Assignment 11: TruncatedSVD

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

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

In [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.metrics import accuracy_score, roc_auc_score
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/2582163/408
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_da

#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
project_data.drop('project_submitted_datetime', axis=1, inplace=True)
project_data.sort_values(by=['Date'], inplace=True)

# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084
project_data = project_data[cols]
#project_data.head(2)
```

In [0]:
```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).r
#price_data.head(2)
```

In [0]:
```
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

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.co

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

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

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

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

Preprocessing of project_subject_subcategories

In [0]:
```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.co

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

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

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

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4
my_counter = Counter()
for word in project_data['clean_subcategories'].values:
    my_counter.update(word.split())

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

1.3 Text preprocessing

In [0]:
```
# merge two column text dataframe:
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(str) + \
                        project_data["project_essay_4"].map(str)
```

In [0]:
```
# https://stackoverflow.com/a/47091490/4084039
import re

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

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

In [0]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', '
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itsel
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because'
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'th
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all'
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than',
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "di
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn'
            'won', "won't", 'wouldn', "wouldn't"]
```

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

project_data['numerical_digits'] = project_data['project_resource_summary'].apply
```

In [0]:
```
project_data['project_grade_category']=project_data['project_grade_category'].str
project_data['project_grade_category']=project_data['project_grade_category'].str
set(project_data['project_grade_category'])
```

Out[22]:
```
{'Grades_3to5', 'Grades_6to8', 'Grades_9to12', 'Grades_PreKto2'}
```

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

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

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

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

Out[25]:
```
((61178, 20), (26220, 20), (21850, 20))
```

In [0]:
```
from sklearn.preprocessing import StandardScaler
def standardize_data(df_tr,df_cv,df_te,column_name):
  standardized_vec = StandardScaler(with_mean=False)
  # here it will learn mu and sigma
  standardized_vec.fit(df_tr[column_name].values.reshape(-1,1))

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

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

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

  return standardized_data_train, standardized_data_traincv, standardized_data_te
```

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

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

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

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

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

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

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

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

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

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

In [0]:
```
price_standardized_tr, price_standardized_val, price_standardized_te = standardiz
print()
project_standardized_tr, project_standardized_val, project_standardized_te = stan
```

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

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

In [0]:
```
cat_one_hot_tr,cat_one_hot_val,cat_one_hot_te,cat_one_hot_list = vectorized_data(
cat_sub_one_hot_tr,cat_sub_one_hot_val,cat_sub_one_hot_te,cat_sub_one_hot_list =
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'S
pecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding  (61178, 9)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'S
pecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding  (26220, 9)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'S
pecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding  (21850, 9)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'S
pecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding  (61178, 9)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'S
pecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding  (26220, 9)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'S
pecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding  (21850, 9)
```

In [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 = vectoriz
teacher_one_hot_tr,teacher_one_hot_val,teacher_one_hot_te,teacher_one_hot_list =
```

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

In [0]:
```
grade_dict = create_dict(X_train,'project_grade_category')
grade_one_hot_tr,grade_one_hot_val,grade_one_hot_te,grade_one_hot_list = vectoriz
```

```
['Grades_9to12', 'Grades_6to8', 'Grades_3to5', 'Grades_PreKto2']
Shape of matrix after one hot encoding  (61178, 4)
['Grades_9to12', 'Grades_6to8', 'Grades_3to5', 'Grades_PreKto2']
Shape of matrix after one hot encoding  (26220, 4)
['Grades_9to12', 'Grades_6to8', 'Grades_3to5', 'Grades_PreKto2']
Shape of matrix after one hot encoding  (21850, 4)
```

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

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

Features Engineering

In [0]:
```
feature_list_x = ['dig_0','dig_1']+grade_one_hot_list+state_one_hot_list+cat_one_
len(feature_list_x)
```

Out[37]: 82

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

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

In [0]:
```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
'''nltk.download('vader_lexicon')
sid = SentimentIntensityAnalyzer()
p = project_data.copy()
sentiments = []
for i in range(p.shape[0]):
  line = p['essay'].iloc[i]
  sentiment = sid.polarity_scores(line)
  sentiments.append([sentiment['neg'], sentiment['pos'], sentiment['neu'], sentim
  p[['neg', 'pos', 'neu', 'compound']] = pd.DataFrame(sentiments)
  '''
#[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
#[nltk_data]   Package vader_lexicon is already up-to-date!
```

Out[39]:
```
"nltk.download('vader_lexicon')\nsid = SentimentIntensityAnalyzer()\np = projec
t_data.copy()\nsentiments = []\nfor i in range(p.shape[0]):\n  line = p['essa
y'].iloc[i]\n  sentiment = sid.polarity_scores(line)\n  sentiments.append([sent
iment['neg'], sentiment['pos'], sentiment['neu'], sentiment['compound']])\n  p
[['neg', 'pos', 'neu', 'compound']] = pd.DataFrame(sentiments)\n  "
```

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

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

In [0]:
```
project_data['words_in_title'] = project_data['project_title'].str.count(' ') + 1
project_data['words_in_essay'] = project_data['essay'].str.count(' ') + 1
project_data['words_in_summary'] = project_data['project_resource_summary'].str.c
```

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

In [0]:
```
project_data_features = project_data.copy()
project_data_features.drop('project_is_approved', axis=1, inplace=True)
y=list(project_data['project_is_approved'])
X_train, X_test, y_train, y_test = train_test_split(project_data_features, y, str
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, stratify=y_tr
```

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

Out[45]: ((67296, 27), (20102, 27), (21850, 27))

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

state_one_hot_tr,state_one_hot_val,state_one_hot_te,_ = vectorized_data(X_train,X
teacher_one_hot_tr,teacher_one_hot_val,teacher_one_hot_te,_ = vectorized_data(X_t
grade_dict = create_dict(X_train,'project_grade_category')
grade_one_hot_tr,grade_one_hot_val,grade_one_hot_te,_ = vectorized_data(X_train,X
one_hot_num_dig_tr = num_hot_encode(X_train,'numerical_digits')
one_hot_num_dig_te = num_hot_encode(X_test,'numerical_digits')
one_hot_num_dig_val = num_hot_encode(X_val,'numerical_digits')
```

```
(67296, 1)
(20102, 1)
(21850, 1)

(67296, 1)
(20102, 1)
(21850, 1)

(67296, 1)
(20102, 1)
(21850, 1)

(67296, 1)
(20102, 1)
(21850, 1)
(67296, 1)
(20102, 1)
(21850, 1)
(67296, 1)
(20102, 1)
(21850, 1)

(67296, 1)
(20102, 1)
(21850, 1)
```

```
(67296, 1)
(20102, 1)
(21850, 1)

(67296, 1)
(20102, 1)
(21850, 1)

(67296, 1)
(20102, 1)
(21850, 1)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'S
pecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding  (67296, 9)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'S
pecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding  (20102, 9)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'S
pecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding  (21850, 9)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'S
pecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding  (67296, 9)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'S
pecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding  (20102, 9)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'S
pecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding  (21850, 9)
['VT', 'WY', 'ND', 'MT', 'RI', 'NE', 'SD', 'NH', 'AK', 'DE', 'ME', 'HI', 'WV',
'DC', 'NM', 'KS', 'IA', 'ID', 'AR', 'CO', 'MN', 'OR', 'MS', 'KY', 'NV', 'MD',
'CT', 'UT', 'TN', 'AL', 'WI', 'AZ', 'VA', 'NJ', 'OK', 'WA', 'MA', 'LA', 'OH',
'IN', 'MO', 'MI', 'PA', 'SC', 'GA', 'IL', 'NC', 'FL', 'NY', 'TX', 'CA']
Shape of matrix after one hot encoding  (67296, 51)
['VT', 'WY', 'ND', 'MT', 'RI', 'NE', 'SD', 'NH', 'AK', 'DE', 'ME', 'HI', 'WV',
'DC', 'NM', 'KS', 'IA', 'ID', 'AR', 'CO', 'MN', 'OR', 'MS', 'KY', 'NV', 'MD',
'CT', 'UT', 'TN', 'AL', 'WI', 'AZ', 'VA', 'NJ', 'OK', 'WA', 'MA', 'LA', 'OH',
'IN', 'MO', 'MI', 'PA', 'SC', 'GA', 'IL', 'NC', 'FL', 'NY', 'TX', 'CA']
Shape of matrix after one hot encoding  (20102, 51)
['VT', 'WY', 'ND', 'MT', 'RI', 'NE', 'SD', 'NH', 'AK', 'DE', 'ME', 'HI', 'WV',
'DC', 'NM', 'KS', 'IA', 'ID', 'AR', 'CO', 'MN', 'OR', 'MS', 'KY', 'NV', 'MD',
'CT', 'UT', 'TN', 'AL', 'WI', 'AZ', 'VA', 'NJ', 'OK', 'WA', 'MA', 'LA', 'OH',
'IN', 'MO', 'MI', 'PA', 'SC', 'GA', 'IL', 'NC', 'FL', 'NY', 'TX', 'CA']
Shape of matrix after one hot encoding  (21850, 51)
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
Shape of matrix after one hot encoding  (67296, 5)
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
Shape of matrix after one hot encoding  (20102, 5)
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
Shape of matrix after one hot encoding  (21850, 5)
['Grades_9to12', 'Grades_6to8', 'Grades_3to5', 'Grades_PreKto2']
Shape of matrix after one hot encoding  (67296, 4)
['Grades_9to12', 'Grades_6to8', 'Grades_3to5', 'Grades_PreKto2']
Shape of matrix after one hot encoding  (20102, 4)
['Grades_9to12', 'Grades_6to8', 'Grades_3to5', 'Grades_PreKto2']
Shape of matrix after one hot encoding  (21850, 4)
Shape of matrix after one hot encoding  (67296, 2)
```

```
Shape of matrix after one hot encoding  (21850, 2)
Shape of matrix after one hot encoding  (20102, 2)
```

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

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

In [0]:
```
def hstack_data(f1_tr, f1_cr, f1_te, f2_tr, f2_cr, f2_te):
  X_tr = hstack((f_tr, f1_tr, f2_tr)).tocsr()
  X_cr = hstack((f_val, f1_cr, f2_cr)).tocsr()
  X_te = hstack((f_te, f1_te, f2_te)).tocsr()
  return X_tr,X_cr,X_te
```

In [0]:
```
'''from sklearn.utils import resample
X_train1 = X_train
X_train_neg = X_train1[X_train.project_is_approved==0]
X_train_neg_upsampled = resample(X_train_neg,replace=True,n_samples=38000,random_
final'''
```

Out[49]:
```
'from sklearn.utils import resample\nX_train1 = X_train\nX_train_neg = X_train1
[X_train.project_is_approved==0]\nX_train_neg_upsampled = resample(X_train_neg,
replace=True,n_samples=38000,random_state=1)\nfinal'
```

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

def tfidf_vec(preprocessed_data_tr,preprocessed_data_val,preprocessed_data_te):
  vectorizer = TfidfVectorizer(min_df=10)
  text_tfidf_tr = vectorizer.fit_transform(preprocessed_data_tr)
  print("Shape of matrix after one hot encodig ",text_tfidf_tr.shape)

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

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

tfidf_vec_essay_tr,tfidf_vec_essay_val,tfidf_vec_essay_te = tfidf_vec(textpreproc
tfidf_vec_titles_tr,tfidf_vec_titles_val,tfidf_vec_titles_te = tfidf_vec(textprep
#tfidf_vec_resource_tr,tfidf_vec_resource_val,tfidf_vec_resource_te = tfidf_vec(t
```

```
100%|██████████| 67296/67296 [00:57<00:00, 1172.09it/s]
100%|██████████| 20102/20102 [00:22<00:00, 907.41it/s]
100%|██████████| 21850/21850 [00:23<00:00, 916.10it/s]

Shape of matrix after one hot encodig  (67296, 13657)
Shape of matrix after one hot encodig  (20102, 13657)

  3%|▌          | 1938/67296 [00:00<00:03, 19376.55it/s]

Shape of matrix after one hot encodig  (21850, 13657)

100%|██████████| 67296/67296 [00:03<00:00, 21070.02it/s]
100%|██████████| 20102/20102 [00:22<00:00, 895.12it/s]
100%|██████████| 21850/21850 [00:01<00:00, 20803.37it/s]

Shape of matrix after one hot encodig  (67296, 2412)
Shape of matrix after one hot encodig  (20102, 2412)
Shape of matrix after one hot encodig  (21850, 2412)
```

2.1 Selecting top 2000 words from essay and project_title

In [0]:
```
xtr=X_train
```

In [0]:
```
X = xtr['essay']+ ' ' + xtr['project_title']
```

In [0]:
```
%%time
# Featurization
tfidf = TfidfVectorizer(use_idf = True, max_df = 0.10)
feat = tfidf.fit_transform(X)
feat.shape
```

```
CPU times: user 10.3 s, sys: 157 ms, total: 10.5 s
Wall time: 10.5 s
```

In [0]:
```
%%time
# Get feature names from tfidf
features = tfidf.get_feature_names()
#print(len(features))
# feature weights based on idf score
coef = tfidf.idf_
coeff_df = pd.DataFrame({'Features' : features, 'Idf_score' : coef})
coeff_df = coeff_df.sort_values("Idf_score", ascending = True)[:2000]
#print(coeff_df)
print("shape of selected features :", coeff_df.shape)
print("Top 10 features :\n\n",coeff_df[0:10])
```

```
shape of selected features : (2000, 2)
Top 10 features :

           Features  Idf_score
50938     wonderful   3.305666
36412    population   3.307008
25156       limited   3.311496
41424       seating   3.312096
4611          based   3.313897
42466           sit   3.314198
22226   instruction   3.314800
38408       readers   3.316004
24951        levels   3.318265
6271          bring   3.332096
CPU times: user 55.7 ms, sys: 992 µs, total: 56.7 ms
Wall time: 56.3 ms
```

2.2 Computing Co-occurance matrix

In [0]:
```
# https://stackoverflow.com/questions/41661801/python-calculate-the-co-occurrence
def cal_occ(sentence,m):
    words = sentence.split(" ")
    for i,word in enumerate(words):
        #print(word)
        for j in range(max(i-window,0),min(i+window,len(words))):
          if (words[j] != words[i]):
            try:
              df.loc[words[i], words[j]] += 1
              df.loc[words[j], words[i]] += 1

            except:
              pass
```

In [0]:
```
%%time
length = len(coeff_df)
m = np.zeros([length,length]) # n is the count of all words
window = 5
zeros = np.zeros((length,length))
df = pd.DataFrame(zeros, index = coeff_df["Features"], columns = coeff_df["Featur
for sentence in X:
    #print(sentence)
    cal_occ(sentence, m)
```

```
CPU times: user 3h 37min 41s, sys: 1.65 s, total: 3h 37min 43s
Wall time: 3h 38min 10s
```

In [0]:
```
df.head()
```

Out[55]:

Features	wonderful	population	limited	seating	based	sit	instruction	readers	levels	brin
Features										
wonderful	0.0	36.0	9.0	39.0	18.0	4.0	7.0	25.0	9.0	68.
population	36.0	0.0	20.0	3.0	8.0	1.0	2.0	2.0	11.0	9.
limited	9.0	20.0	0.0	65.0	27.0	13.0	11.0	12.0	11.0	15.
seating	39.0	3.0	65.0	0.0	70.0	160.0	35.0	3.0	15.0	80.
based	18.0	8.0	27.0	70.0	0.0	18.0	184.0	8.0	47.0	19.

5 rows × 2000 columns

In [0]:
```
df.shape
```

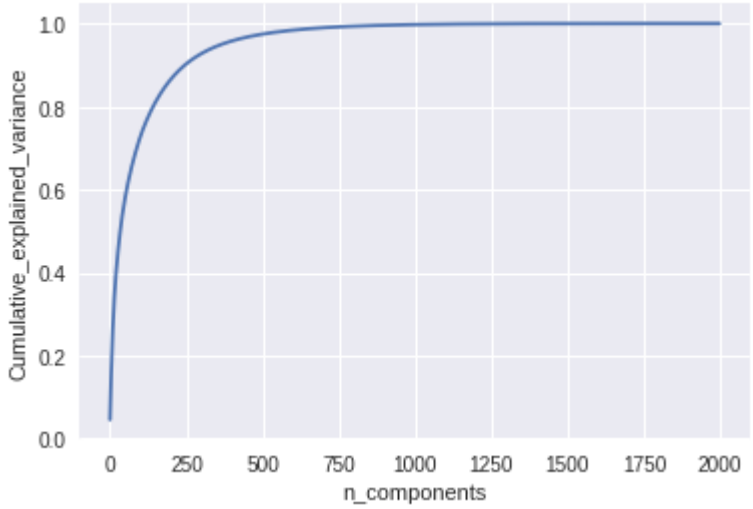
Out[56]:
```
(2000, 2000)
```

In [0]:
```
```

In [0]:
```
with open('comatrix', 'rb') as f:
   df = pickle.load(f)
```

2.3 Applying TruncatedSVD and Calculating Vectors for essay and project_title

In [0]:
```
%%time
from sklearn.decomposition import TruncatedSVD
# https://chrisalbon.com/machine_learning/feature_engineering/select_best_number_
# Create and run an TSVD with one less than number of features
tsvd = TruncatedSVD(n_components=1999)
X_tsvd = tsvd.fit(df) ################## taking 4500 because on 5000 it crashes
percentage_var_explained = tsvd.explained_variance_ / np.sum(tsvd.explained_varia
cum_var_explained = np.cumsum(percentage_var_explained)

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



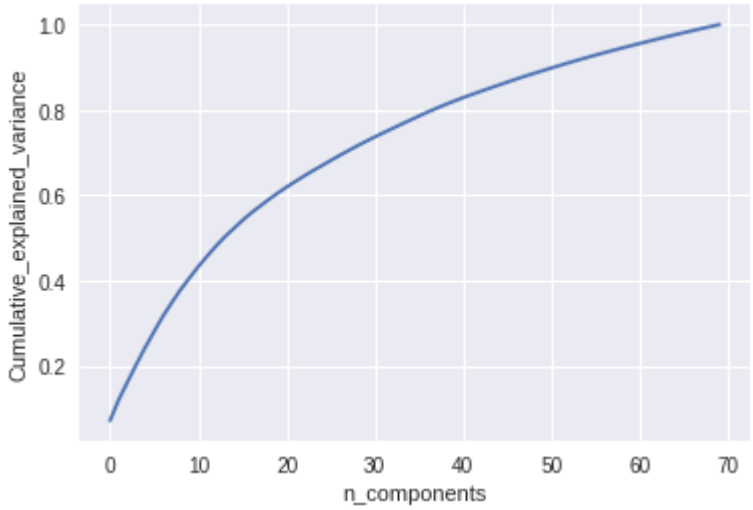

```
CPU times: user 32.3 s, sys: 10 s, total: 42.3 s
Wall time: 32.1 s
```

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

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

    # Set initial number of features
    n_components = 0

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

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

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

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

    # Return the number of components
    return n_components
# Run function
n=select_n_components(tsvd_var_ratios, 0.65)
n
```

Out[55]: 71

In [0]:
```
%%time
from sklearn.decomposition import TruncatedSVD
# https://chrisalbon.com/machine_learning/feature_engineering/select_best_number_
# Create and run an TSVD with one less than number of features
tsvd = TruncatedSVD(n_components=70)
X_tsvd = tsvd.fit_transform(df)
#X_tsvd_data = tsvd.transform(X_tsvd)

percentage_var_explained = tsvd.explained_variance_ / np.sum(tsvd.explained_varia
cum_var_explained = np.cumsum(percentage_var_explained)

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




```
CPU times: user 1.29 s, sys: 538 ms, total: 1.83 s
Wall time: 1.81 s
```

In [0]:
```
best_features = [df.index[i] for i in tsvd.components_[0].argsort()[::-1]]
len(best_features)
```

Out[57]:
```
2000
```

2.4 Merge the features from step 3 and step 4

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

def tfidf_vec(preprocessed_data_tr,preprocessed_data_val,preprocessed_data_te):
  vectorizer = TfidfVectorizer(vocabulary=list(set(best_features)))
  text_tfidf_tr = vectorizer.fit_transform(preprocessed_data_tr)
  print("Shape of matrix after one hot encodig ",text_tfidf_tr.shape)

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

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

In [0]:
```
tfidf_vec_essay_tr,tfidf_vec_essay_val,tfidf_vec_essay_te = tfidf_vec(textpreproc
tfidf_vec_titles_tr,tfidf_vec_titles_val,tfidf_vec_titles_te = tfidf_vec(textprep
tfidf_vec_resource_tr,tfidf_vec_resource_val,tfidf_vec_resource_te = tfidf_vec(te
```

```
100%|██████████| 67296/67296 [02:04<00:00, 538.80it/s]
100%|██████████| 20102/20102 [00:37<00:00, 534.52it/s]
100%|██████████| 21850/21850 [00:40<00:00, 538.55it/s]

Shape of matrix after one hot encodig  (67296, 2000)
Shape of matrix after one hot encodig  (20102, 2000)

  0%|          | 0/67296 [00:00<?, ?it/s]

Shape of matrix after one hot encodig  (21850, 2000)

100%|██████████| 67296/67296 [00:05<00:00, 12689.98it/s]
100%|██████████| 20102/20102 [00:38<00:00, 520.29it/s]
100%|██████████| 21850/21850 [00:01<00:00, 13359.11it/s]

Shape of matrix after one hot encodig  (67296, 2000)
Shape of matrix after one hot encodig  (20102, 2000)

  1%|          | 772/67296 [00:00<00:08, 7717.67it/s]

Shape of matrix after one hot encodig  (21850, 2000)

100%|██████████| 67296/67296 [00:12<00:00, 5466.96it/s]
100%|██████████| 20102/20102 [00:35<00:00, 561.89it/s]
100%|██████████| 21850/21850 [00:03<00:00, 5600.03it/s]

Shape of matrix after one hot encodig  (67296, 2000)
Shape of matrix after one hot encodig  (20102, 2000)
Shape of matrix after one hot encodig  (21850, 2000)
```

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

In [0]:
```
tfidf_titles_tr = tsvd.fit_transform(tfidf_vec_titles_tr)
tfidf_titles_val = tsvd.transform(tfidf_vec_titles_val)
tfidf_titles_te = tsvd.transform(tfidf_vec_titles_te)
```

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

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

Out[63]:
```
((67296, 230), (20102, 230), (21850, 230))
```

In [0]:
```
xtrain = pd.DataFrame(X_tr.toarray())
```

In [0]:
```
xtrain['label'] = y_train
final = xtrain
```

In [0]:
```
final.sort_values(by='label', ascending=False, inplace=True)
```

In [0]:
```
n_zeros = final[final['label']==0]
n_ones = final[final['label']==1]
diff = n_ones.shape[0] - n_zeros.shape[1]
n_zeros.shape, n_ones.shape
```

Out[71]:
```
((10190, 231), (57106, 231))
```

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

In [0]:
```
final.shape
```

Out[81]:
```
(84769, 231)
```

In [0]:
```
y_train = final['label']
X_tr = final.drop('label', axis=1)
```

In [0]:
```
import scipy
X_tr = scipy.sparse.csr_matrix(X_tr.values)
```

In [0]:
```
'''with open('train.pickle', 'rb') as f:
    X_tr, y_train = pickle.load(f)'''
```

In [0]:
```
'''with open('test.pickle', 'rb') as f:
    X_te, y_test = pickle.load(f)'''
```

In [0]:
```
'''with open('val.pickle', 'rb') as f:
    X_cr, y_val = pickle.load(f)'''
```

In [0]:
```
'''with open('comatrix', 'rb') as f:
   df = pickle.load(f)'''
```

2.5 Apply XGBoost on the Final Features from the above

section

In [0]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.htm
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix
import seaborn as sns
def 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_
  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]:
```
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='balanc
        clf.fit(X_tr, y_train)
        tr_score = roc_auc_score(y_true=np.array(y_train), y_score=clf.predict_pr
        cv_score = roc_auc_score(y_true=np.array(y_val), y_score=clf.predict_prob
        #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://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()
```

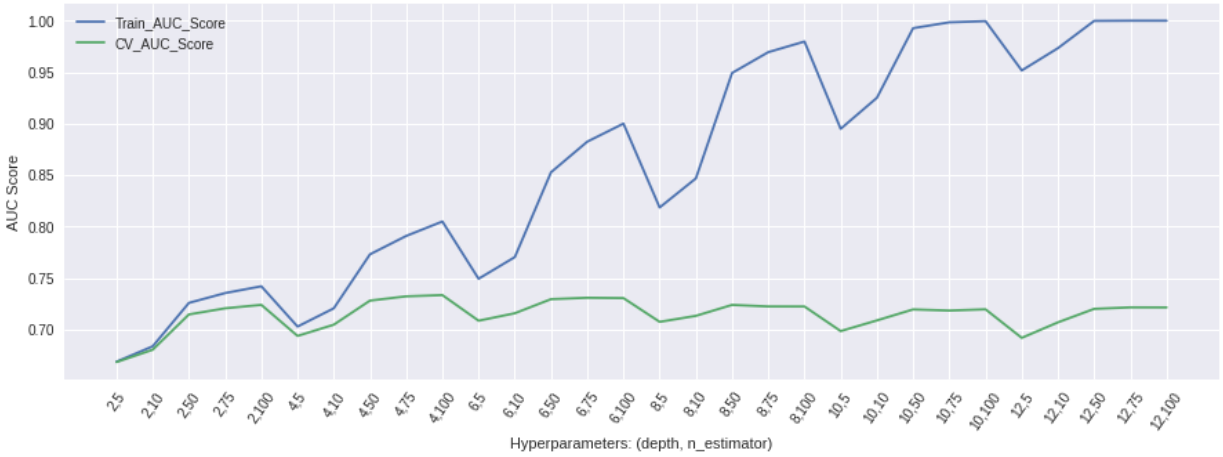
In [0]:
```
def plot_auc(df1):
  df = df1[['Train_AUC_Score','CV_AUC_Score']]
  df1['depth'] = df1['depth'].astype(str)
  df1['n_estimator'] = df1['n_estimator'].astype(str)

  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), a
  ax
```

In [0]:
```
#https://datascience.stackexchange.com/questions/28493/confusion-matrix-get-items
def get_confusion_matrix_values(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    df = pd.DataFrame(data=cm, index=labels, columns=labels)
    #tn, fp, fn, tp = cm.ravel()
    #print(tn,fp,fn,tp)
    #print("Confusion Matrix : ")
    plt.figure(figsize=(10,7))
    sns.heatmap(df, annot=True,fmt = "d")
    plt.title("Confusion Matrix",fontsize=20)
    plt.xlabel("Predicted Label",fontsize=15)
    plt.ylabel("True Label",fontsize=15)
    plt.show()
    TN, FP, FN, TP = cm[0][0], cm[0][1], cm[1][0], cm[1][1]
    print("True Positives :", TP)
    print("False Positives :", FP)
    print("True Negatives :", TN)
    print("False Negatives :", FN)
```

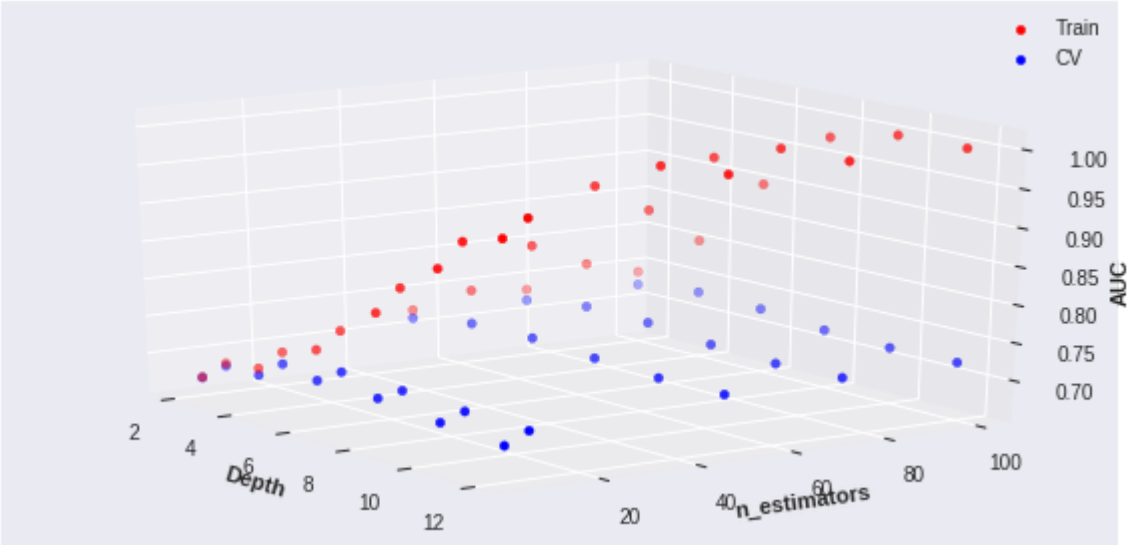
In [0]:
```
optimal_hyp_gbdt(X_tr,y_train,X_cr,y_val)
```

```
execution time in minutes: 101.96842641830445
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_sc
```

```
optimal depth: 4
n_estimator: 100
CV AUC score: 0.7336074079175493
```

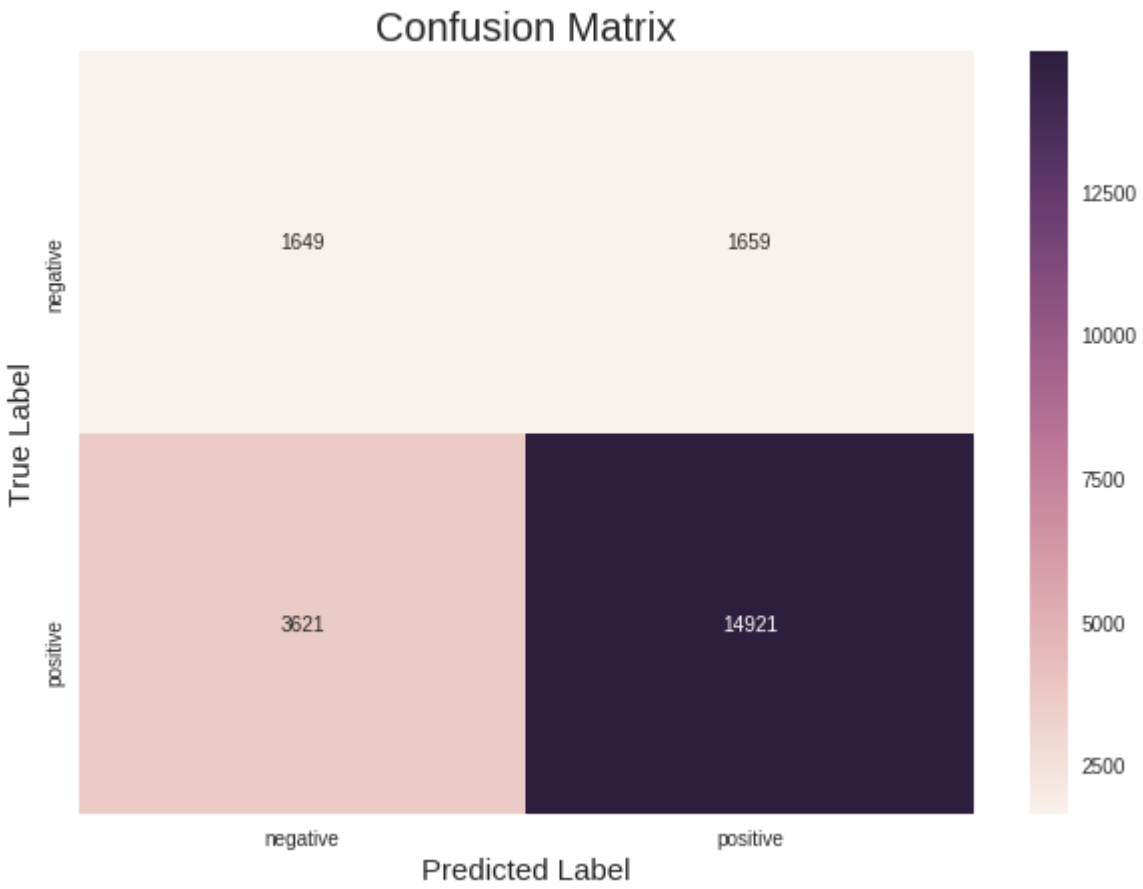
In [0]:
```
%time
model = XGBClassifier(max_depth = int(best_d), n_estimators = int(n_est),class_we
model.fit(X_tr, y_train)
```

```
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 7.87 µs
```

Out[97]:
```
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=500, 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)
```

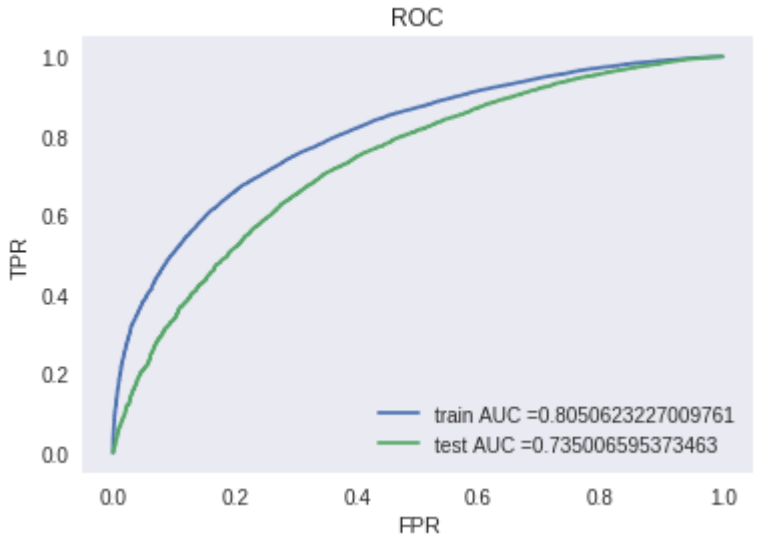
In [0]:
```
y_pred_te = model.predict(X_te)
```

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




```
True Positives : 14921
False Positives : 1659
True Negatives : 1649
False Negatives : 3621
```

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: 2.689415959517161
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[101]: 0.735006595373463

Summary

In [0]:
```
from prettytable import PrettyTable
import sys
sys.stdout.write("\033[1;30m")
x = PrettyTable()
x.field_names = ["model", "Optimal Depth","n_estimator", "AUC"]

x.add_row(["RF with truncated svd", 4, 100, 0.7350])
print(x)
```

```
+----------------------+---------------+-------------+-------+
|         model        | Optimal Depth | n_estimator |  AUC  |
+----------------------+---------------+-------------+-------+
| RF with truncated svd |       4       |     100     | 0.735 |
+----------------------+---------------+-------------+-------+
```

Steps Performed

- Do Text Preprocessing.
- Split the data into train, cv abd test dataset respectively.
- Since the data was imbalanced so perform the upsampling on train dataset.
- Apply TruncatedSVD on TfidfVectorizer of essay text ,using elbow method n_components was calculated as 70.
- Concatenate all the features and apply gbdt model to it.
- Plot auc curve on train and test data and also plot confusion matrix.