530216
```

4.2 Applying GBDT on TFIDF W2V, SET 4

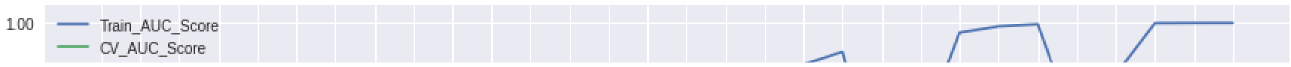
In [0]:

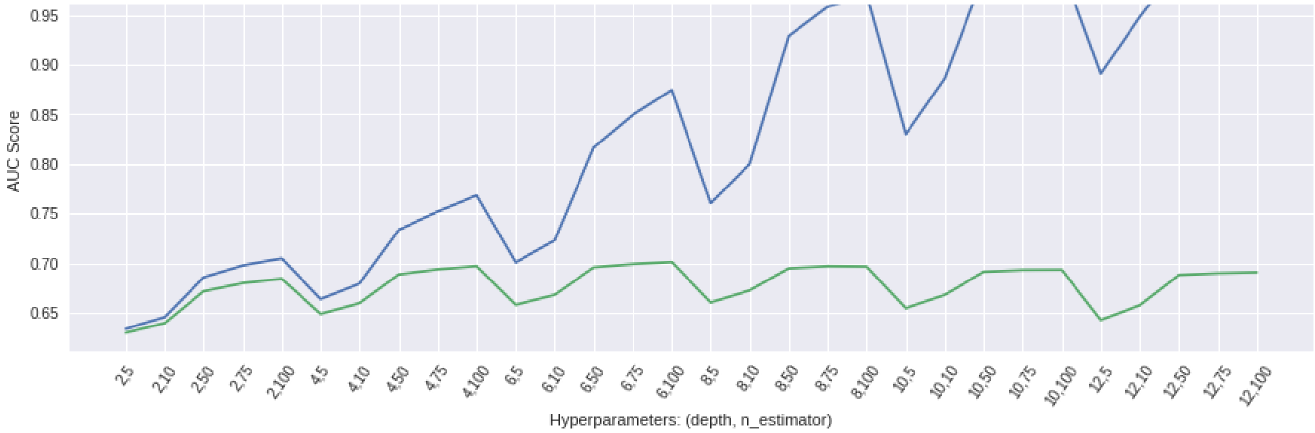
```
optimal_hyp_gbdt(X_tr,y_train,X_cr,y_val)
```

```
execution time in minutes: 24.36458974679311
execution time in hours: 0
```

In [0]:

```
plot_auc(df1)
```





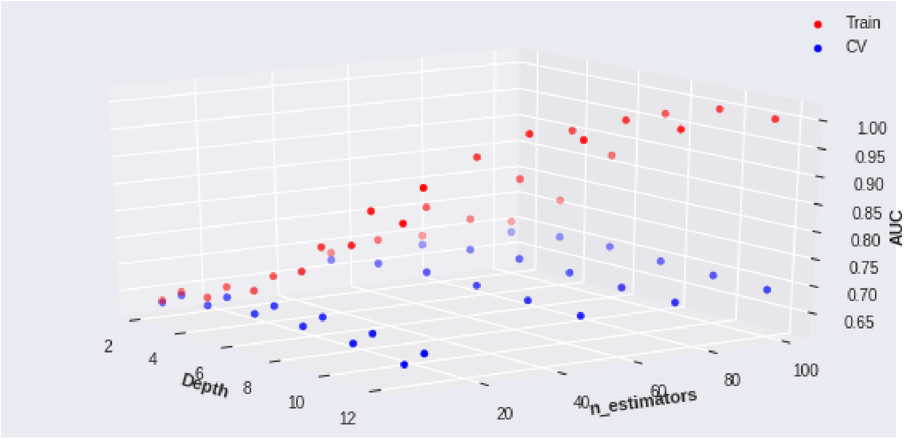
In [0]:

```
best_d,n_est,tr_auc_score,cv_auc_score = df1.iloc[df1.CV_AUC_Score.argmax()]
print("optimal depth:",best_d,"\nn_estimator:",n_est,"\nCV AUC score:", cv_auc_score)
```

```
optimal depth: 6
n_estimator: 100
CV AUC score: 0.7011028809826538
```

In [0]:

```
x = df1['depth'].astype(float).values
y = df1['n_estimator'].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]:

```
model = XGBClassifier(max_depth = int(best_d), n_estimators = int(n_est),class_weight='balanced', r
andom_state = 0)
model.fit(X_tr, y_train)
```

Out[0]:

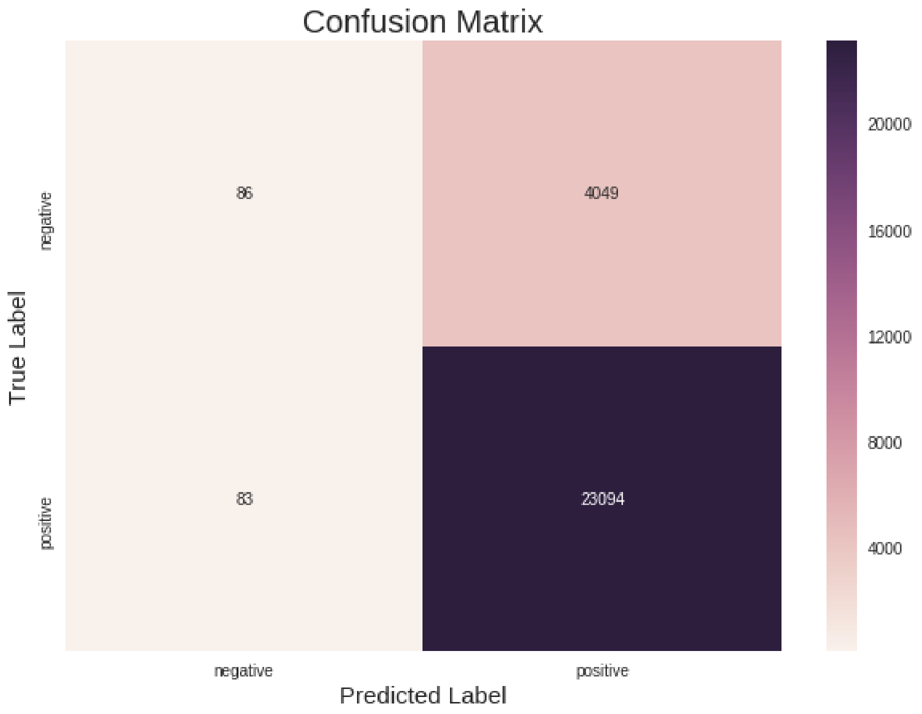
```
XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced',
       colsample_bylevel=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=6, min_child_weight=1, missing=None,
       n_estimators=100, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,
       subsample=1)
```

In [0]:

```
y_pred_te = model.predict(X_te)
labels = ["negative","positive"]
```

In [0]:

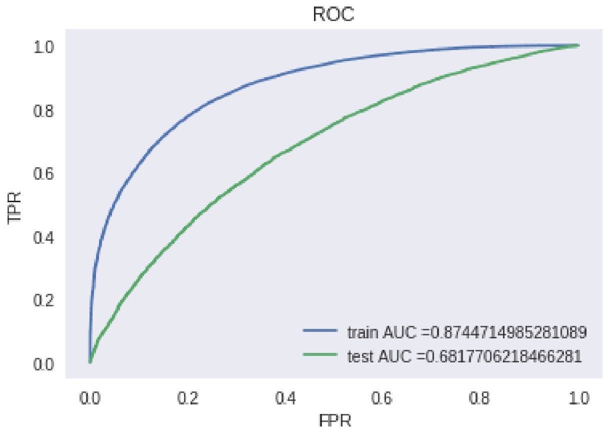
```
get_confusion_matrix_values(np.array(y_test), y_pred_te)
```



```
True Positives : 23094
False Positives : 4049
True Negatives : 86
False Negatives : 83
```

In [0]:

```
start = time.time()
ROC_plot_gbdt(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: 1.3221298853556316
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]:

```
0.6817706218466281
```

Summary

In [198]:

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

x = PrettyTable()
x.field_names = ["Vectorizer", "Optimal Depth","n_estimator", "AUC"]

x.add_row(["BOW", 12, 200, 0.6968])
x.add_row(["TFIDF", 12, 200, 0.6960])
x.add_row(["AVGW2V", 10, 200, 0.6739])
x.add_row(["TFIDFW2V", 12, 200, 0.6688])

print("======================== RF =========================")
print(x)

x = PrettyTable()
x.field_names = ["Vectorizer", "Optimal Depth","n_estimator", "AUC"]

x.add_row(["BOW", 10, 100, 0.7302])
x.add_row(["TFIDF", 10, 100, 0.7292])
x.add_row(["AVGW2V", 6, 100, 0.6836])
x.add_row(["TFIDFW2V", 6, 100, 0.6817])

print("\n\n======================= GBDT ========================")
print(x)
```

```
======================== RF =========================
+------------+---------------+-------------+--------+
| Vectorizer | Optimal Depth | n_estimator |  AUC   |
+------------+---------------+-------------+--------+
|    BOW     |       12      |     200     | 0.6968 |
|   TFIDF    |       12      |     200     | 0.696  |
|   AVGW2V   |       10      |     200     | 0.6739 |
|  TFIDFW2V  |       12      |     200     | 0.6688 |
+------------+---------------+-------------+--------+


======================= GBDT ========================
+------------+---------------+-------------+--------+
| Vectorizer | Optimal Depth | n_estimator |  AUC   |
+------------+---------------+-------------+--------+
|    BOW     |       10      |     100     | 0.7302 |
|   TFIDF    |       10      |     100     | 0.7292 |
|   AVGW2V   |       6       |     100     | 0.6836 |
|  TFIDFW2V  |       6       |     100     | 0.6817 |
+------------+---------------+-------------+--------+
```

Conclusion

- GBDT outperforms RF in terms of AUC.
- BOW and TFIDF vectorizer gives better AUC than the rest of vectorizers.
- For this dataset, Model works better with depth between 6-12

Procedure followed :

- 1) Do Text Preprocessing.
- 2) Random split the dataset into train data, cross validate data and test data.
- 3) Do response coding for categorical data.
- 4) Standardize the numerical data.
- 5) Vectorize the text data by fitting the train data and then transform train, val and test data.

- 6) Train the RF and GBDT and finding the optimal n_estimator and optimal depth using the Max AUC method(used for tuning hyperparameters.)
- 7) Plot 3D plot of Train and Cv AUC score with hyperparameters depth and n_estimator respectively.
- 8) After getting the optimal n_estimator and optimal depth , Train the model again and predict the values.
- 9) Plot confusion matrix.
- 10)Plot ROC curve for train and test data respectively.

Repeat from 5) to 10) for BoW, TFIDF, Avg Word2Vec and TFIDF Word2Vec vectorizer