Apply TruncatedSVD

In [1]:

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

1) Splitting data into Train and cross validation(or test): Stratified Sampling

In [2]:

```
project_data = pd.read_csv("train_data.csv", nrows = 30000)
resource_data = pd.read_csv("resources.csv", nrows = 30000)
```

In [3]:

```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
```

```
Number of data points in train data (30000, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [4]:

```
# Let's check for any "null" or "missing" values
project_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 17 columns):
Unnamed: 0                                      30000 non-null int64
id                                              30000 non-null object
teacher_id                                      30000 non-null object
teacher_prefix                                  29999 non-null object
school_state                                    30000 non-null object
project_submitted_datetime                      30000 non-null object
project_grade_category                          30000 non-null object
project_subject_categories                      30000 non-null object
project_subject_subcategories                   30000 non-null object
project_title                                   30000 non-null object
project_essay_1                                 30000 non-null object
project_essay_2                                 30000 non-null object
project_essay_3                                 1013 non-null object
project_essay_4                                 1013 non-null object
project_resource_summary                        30000 non-null object
teacher_number_of_previously_posted_projects    30000 non-null int64
project_is_approved                             30000 non-null int64
dtypes: int64(3), object(14)
memory usage: 3.9+ MB
```

In [5]:

```
project_data['teacher_prefix'].isna().sum()
```

Out[5]:

```
1
```

In [6]:

```
# "teacher_prefix" seems to contain 2 "missing" values, let't use mode replacement strategy to fil
l those missing values
project_data['teacher_prefix'].mode()
```

Out[6]:

```
0    Mrs.
dtype: object
```

In [7]:

```
# Let's replace the missing values with "Mrs." , as it is the mode of the "teacher_prefix"
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna('Mrs.')
```

In [8]:

```
project_data['teacher_prefix'].value_counts()
```

Out[8]:

```
Mrs.       15683
Ms.        10779
Mr.         2895
Teacher      643
Name: teacher_prefix, dtype: int64
```

In [9]:

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

In [10]:

```
# Let's select only the selected features or columns, dropping "project_resource_summary" as it is optional
#
project_data.drop(['id','teacher_id','project_submitted_datetime','project_resource_summary'],axis=1, inplace=True)
project_data.columns
```

Out[10]:

```
Index(['Unnamed: 0', 'teacher_prefix', 'school_state',
       'project_grade_category', 'project_subject_categories',
       'project_subject_subcategories', 'project_title', 'project_essay_1',
       'project_essay_2', 'project_essay_3', 'project_essay_4',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'price', 'quantity'],
      dtype='object')
```

In [11]:

```
# Data seems to be highly imbalanced since the ratio of "class 1" to "class 0" is nearly 5.5
project_data['project_is_approved'].value_counts()
```

Out[11]:

```
1    25380
0     4620
Name: project_is_approved, dtype: int64
```

In [12]:

```
number_of_approved = project_data['project_is_approved'][project_data['project_is_approved'] == 1].count()
number_of_not_approved = project_data['project_is_approved'][project_data['project_is_approved'] == 0].count()

print("Ratio of Project approved to Not approved is:", number_of_approved/number_of_not_approved)
```

```
Ratio of Project approved to Not approved is: 5.4935064935064934
```

In [13]:

```
# 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 [14]:

```
project_data.head(2)
```

Out[14]:

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcateg
0	160221	Mrs.	IN	Grades PreK-2	Literacy & Language	ESL, Literacy
1	140945	Mr.	FL	Grades 6-8	History & Civics, Health &	Civics & Government, Team

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	Sports project_subject_categories	Sports project_subject_subcatego

In [15]:

```
# Let's drop the project essay columns from the dadaset now, as we have captured the essay text da
ta into single "essay" column
project_data.drop(['project_essay_1','project_essay_2','project_essay_3','project_essay_4'],axis=1
, inplace=True)
```

In [16]:

```
def remove_last_word(sentence):
    return " ".join([word for word in sentence.split()][:-1])
```

In [17]:

```
project_data['essay'] = project_data['essay'].apply(remove_last_word)
```

In [18]:

```
project_data['essay_and_title'] = project_data['essay'].map(str) + " " + project_data['project_titl
e']
```

In [19]:

```
y = project_data['project_is_approved'].values
X = project_data.drop(['project_is_approved'], axis=1)
X.head(1)
```

Out[19]:

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcatego
0	160221	Mrs.	IN	Grades PreK-2	Literacy & Language	ESL, Literacy

In [20]:

```
# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
```

2) Make Data Model Ready: encoding numerical, categorical features

In [21]:

```
def cleaning_text_data(list_text_feature,df,old_col_name,new_col_name):

    # 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
    feature_list = []
    for i in list_text_feature:
```

```
        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 & Sc
ience"=> "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 & Sc
ience"=>"Math&Science"
            temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
            temp = temp.replace('&','_') # we are replacing the & value into
        feature_list.append(temp.strip())

    df[new_col_name] = feature_list
    df.drop([old_col_name], axis=1, inplace=True)

    from collections import Counter
    my_counter = Counter()
    for word in df[new_col_name].values:
        my_counter.update(word.split())

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

In [22]:

```
def clean_project_grade(list_text_feature,df,old_col_name,new_col_name):

    # 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
    feature_list = []
    for i in list_text_feature:
        temp = i.split(' ')
        last_dig = temp[-1].split('-')
        fin = [temp[0]]
        fin.extend(last_dig)
        feature = '_'.join(fin)
        feature_list.append(feature.strip())

    df[new_col_name] = feature_list
    df.drop([old_col_name], axis=1, inplace=True)

    from collections import Counter
    my_counter = Counter()
    for word in df[new_col_name].values:
        my_counter.update(word.split())

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

2.1) Text Preprocessing: project_subject_categories

In [23]:

```
x_train_sorted_category_dict = cleaning_text_data(X_train['project_subject_categories'],X_train,'p
roject_subject_categories','clean_categories')
x_test_sorted_category_dict =
cleaning_text_data(X_test['project_subject_categories'],X_test,'project_subject_categories','clean_
categories')
```

2.2) Text Preprocessing : project_subject_subcategories

In [24]:

```
x_train_sorted_subcategories = cleaning_text_data(X_train['project_subject_subcategories'],X_train
,'project_subject_subcategories','clean_subcategories')
x_test_sorted_subcategories = cleaning_text_data(X_test['project_subject_subcategories'],X_test,'p
roject_subject_subcategories','clean_subcategories')
```

2.3) Text Preprocessing: project_grade_category

In [25]:

```
x_train_sorted_grade =
clean_project_grade(X_train['project_grade_category'],X_train,'project_grade_category','clean_grade
')
x_test_sorted_grade =
clean_project_grade(X_test['project_grade_category'],X_test,'project_grade_category','clean_grade'
)
```

2.4) Text Preprocessing (stowords): project_essay, project_title, essay_and_title_text

In [26]:

```
# 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 [27]:

```
# 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', "d
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 [28]:

```
# Combining all the above stundents
from tqdm import tqdm
def process_text(df,col_name):
    preprocessed_feature = []
    # tqdm is for printing the status bar
    for sentance in tqdm(df[col_name].values):
        sent = decontracted(sentance)
        sent = sent.replace('\\r', ' ')
        sent = sent.replace('\\"', ' ')
        sent = sent.replace('\\nan',' ')
        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_feature.append(sent.lower().strip())
    return preprocessed_feature
```

essay

In [29]:

```
x_train_essay_preprocessed = process_text(X_train,'essay')
x_test_essay_preprocessed = process_text(X_test,'essay')
```

```
100%|████████████████████████████████████████████████████████████████| 20100/20100 [00:
32<00:00, 612.38it/s]
100%|████████████████████████████████████████████████████████████████| 9900/9900
[00:14<00:00, 661.06it/s]
```

project_title

In [30]:

```
x_train_title_preprocessed = process_text(X_train,'project_title')
x_test_title_preprocessed = process_text(X_test,'project_title')
```

```
100%|████████████████████████████████████████████████████████████████| 20100/20100
[00:01<00:00, 15407.30it/s]
100%|████████████████████████████████████████████████████████████████| 9900/9900
[00:00<00:00, 10204.28it/s]
```

essay_and_title_text

In [31]:

```
x_train_essay_and_title_text_preprocessed = process_text(X_train,'essay_and_title')
x_test_essay_and_title_text_preprocessed = process_text(X_test, 'essay_and_title')
```

```
100%|████████████████████████████████████████████████████████████████| 20100/20100 [00:
32<00:00, 621.69it/s]
100%|████████████████████████████████████████████████████████████████| 9900/9900
[00:16<00:00, 609.23it/s]
```

2.5) Vectorizing Categorical Data

project_subject_categories (clean_categories)

In [32]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
def cat_vectorizer(X_train,df,col_name):
    vectorizer = CountVectorizer()
    vectorizer.fit(X_train[col_name].values)
    feature_one_hot = vectorizer.transform(df[col_name].values)
    print(vectorizer.get_feature_names())
    return feature_one_hot, vectorizer.get_feature_names()
```

In [33]:

```
x_train_cat_one_hot, x_train_cat_feat_list = cat_vectorizer(X_train,X_train,'clean_categories')
x_test_cat_one_hot, x_test_cat_feat_list = cat_vectorizer(X_train,X_test,'clean_categories')
```

```
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
```

In [34]:

```
# shape after categorical one hot encoding
print(x_train_cat_one_hot.shape)
print(x_test_cat_one_hot.shape)
```

```
(20100, 9)
(9900, 9)
```

project_subject_subcategory (clean_subcategory)

In [35]:

```
x_train_subcat_one_hot, x_train_subcat_feat_list =
cat_vectorizer(X_train,X_train,'clean_subcategories')
x_test_subcat_one_hot, x_test_subcat_feat_list =
cat_vectorizer(X_train,X_test,'clean_subcategories')
```

```
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience',
'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness',
'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'm
athematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socia
lsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience',
'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness',
'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'm
athematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socia
lsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
```

In [36]:

```
# shape after categorical one hot encoding
print(x_train_subcat_one_hot.shape)
print(x_test_subcat_one_hot.shape)
```

```
(20100, 30)
(9900, 30)
```

school_state

In [37]:

```
# we use count vectorizer to convert the values into one hot encoding
# CountVectorizer for "school_state"
x_train_state_one_hot, x_train_state_feat_list = cat_vectorizer(X_train,X_train,'school_state')
x_test_state_one_hot, x_test_state_feat_list = cat_vectorizer(X_train,X_test,'school_state')
```

```
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'k
s', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm',
'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv
', 'wy']
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'k
s', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm',
'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv
', 'wy']
```

In [38]:

```
# shape after categorical one hot encoding
print(x_train_state_one_hot.shape)
print(x_test_state_one_hot.shape)
```

```
(20100, 51)
(9900, 51)
```

teacher_prefix

In [39]:

```
# we use count vectorizer to convert the values into one hot encoding
# CountVectorizer for teacher_prefix
x_train_teacher_prefix_one_hot,x_train_teacher_prefix_feat_list = cat_vectorizer(X_train,X_train,'
teacher_prefix')
x_test_teacher_prefix_one_hot,x_test_teacher_prefix_feat_list =
cat_vectorizer(X_train,X_test,'teacher_prefix')
```

```
['mr', 'mrs', 'ms', 'teacher']
['mr', 'mrs', 'ms', 'teacher']
```

In [40]:

```
# shape after categorical one hot encoding
print(x_train_teacher_prefix_one_hot.shape)
print(x_test_teacher_prefix_one_hot.shape)
```

```
(20100, 4)
(9900, 4)
```

project_grade_category

In [41]:

```
# using count vectorizer for one-hot encoding of project_grade_category
x_train_grade_one_hot, x_train_grade_feat_list = cat_vectorizer(X_train,X_train,'clean_grade')
x_test_grade_one_hot, x_test_grade_feat_list = cat_vectorizer(X_train,X_test,'clean_grade')
```

```
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
```

In [42]:

```
# shape after categorical one hot encoding
print(x_train_grade_one_hot.shape)
print(x_test_grade_one_hot.shape)
```

```
(20100, 4)
(9900, 4)
```

Vectorizing Numerical Features

We have 2 numerical features left, "price" and "teacher_number_of_previously_posted_projects". Let's check for the "missing" or "NaN" values present in those numerical features and use "Mean Replacement" for "price" and "Mode Replacement" for "teacher_number_of_previously_posted_projects".

In [43]:

```
print("Total number of \"Missing\" Values present in X_train price:",X_train['price'].isna().sum()
)
print("Total number of \"Missing\" Values present in X_test price:",X_test['price'].isna().sum())
```

```
Total number of "Missing" Values present in X_train price: 19708
Total number of "Missing" Values present in X_test price: 9725
```

In [44]:

```
print("Total number of \"Missing\" Values present in X_train previous teacher number:",X_train['te
acher_number_of_previously_posted_projects'].isna().sum())
print("Total number of \"Missing\" Values present in X_test previous teacher number:",X_test['teac
her_number_of_previously_posted_projects'].isna().sum())
```

```
Total number of "Missing" Values present in X_train previous teacher number: 0
Total number of "Missing" Values present in X_test previous teacher number: 0
```

In [45]:

```
print("Total number of \"Missing\" Values present in X_train quantity:",X_train['quantity'].isna()
.sum())
print("Total number of \"Missing\" Values present in X_test quantity:",X_test['quantity'].isna().s
um())
```

```
Total number of "Missing" Values present in X_train quantity: 19708
Total number of "Missing" Values present in X_test quantity: 9725
```

In [46]:

```
X_train['price'].mean()
```

Out[46]:

```
260.6149489795919
```

In [56]:

```
X_train['price'] = X_train['price'].fillna(260.6149)
```

In [48]:

```
X_test['price'].mean()
```

Out[48]:

```
319.0982857142857
```

In [57]:

```
X_test['price'] = X_test['price'].fillna(319.0982)
```

In [58]:

```
print(X_train['quantity'].mean())
print(X_test['quantity'].mean())
```

```
18.75501032835794
18.46459469697114
```

In [59]:

```
X_train['quantity'] = X_train['quantity'].fillna(18.7550)
X_test['quantity'] = X_test['quantity'].fillna(18.4645)
```

In [60]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler
def scaler_function(df,col_name):

    scaler = StandardScaler()
    scaler.fit(df[col_name].values.reshape(-1,1)) # finding the mean and standard deviation of this
data

    # Now standardize the data with above maen and variance.
    print(f"Mean : {scaler.mean_[0]}, Standard deviation : {np.sqrt(scaler.var_[0])}")
    scaled = scaler.transform(df[col_name].values.reshape(-1, 1))
    return scaled
```

teacher_number_of_previously_posted_projects

In [61]:

```
x_train_teacher_number = scaler_function(X_train,'teacher_number_of_previously_posted_projects')
x_test_teacher_number = scaler_function(X_test,'teacher_number_of_previously_posted_projects')
```

```
Mean : 11.165572139303483, Standard deviation : 27.905393633254583
Mean : 11.305151515151515, Standard deviation : 27.978583326842035
```

price

In [62]:

```
x_train_price = scaler_function(X_train,'price')
x_test_price = scaler_function(X_test,'price')
```

```
Mean : 285.360206925373, Standard deviation : 34.407267638999805
Mean : 265.69891414141404, Standard deviation : 81.48858152608894
```

quantity

In [63]:

```
x_train_quantity = scaler_function(X_train,'quantity')
x_test_quantity = scaler_function(X_test,'quantity')
```

```
Mean : 18.755010328358214, Standard deviation : 3.2539585851405275
Mean : 18.464594696969694, Standard deviation : 4.406907178142603
```

Sentiment Score of essays

In [64]:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# import nltk
# nltk.download('vader_lexicon')
def compute_sentiment_score(df):
    score_list = []
    sid = SentimentIntensityAnalyzer()
    for essay in df['essay']:
        ss = sid.polarity_scores(essay)
        score_list.append(ss)
```

```
        score_list.append(ss)
    return score_list
# we can use these 4 things as features/attributes (neg, neu, pos, compound)
# neg: 0.0, neu: 0.753, pos: 0.247, compound: 0.93
```

```
C:\ProgramData\Anaconda3\lib\site-packages\nltk\twitter\__init__.py:20: UserWarning:

The twython library has not been installed. Some functionality from the twitter package will not b
e available.
```

In [65]:

```
x_train_score = compute_sentiment_score(X_train)
x_test_score = compute_sentiment_score(X_test)
```

In [66]:

```
def populate_list(score_dicts):

    neg_score = []
    neu_score = []
    pos_score = []
    compound_score = []
    for dict_ in score_dicts:
        neg_score.append(dict_['neg'])
        neu_score.append(dict_['neu'])
        pos_score.append(dict_['pos'])
        compound_score.append(dict_['compound'])
    return neg_score, neu_score, pos_score, compound_score
```

In [67]:

```
x_train_neg, x_train_neu, x_train_pos, x_train_compound = populate_list(x_train_score)
x_test_neg, x_test_neu, x_test_pos, x_test_compound = populate_list(x_test_score)
```

In [68]:

```
X_train['words_project_title'] = X_train['project_title'].apply(lambda x: len(x.split()))
X_train['words_essay'] = X_train['essay'].apply(lambda x: len(x.split()))
```

In [69]:

```
X_test['words_project_title'] = X_test['project_title'].apply(lambda x: len(x.split()))
X_test['words_essay'] = X_test['essay'].apply(lambda x: len(x.split()))
```

Let's join the all the sentiment scores to the respective dataframes

In [70]:

```
# for training set
X_train['neg'] = x_train_neg
X_train['neu'] = x_train_neu
X_train['pos'] = x_train_pos
X_train['compound'] = x_train_compound

# for testing set
X_test['neg'] = x_test_neg
X_test['neu'] = x_test_neu
X_test['pos'] = x_test_pos
X_test['compound'] = x_test_compound
```

Vectorizing newly added numerical features: words_project_title, words_essay

In [71]:

```
#Let's do for all the title words
x_train_words_title_scaled = scaler_function(X_train,'words_project_title')
x_test_words_title_scaled = scaler_function(X_test,'words_project_title')
# x_cv_words_title_scaled = scaler_function(X_cv,'words_project_title')

# let's do for all the essay words
x_train_words_essay_scaled = scaler_function(X_train,'words_essay')
x_test_words_essay_scaled = scaler_function(X_test,'words_essay')
# x_cv_words_essay_scaled = scaler_function(X_cv,'words_essay')
```

```
Mean : 5.153084577114428, Standard deviation : 2.095101592856299
Mean : 5.176868686868687, Standard deviation : 2.1058193282812567
Mean : 253.94990049751243, Standard deviation : 65.22893115933378
Mean : 253.7188888888889, Standard deviation : 64.79149360224594
```

Step 1: Selecting top 2k features from essay_and title_text

In [72]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
# We are considering only the words which appeared in at least 10 documents(rows or projects).
#https://web.stanford.edu/class/cs224n/assignments/a1_preview/exploring_word_vectors.html
#https://datascience.stackexchange.com/questions/40038/how-to-implement-word-to-word-co-occurence-matrix-in-python
#https://www.cnblogs.com/shiyublog/p/11136940.html

vectorizer = TfidfVectorizer(use_idf = True)
x_train_essay_title_tfidf = vectorizer.fit_transform(x_train_essay_and_title_text_preprocessed)
feature_names = vectorizer.get_feature_names()
idf_values = vectorizer.idf_
```

In [73]:

```
indices = np.argsort(idf_values)[::-1]
top_2k_words = [feature_names[i] for i in indices][:2000]
idf_val = [idf_values[i] for i in indices][:2000]
features_dict = {}
for word, idf_ in zip(top_2k_words, idf_val):
    features_dict[word] = idf_
```

In [74]:

```
cooccurrencematrix = np.zeros((2000,2000))
context_window = 5
```

In [75]:

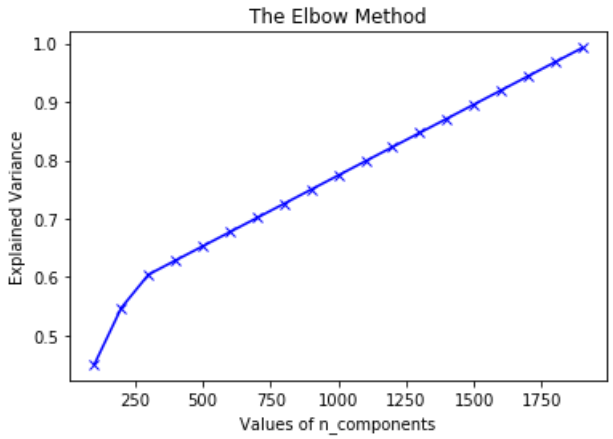
```
for sent in tqdm(x_train_essay_and_title_text_preprocessed):
    words = sent.split()
    for index,word in enumerate(words):
        if word in top_2k_words:
            for j in range(max(index - context_window,0), min(index + context_window, len(words) - 1
) + 1):
                if words[j] in top_2k_words:
                    cooccurrencematrix[top_2k_words.index(words[j]),top_2k_words.index(word)] += 1
                else:
                    continue
        else:
            continue
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 20100/20100 [01:
56<00:00, 173.23it/s]
```

In [76]:

```
from sklearn.decomposition import TruncatedSVD
variance_sum = []
components = [100 * x for x in range(1,20)]
for comp in components:
    svd = TruncatedSVD(n_components = comp)
```

```
    svd.fit(cooccurrencematrix)
    variance_sum.append(svd.explained_variance_ratio_.sum())

plt.plot(components,variance_sum,'bx-')
plt.xlabel('Values of n_components')
plt.ylabel('Explained Variance')
plt.title('The Elbow Method')
plt.show()
```



In [77]:

```
# from the graph, we can find that the optimum value of the n_components is 500
best_n_components = 500
```

In [78]:

```
final_matrix = cooccurrencematrix[:,:500]
```

In [79]:

```
final_matrix.shape
```

Out[79]:

```
(2000, 500)
```

Step 2. Vectorizing essays and title text

In [80]:

```
def vectorize_text(text,matrix):
    vectors = []
    index = 0
    for sent in tqdm(text):
        cur_vect = np.zeros((500))
        WORD_COUNT = 0
        words = sent.split()
        for word in words:
            if word in top_2k_words:
                index = top_2k_words.index(word)
            word_vect = matrix[index]
            cur_vect += word_vect
            WORD_COUNT += 1
        if WORD_COUNT:
            cur_vect /= WORD_COUNT
        else:
            cur_vect /= 1
        vectors.append(cur_vect)
    return vectors
```

In [81]:

```
x_train_essay_vectors = vectorize_text(x_train_essay_preprocessed,final_matrix)
```

```
100%|████████████████████████████████████████████████████████████████| 20100/20100 [02:
12<00:00, 151.68it/s]
```

In [82]:

```
x_test_essay_vectors = vectorize_text(x_test_essay_preprocessed, final_matrix)
```

```
100%|████████████████████████████████████████████████████████████████| 9900/9900
[01:02<00:00, 158.44it/s]
```

In [83]:

```
x_train_title_vectors = vectorize_text(x_train_title_preprocessed, final_matrix)
x_test_title_vectors = vectorize_text(x_test_title_preprocessed, final_matrix)
```

```
100%|████████████████████████████████████████████████████████████████| 20100/20100
[00:04<00:00, 4958.24it/s]
100%|████████████████████████████████████████████████████████████████| 9900/9900
[00:02<00:00, 4771.27it/s]
```

We have following features

In [84]:

```
# train_data
print("Train Data")
# categorical
print("Categorical")
print(x_train_cat_one_hot.shape)
print(x_train_subcat_one_hot.shape)
print(x_train_state_one_hot.shape)
print(x_train_teacher_prefix_one_hot.shape)
print(x_train_grade_one_hot.shape)
print("################################################")

# Numerical
print("Numerical")
print(x_train_teacher_number.shape)
print(x_train_price.shape)
print(x_train_quantity.shape)
print(x_train_words_title_scaled.shape)
print(x_train_words_essay_scaled.shape)
print("################################################")

# Sentiment score
print("Sentiment Score")
print(np.asarray(x_train_neg).shape)
print(np.asarray(x_train_neu).shape)
print(np.asarray(x_train_pos).shape)
print(np.asarray(x_train_compound).shape)
print("################################################")

# Truncated SVD text vectorized data
print("TruncatedSVD vectorized text data")
print(np.asarray(x_train_essay_vectors).shape)
print(np.asarray(x_train_title_vectors).shape)
print("################################################################################################
################")
```

```
Train Data
Categorical
(20100, 9)
(20100, 30)
(20100, 51)
(20100, 4)
(20100, 4)
################################################
```

```
##############################################
Numerical
(20100, 1)
(20100, 1)
(20100, 1)
(20100, 1)
(20100, 1)
##############################################
Sentiment Score
(20100,)
(20100,)
(20100,)
(20100,)
##############################################
TruncatedSVD vectorized text data
(20100, 500)
(20100, 500)
##########################################################################################################
#######
```

In [85]:

```
from scipy.sparse import hstack, csr_matrix
```

In [86]:

```
train_data =
hstack((x_train_cat_one_hot,x_train_subcat_one_hot,x_train_state_one_hot,x_train_teacher_prefix_one
_hot,\
                    x_train_grade_one_hot,x_train_teacher_number,x_train_price,x_train_quantity,x_t
rain_words_title_scaled,\
                    x_train_words_essay_scaled, X_train['neg'].values.reshape(-1,1) , X_train['neu'
.values.reshape(-1,1),\
                    X_train['pos'].values.reshape(-1,1), X_train['compound'].values.reshape(-1,1),x
train_essay_vectors,\
                    x_train_title_vectors)).tocsr()
```

In [87]:

```
train_data.shape
```

Out[87]:

```
(20100, 1107)
```

In [88]:

```
# train_data
print("Test Data")
# categorical
print("Categorical")
print(x_test_cat_one_hot.shape)
print(x_test_subcat_one_hot.shape)
print(x_test_state_one_hot.shape)
print(x_test_teacher_prefix_one_hot.shape)
print(x_test_grade_one_hot.shape)
print("##############################################")

# Numerical
print("Numerical")
print(x_test_teacher_number.shape)
print(x_test_price.shape)
print(x_test_quantity.shape)
print(x_test_words_title_scaled.shape)
print(x_test_words_essay_scaled.shape)
print("##############################################")

# Sentiment score
print("Sentiment Score")
print(np.asarray(x_test_neg).shape)
print(np.asarray(x_test_neu).shape)
print(np.asarray(x_test_pos).shape)
print(np.asarray(x_test_compound).shape)
```

```
print("############################################")

# Truncated SVD text vectorized data
print("TruncatedSVD vectorized text data")
print(np.asarray(x_test_essay_vectors).shape)
print(np.asarray(x_test_title_vectors).shape)
print("############################################################################################
###############")
```

```
Test Data
Categorical
(9900, 9)
(9900, 30)
(9900, 51)
(9900, 4)
(9900, 4)
############################################
Numerical
(9900, 1)
(9900, 1)
(9900, 1)
(9900, 1)
(9900, 1)
############################################
Sentiment Score
(9900,)
(9900,)
(9900,)
(9900,)
############################################
TruncatedSVD vectorized text data
(9900, 500)
(9900, 500)
############################################################################################
#######
```

In [89]:

```
test_data =
hstack((x_test_cat_one_hot,x_test_subcat_one_hot,x_test_state_one_hot,x_test_teacher_prefix_one_hot
,\
                    x_test_grade_one_hot,x_test_teacher_number,x_test_price,x_test_quantity,x_test_
words_title_scaled,\
                    x_test_words_essay_scaled, X_test['neg'].values.reshape(-1,1) , X_test['neu'].va
lues.reshape(-1,1),\
                    X_test['pos'].values.reshape(-1,1), X_test['compound'].values.reshape(-1,1),x_te
st_essay_vectors,\
                    x_test_title_vectors)).tocsr()
```

In [90]:

```
test_data.shape
```

Out[90]:

```
(9900, 1107)
```

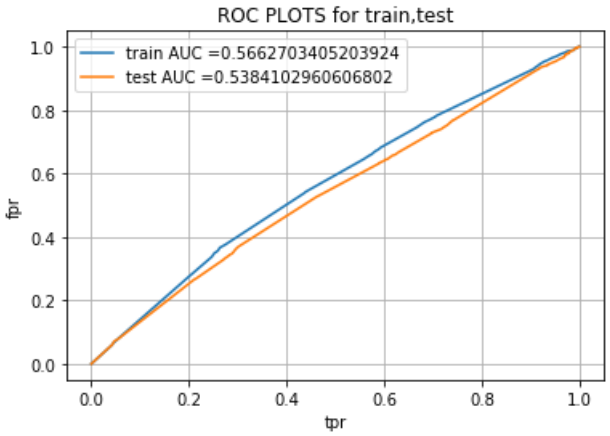
In [91]:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
import math
```

In [86]:

```
GB_ = GradientBoostingClassifier(n_estimators = 10, max_depth = 3)
GB_.fit(train_data, y_train)
```

```
y_train_pred = GB_.predict_proba(train_data)
y_test_pred =  GB_.predict_proba(test_data)

y_train_pred_prob = []
y_test_pred_prob = []

for index in range(len(y_train_pred)):
    y_train_pred_prob.append(y_train_pred[index][1])

for index in range(len(y_test_pred)):
    y_test_pred_prob.append(y_test_pred[index][1])


train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred_prob)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred_prob)



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("tpr")
plt.ylabel("fpr")
plt.title("ROC PLOTS for train,test")
plt.grid()
plt.show()
```



Without doing any hyper-parameter tuning we are able to achieve 0.5662 auc score for train_data and 0.5384 for test_data

In [90]:

```
GB_ = GradientBoostingClassifier()
parameters = {'n_estimators':[10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth':[2,3,4,5,6,7,8,9,1
0]}
clf = RandomizedSearchCV(GB_, parameters,n_iter = 8, scoring='roc_auc', return_train_score=True)
clf.fit(train_data[:5000], y_train[:5000])

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_n_estimators'])
```

In [91]:

```
results
```

Out[91]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_estimators	param_max_depth	params
0	0.483600	0.041824	0.004000	0.000552	10	8	{'n_estimators': 10, 'max_depth': 8}

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_estimators	param_max_depth	params
2	2.579581	0.052141	0.011306	0.002013	100	6	{'n_estimators': 100, 'max_depth': 6}
4	2.150332	0.098836	0.009605	0.002968	100	5	{'n_estimators': 100, 'max_depth': 5}
5	4.676806	0.281089	0.014794	0.000684	100	9	{'n_estimators': 100, 'max_depth': 9}
6	3.778265	0.121593	0.013095	0.000580	100	8	{'n_estimators': 100, 'max_depth': 8}
7	3.002494	0.248599	0.011725	0.003121	300	2	{'n_estimators': 300, 'max_depth': 2}
3	15.848312	0.372998	0.062102	0.003078	500	7	{'n_estimators': 500, 'max_depth': 7}
1	33.364646	0.900058	0.244238	0.090847	1000	7	{'n_estimators': 1000, 'max_depth': 7}

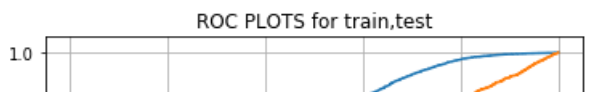
8 rows × 22 columns

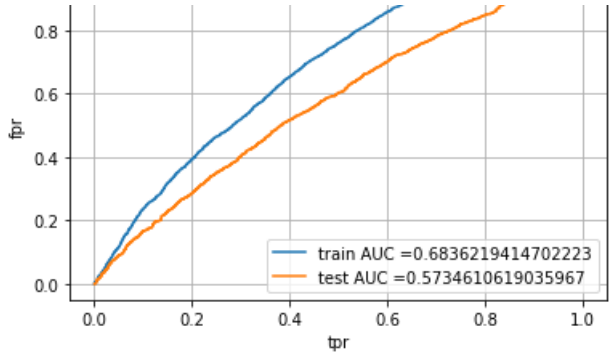
In [92]:

```
# from the table, we choose n_estimators as 10 maz_depth as 2
best_estimators = 100
best_depth = 8
```

In [94]:

```
GB_ = GradientBoostingClassifier(n_estimators = best_estimators, max_depth = best_depth)
GB_.fit(train_data, y_train)


y_train_pred = GB_.predict_proba(train_data)
y_test_pred =  GB_.predict_proba(test_data)

y_train_pred_prob = []
y_test_pred_prob = []

for index in range(len(y_train_pred)):
    y_train_pred_prob.append(y_train_pred[index][1])

for index in range(len(y_test_pred)):
    y_test_pred_prob.append(y_test_pred[index][1])


train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred_prob)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred_prob)



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("tpr")
plt.ylabel("fpr")
plt.title("ROC PLOTS for train,test")
plt.grid()
plt.show()
```





With hyper-parameter tuning we are able to slightly improve the train auc to 0.6836 and test auc to 0.5734. We can also try various values from table in order to observe the auc score for train and test.

Implemented the comments mentioned, I haven't put diagonal elements to zero as they wont affect the final result

In [97]:

```
Corpus =  ["abc def ijk pqr", "pqr klm opq","lmn pqr xyz abc def pqr abc"]
top_words = ["abc", "pqr", "def"]
window_size = 2

matrix = np.zeros((3,3))

for sent in tqdm(Corpus):
    words = sent.split()
    for index,word in enumerate(words):
        if word in top_words:
            for j in range(max(index - window_size,0), min(index + window_size, len(words) - 1) + 1)
:
                if words[j] in top_words:
                    matrix[top_words.index(words[j]),top_words.index(word)] += 1
                else:
                    continue
        else:
            continue
for i in range(matrix.shape[0]):
    for j in range(matrix.shape[1]):
        if i == j:
            matrix[i][j] = 0
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 3/3
[00:00<00:00, 3013.87it/s]
```

In [98]:

```
matrix
```

Out[98]:

```
array([[0., 3., 3.],
       [3., 0., 2.],
       [3., 2., 0.]])
```

In [93]:

```
import sys
import math

import numpy as np
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score

# you might need to install this one
import xgboost as xgb
```

```
class XGBoostClassifier():
    def __init__(self, num_boost_round=10, **params):
        self.clf = None
        self.num_boost_round = num_boost_round
        self.params = params
        self.params.update({'objective': 'multi:softprob', 'num_class':2})

    def fit(self, X, y, num_boost_round=None):
        num_boost_round = num_boost_round or self.num_boost_round
        self.label2num = {label: i for i, label in enumerate(sorted(set(y)))}
        dtrain = xgb.DMatrix(X, label=[self.label2num[label] for label in y])
        self.clf = xgb.train(params=self.params, dtrain=dtrain, num_boost_round=num_boost_round, ve
rbose_eval=1)

    def predict(self, X):
        num2label = {i: label for label, i in self.label2num.items()}
        Y = self.predict_proba(X)
        y = np.argmax(Y, axis=1)
        return np.array([num2label[i] for i in y])

    def predict_proba(self, X):
        dtest = xgb.DMatrix(X)
        return self.clf.predict(dtest)

    def score(self, X, y):
        Y = self.predict_proba(X)[:,1]
        return roc_auc_score(y, Y)

    def get_params(self, deep=True):
        return self.params

    def set_params(self, **params):
        if 'num_boost_round' in params:
            self.num_boost_round = params.pop('num_boost_round')
        if 'objective' in params:
            del params['objective']
        self.params.update(params)
        return self



parameters = {
    'num_boost_round': [100, 250, 500],
    'eta': [0.05, 0.1, 0.3],
    'max_depth': [6, 9, 12],
    'subsample': [0.9, 1.0],
    'colsample_bytree': [0.9, 1.0],
    'num_class':2
}

clf = XGBoostClassifier(parameters)

########################################################################
#               Change from here                                       #


# fit the xgboost on train_data
clf.fit(train_data, y_train, num_boost_round = 10)


# predict the xgboost on test_data
predicted_y = clf.predict(test_data)

# predict the xgboost on train_data
predicted_x = clf.predict(train_data)

# auc score for train_data
auc_train = clf.score(train_data, y_train)

# auc score for test_data
auc_test = clf.score(test_data, y_test)



########################################################################

# clf = GridSearchCV(clf, parameters)
```

```
# clf = GridSearchCV(clf, parameters)
# X = np.array([[1,2], [3,4], [2,1], [4,3], [1,0], [4,5]])
# Y = np.array([0, 1, 0, 1, 0, 1])
# clf.fit(X, Y)

# # print(clf.grid_scores_)
# best_parameters, score, _ = max(clf.grid_scores_, key=lambda x: x[1])
# print('score:', score)
# for param_name in sorted(best_parameters.keys()):
#     print("%s: %r" % (param_name, best_parameters[param_name]))
```

Let's calculate heatmaps for hyper-parameters Vs auc-score for xgboost

In [105]:

```
# Let's prepare for num_boost_round = [100,250.500]

# clf_1
parameter_1 = {'num_boost_round':100}
clf_1 = XGBoostClassifier(parameter_1)
clf_1.fit(train_data, y_train, num_boost_round = 100)
auc_train_nbr_1 = clf_1.score(train_data, y_train)
auc_test_nbr_1 = clf_1.score(test_data, y_test)

# clf_2
parameter_2 = {'num_boost_round':250}
clf_2 = XGBoostClassifier(parameter_2)
clf_2.fit(train_data, y_train, num_boost_round = 250)
auc_train_nbr_2 = clf_2.score(train_data, y_train)
auc_test_nbr_2 = clf_2.score(test_data, y_test)

# clf_3
parameter_3 = {'num_boost_round':500}
clf_3 = XGBoostClassifier(parameter_3)
clf_3.fit(train_data, y_train, num_boost_round = 500)
auc_train_nbr_3 = clf_3.score(train_data, y_train)
auc_test_nbr_3 = clf_3.score(test_data, y_test)
```

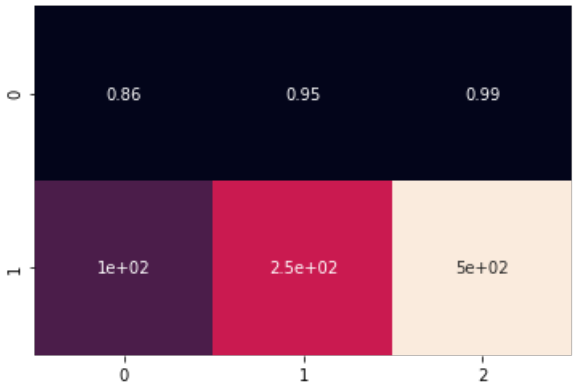
In [106]:

```
print("Train Auc score Vs Num_boost_round Heatmap")
auc_nbr_train = [[auc_train_nbr_1, auc_train_nbr_2, auc_train_nbr_3], [100,0.05,6]]
sns.heatmap(auc_nbr_train, annot = True,cbar=False)
```

```
Train Auc score Vs Num_boost_round Heatmap
```

Out[106]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x258e8e47160>
```



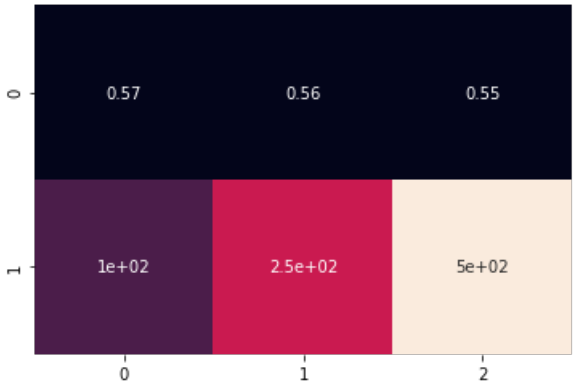
In [107]:

```
print("Test Auc score Vs Num_boost_round Heatmap")
auc_nbr_test = [[auc_test_nbr_1, auc_test_nbr_2, auc_test_nbr_3], [100,250,500]]
sns.heatmap(auc_nbr_test, annot = True,cbar=False)
```

Test Auc score Vs Num_boost_round Heatmap

Out[107]:

<matplotlib.axes._subplots.AxesSubplot at 0x258e8f05898>



In [110]:

```
# Let's prepare for eta = [0.05, 0.1, 0.3]

# clf_1
parameter_1 = {'eta':0.05}
clf_1 = XGBoostClassifier(parameter_1)
clf_1.fit(train_data, y_train,  num_boost_round = 100)
auc_train_eta_1 = clf_1.score(train_data, y_train)
auc_test_eta_1 = clf_1.score(test_data, y_test)

# clf_2
parameter_2 = {'eta':0.1}
clf_2 = XGBoostClassifier(parameter_2)
clf_2.fit(train_data, y_train, num_boost_round = 100)
auc_train_eta_2 = clf_2.score(train_data, y_train)
auc_test_eta_2 = clf_2.score(test_data, y_test)

# clf_3
parameter_3 = {'eta':0.3}
clf_3 = XGBoostClassifier(parameter_3)
clf_3.fit(train_data, y_train, num_boost_round = 100)
auc_train_eta_3 = clf_3.score(train_data, y_train)
auc_test_eta_3 = clf_3.score(test_data, y_test)
```

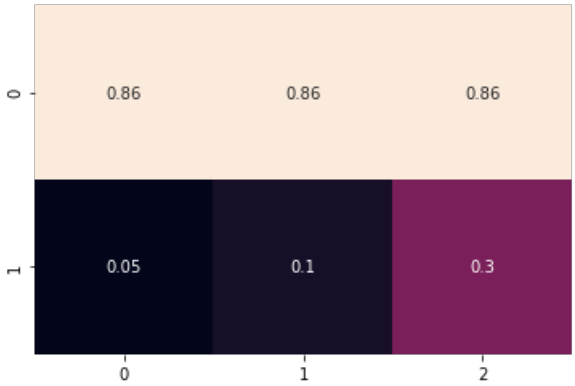
In [114]:

```
print("Train Auc score Vs eta Heatmap")
auc_eta_train = [[auc_train_eta_1, auc_train_eta_2, auc_train_eta_3], [0.05,0.1,0.3]]
sns.heatmap(auc_eta_train, annot = True,cbar=False)
```

Train Auc score Vs eta Heatmap

Out[114]:

<matplotlib.axes._subplots.AxesSubplot at 0x258e8f7a668>



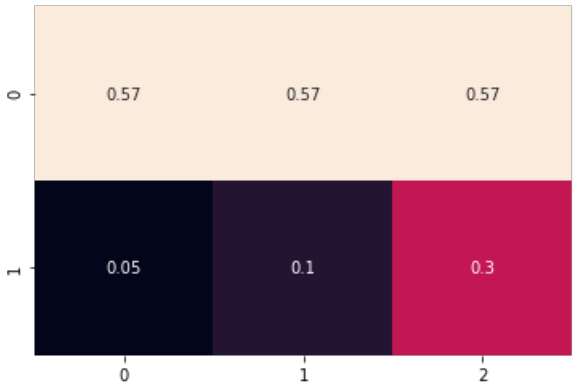
In [115]:

```
print("Test Auc score Vs Num_boost_round Heatmap")
auc_eta_test = [[auc_test_eta_1, auc_test_eta_2, auc_test_eta_3], [0.05,0.1,0.3]]
sns.heatmap(auc_eta_test, annot = True,cbar=False)
```

```
Test Auc score Vs Num_boost_round Heatmap
```

Out[115]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x258e8fb3ba8>
```



In [120]:

```
# Let's prepare for max_depth = [6,9,12]

# clf_1
parameter_1 = {'max_depth':6}
clf_1 = XGBoostClassifier(parameter_1)
clf_1.fit(train_data, y_train,  num_boost_round = 100)
auc_train_md_1 = clf_1.score(train_data, y_train)
auc_test_md_1 = clf_1.score(test_data, y_test)

# clf_2
parameter_2 = {'max_depth':9}
clf_2 = XGBoostClassifier(parameter_2)
clf_2.fit(train_data, y_train, num_boost_round = 100)
auc_train_md_2 = clf_2.score(train_data, y_train)
auc_test_md_2 = clf_2.score(test_data, y_test)

# clf_3
parameter_3 = {'max_depth':12}
clf_3 = XGBoostClassifier(parameter_3)
clf_3.fit(train_data, y_train, num_boost_round = 100)
auc_train_md_3 = clf_3.score(train_data, y_train)
auc_test_md_3 = clf_3.score(test_data, y_test)
```

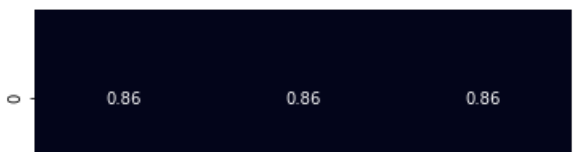
In [121]:

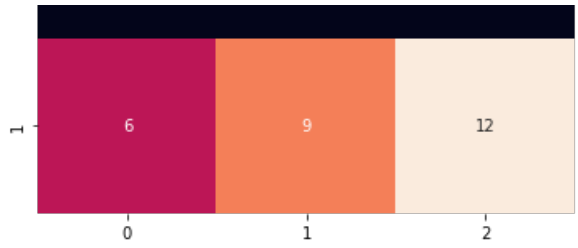
```
print("Train Auc score Vs eta Heatmap")
auc_md_train = [[auc_train_md_1, auc_train_md_2, auc_train_md_3], [6,9,12]]
sns.heatmap(auc_md_train, annot = True,cbar=False)
```

```
Train Auc score Vs eta Heatmap
```

Out[121]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x258e9002b70>
```





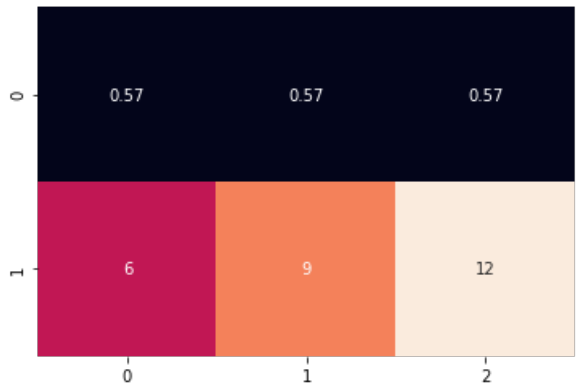
In [122]:

```
print("Test Auc score Vs Num_boost_round Heatmap")
auc_md_test = [[auc_test_md_1, auc_test_md_2, auc_test_md_3], [6,9,12]]
sns.heatmap(auc_md_test, annot = True,cbar=False)
```

```
Test Auc score Vs Num_boost_round Heatmap
```

Out[122]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x258e9044f98>
```



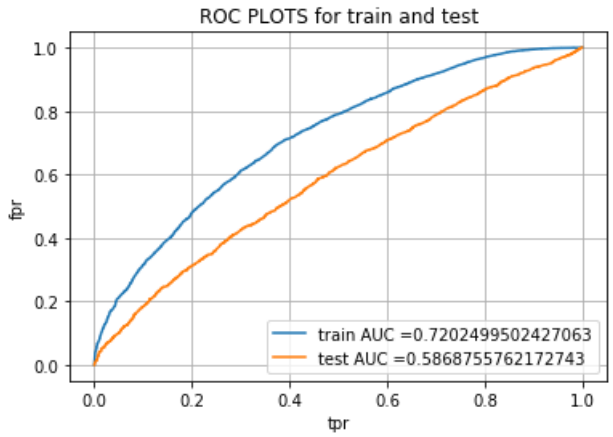
In [122]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (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))
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [121]:

```
y_train_pred = clf.predict_proba(train_data)
y_test_pred =  clf.predict_proba(test_data)

y_train_pred_prob = []
y_test_pred_prob = []

for index in range(len(y_train_pred)):
    y_train_pred_prob.append(y_train_pred[index][1])

for index in range(len(y_test_pred)):
    y_test_pred_prob.append(y_test_pred[index][1])


train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred_prob)
```

```
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred_prob)

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("tpr")
plt.ylabel("fpr")
plt.title("ROC PLOTS for train and test")
plt.grid()
plt.show()
```



In [115]:

```
# Let's plot confusion matrix for Xgboost
```

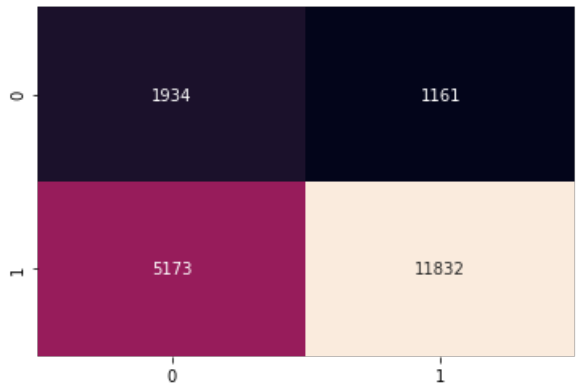
In [123]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
sns.heatmap(confusion_matrix(y_train, predict_with_best_t(y_train_pred_prob, best_t)),annot = True,
fmt = "d", cbar=False)
```

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

the maximum value of tpr*(1-fpr) 0.43478779167393034 for threshold 0.825
Train confusion matrix
```

Out[123]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x17525c410f0>
```



In [124]:

```
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, predict_with_best_t(y_test_pred_prob, best_t)), annot = True,
fmt = "d", cbar=False)
```

Test confusion matrix

Out[124]:

<matplotlib.axes._subplots.AxesSubplot at 0x17525da1a20>