DonorsChoose

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

import time
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 [2]:

```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

In [3]:

```
print(len(project_data))
print(len(resource_data))
```

```
109248
1541272
```

In [4]:

```
from sklearn.utils import resample
```

In [5]:

```
project_data=resample(project_data,n_samples=50000)
```

In [6]:

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


#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
project_data.drop('project_submitted_datetime', axis=1, inplace=True)
project_data.sort_values(by=['Date'], inplace=True)


# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084039
project_data = project_data[cols]

print(cols)
project_data.head(2)
```

```
['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state', 'Date',
'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']
```

Out[6]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_categ
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	2016-04-27 00:46:53	Grades PreK-2
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5

In [7]:

```
len(project_data['project_is_approved'])
```

Out[7]:

```
50000
```

In [8]:

```
filtered = project_data.loc[project_data['project_is_approved'] == 1]
```

In [9]:

```
print(len(filtered))
```

```
42301
```

In [10]:

```
# https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexes-for-all-groups-in
-one-step
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()

# join two dataframes in python:
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

In [11]:

```
#project_data = project_data.sample(frac=0.5)
```

Preprocessing data

1.2 preprocessing of `project_subject_categories`

In [12]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # 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]))
```

In [13]:

```
preprocessed_grade=project_data['project_grade_category']
```

In [14]:

```
new=[i.replace("-","_") for i in preprocessed_grade]
new=[i.replace(" ","_") for i in new]
```

In [15]:

```
project_data['preprocessed_grade']=new
```

In [16]:

```
print(project_data['preprocessed_grade'])
```

```
0        Grades_PreK_2
1           Grades_3_5
2           Grades_3_5
3          Grades_9_12
4        Grades_PreK_2
5        Grades_PreK_2
6           Grades_3_5
7           Grades_3_5
```

```
8           Grades_3_5
9        Grades_PreK_2
10       Grades_PreK_2
11       Grades_PreK_2
12       Grades_PreK_2
13       Grades_PreK_2
14         Grades_9_12
15         Grades_9_12
16          Grades_3_5
17          Grades_3_5
18         Grades_9_12
19          Grades_6_8
20          Grades_3_5
21          Grades_3_5
22          Grades_3_5
23         Grades_9_12
24       Grades_PreK_2
25       Grades_PreK_2
26          Grades_6_8
27         Grades_9_12
28          Grades_3_5
29          Grades_3_5
             ...
49970    Grades_PreK_2
49971    Grades_PreK_2
49972       Grades_3_5
49973       Grades_3_5
49974       Grades_3_5
49975    Grades_PreK_2
49976       Grades_6_8
49977       Grades_6_8
49978    Grades_PreK_2
49979    Grades_PreK_2
49980       Grades_3_5
49981       Grades_3_5
49982       Grades_3_5
49983    Grades_PreK_2
49984       Grades_6_8
49985      Grades_9_12
49986    Grades_PreK_2
49987    Grades_PreK_2
49988    Grades_PreK_2
49989    Grades_PreK_2
49990       Grades_3_5
49991       Grades_6_8
49992      Grades_9_12
49993      Grades_9_12
49994       Grades_6_8
49995       Grades_3_5
49996      Grades_9_12
49997    Grades_PreK_2
49998    Grades_PreK_2
49999       Grades_3_5
Name: preprocessed_grade, Length: 50000, dtype: object
```

In [17]:

```
print(project_data['clean_categories'].unique())
```

```
['Literacy_Language' 'Math_Science History_Civics'
 'AppliedLearning Music_Arts' 'Math_Science AppliedLearning'
 'Math_Science' 'AppliedLearning Health_Sports'
 'AppliedLearning Literacy_Language' 'Health_Sports'
 'Literacy_Language SpecialNeeds' 'Math_Science SpecialNeeds'
 'Literacy_Language Math_Science' 'Math_Science Literacy_Language'
 'AppliedLearning History_Civics' 'AppliedLearning'
 'Health_Sports Literacy_Language' 'Literacy_Language Music_Arts'
 'Math_Science Music_Arts' 'SpecialNeeds Health_Sports' 'Music_Arts'
 'History_Civics Literacy_Language' 'Health_Sports SpecialNeeds'
 'SpecialNeeds' 'SpecialNeeds Music_Arts' 'AppliedLearning SpecialNeeds'
 'Health_Sports AppliedLearning' 'AppliedLearning Math_Science'
 'History_Civics' 'History_Civics Music_Arts'
 'Literacy_Language History_Civics' 'Health_Sports Music_Arts'
 'History_Civics SpecialNeeds' 'Math_Science Health_Sports'
 'Literacy_Language AppliedLearning' 'Music_Arts History_Civics'
 'History_Civics AppliedLearning' 'History_Civics Math_Science'
 'History_Civics AppliedLearning' 'History_Civics Math_Science'
 'Health_Sports History_Civics' 'Health_Sports Math_Science'
 'Music_Arts Health_Sports' 'Music_Arts SpecialNeeds'
 'Literacy_Language Health_Sports' 'Music_Arts AppliedLearning'
 'SpecialNeeds Warmth Care_Hunger' 'Health_Sports Warmth Care_Hunger'
 'Warmth Care_Hunger' 'Literacy_Language Warmth Care_Hunger'
 'Math_Science Warmth Care_Hunger' 'Music_Arts Warmth Care_Hunger'
 'History_Civics Health_Sports' 'AppliedLearning Warmth Care_Hunger']
```

1.3 preprocessing of `project_subject_subcategories`

In [18]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039

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

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

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

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

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

1.4 Preprocessing of project_grade_category

1.3 Text preprocessing

In [19]:

```
# 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 [20]:

```
# 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
    # 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 [21]:

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

In [22]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['essay'].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_essays.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████████████| 50000/50000
[00:43<00:00, 1148.62it/s]
```

1.4 Preprocessing of `project_title`

In [23]:

```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_titles = []
# tqdm is for printing the status bar
for sentence in tqdm(project_data['project_title'].values):
    sent = decontracted(sentence)
    sent = decontracted(sentence)
    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.lower() for e in sent.split() if e not in stopwords)
    preprocessed_titles.append(sent.lower().strip())
```

```
100%|███████████████████████████████████████████████████████████████████████████| 50000/50000
[00:02<00:00, 24713.97it/s]
```

In [24]:

```
#Adding processed columns at place of original columns
project_data['clean_essays'] = preprocessed_essays
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
project_data.drop(['project_essay_4'], axis=1, inplace=True)
```

In [25]:

```
project_data['project_resource_summary']
preprocessed_resource_summary=[]
for sentence in tqdm(project_data['project_resource_summary'].values):
    sent = decontracted(sentence)
    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.lower() for e in sent.split() if e not in stopwords)
    preprocessed_resource_summary.append(sent.lower().strip())
```

```
100%|███████████████████████████████████████████████████████████████████████████| 50000/50000
[00:04<00:00, 11470.32it/s]
```

In [26]:

```
project_data['clean_resource_summary'] = preprocessed_resource_summary
```

In [27]:

```
project_data['clean_titles'] = preprocessed_titles
```

In [28]:

```
# we cannot remove rows where teacher prefix is not available therefore we are replacing 'nan' val
ue with
# 'null'(string)
#https://stackoverflow.com/questions/42224700/attributeerror-float-object-has-no-attribute-split
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna('null')
```

In [29]:

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

Out[29]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category
0	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	2016-04-27 00:46:53	Grades PreK-2

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category
1	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5

In [30]:

```
filtered_negative = project_data.loc[project_data['project_is_approved'] == 0]
print(len(filtered_negative))
#print(len(filtered_positive))
filtered_positive = project_data.loc[project_data['project_is_approved'] == 1]
sample_positive = filtered_positive.take(np.random.permutation(len(filtered_positive))[:50000])
```

```
7699
```

In [31]:

```
print(len(filtered_positive))
print(len(sample_positive))
```

```
42301
42301
```

In [32]:

```
project_data = pd.concat([filtered_negative, sample_positive]).sort_index(kind='merge')
```

In [33]:

```
project_data.count()
```

Out[33]:

```
Unnamed: 0                                      50000
id                                              50000
teacher_id                                      50000
teacher_prefix                                  50000
school_state                                    50000
Date                                            50000
project_grade_category                          50000
project_title                                   50000
project_resource_summary                        50000
teacher_number_of_previously_posted_projects    50000
project_is_approved                             50000
price                                           50000
quantity                                        50000
clean_categories                                50000
preprocessed_grade                              50000
clean_subcategories                             50000
essay                                           50000
clean_essays                                    50000
clean_resource_summary                          50000
clean_titles                                    50000
dtype: int64
```

So far we have preprocessed the data. Next is to split and vectorize data for BoW,TFIDF,Avg W2Vec and TFIDF weighted W2Vec

1.Splitting data

In [34]:

```
Project_data_new =project_data.iloc[:20000,:]
```

In [35]:

```
(Project_data_new).shape
```

Out[35]:

```
(20000, 20)
```

In [36]:

```
y = project_data['project_is_approved'].values
y_new = Project_data_new['project_is_approved'].values
project_data.drop(['project_is_approved'], axis=1, inplace=True)
Project_data_new.drop(['project_is_approved'], axis=1, inplace=True)
X = project_data
X_new = Project_data_new
```

In [37]:

```
# 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)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
X_train_new, X_test_new, y_train_new, y_test_new = train_test_split(X_new, y_new, test_size=0.33, s
tratify=y_new)
X_train_new, X_cv_new, y_train_new, y_cv_new = train_test_split(X_train_new, y_train_new, test_size
=0.33, stratify=y_train_new)
```

In [38]:

```
x = np.count_nonzero(y_test)
print(len(y_test) - x)
```

```
2541
```

In [39]:

```
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)
```

```
(22445, 19) (22445,)
(11055, 19) (11055,)
(16500, 19) (16500,)
====================================================================================================
```

In [40]:

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

2.Vectorizing data

Response Coding for Categorical features

In [157]:

```
print(X_train['teacher_prefix'].unique())
print(X_test['teacher_prefix'].unique())
#print(X_train['clean_categories'].unique())
#print(X_test['clean_categories'].unique())
list_notpresent_cat = [i for i in X_test['clean_categories'].unique() if i not in X_train['clean_ca
tegories'].unique() ]
```

```
print(list_notpresent_cat)
#print(X_train['clean_subcategories'].unique())
#print(X_test['clean_subcategories'].unique())
list_notpresent_sub = [i for i in X_test['clean_subcategories'].unique() if i not in X_train['clean
_subcategories'].unique() ]
print(list_notpresent_sub)
#print(X_train['school_state'].unique())
#print(X_test['school_state'].unique())
list_notpresent_state = [i for i in X_test['school_state'].unique() if i not in
X_train['school_state'].unique() ]
print(list_notpresent_state)

print(X_train['project_grade_category'].unique())
print(X_test['project_grade_category'].unique())
```

```
['Mrs.' 'Ms.' 'Teacher' 'Mr.' 'Dr.' 'null']
['Ms.' 'Mrs.' 'Mr.' 'Teacher' 'Dr.']
['Music_Arts Warmth Care_Hunger']
['CommunityService Literacy', 'ParentInvolvement Warmth Care_Hunger', 'FinancialLiteracy
ParentInvolvement', 'Gym_Fitness PerformingArts', 'SocialSciences TeamSports', 'VisualArts Warmth
Care_Hunger', 'CommunityService PerformingArts', 'College_CareerPrep NutritionEducation',
'Health_LifeScience PerformingArts', 'CommunityService Other', 'ForeignLanguages PerformingArts',
'Extracurricular Literature_Writing', 'EarlyDevelopment Warmth Care_Hunger', 'ParentInvolvement
SocialSciences', 'Other SocialSciences', 'History_Geography PerformingArts', 'Health_Wellness Perf
ormingArts', 'Economics Other', 'Extracurricular NutritionEducation', 'Extracurricular
History_Geography', 'FinancialLiteracy Health_LifeScience', 'Music SocialSciences',
'ParentInvolvement PerformingArts']
[]
['Grades PreK-2' 'Grades 9-12' 'Grades 3-5' 'Grades 6-8']
['Grades PreK-2' 'Grades 3-5' 'Grades 9-12' 'Grades 6-8']
```

In [83]:

```
from sklearn.cross_validation import cross_val_score
def findresponse_coding(category):
    Prob_pos=[]
    Prob_neg=[]
    unique_arr=(X_train[category].unique())
    for i in unique_arr:
        X_category = X_train[ X_train[category]==i]
        X_pos = X_category[X_category['y_train'] ==1]
        prob = len(X_pos)/len(X_category)
        Prob_pos.append(prob)
        Prob_neg.append(1 - prob)
    dict_pos={}
    dict_neg={}
    unique_arr = list(unique_arr)
    for i in range(len(unique_arr)):
        dict_pos [unique_arr[i]] = Prob_pos[i]
        dict_neg [unique_arr[i]] = Prob_neg[i]
    list_pos=[]
    list_neg=[]
    for i in X_train[category]:
        list_pos.append(dict_pos[i])
        list_neg.append(dict_neg[i])
    X_train[category + '_pos'] = list_pos
    X_train[category + '_neg'] = list_neg
```

In [168]:

```
def response_for_test(category , list_notpresent):
    list_pos=[]
    list_neg =[]
    list_train_pos = list(X_train[category + '_pos'])
    list_train_neg = list(X_train[category + '_neg'])
    for index,i in enumerate(list(X_test[category])):
        if i in list_notpresent:
            list_pos.append (0.5)
            list_neg.append (0.5)
        else:
            list_pos .append (list_train_pos[index])
            list_neg .append (list_train_neg[index])
            list_neg .append (list_train_neg[index])
    X_test[category + '_pos'] = list_pos
    X_test[category + '_neg'] = list_neg
```

In [169]:

```
findresponse_coding('teacher_prefix')
findresponse_coding('clean_categories')
findresponse_coding('school_state')
findresponse_coding('clean_subcategories')
findresponse_coding('project_grade_category')
```

In [187]:

```
response_for_test('clean_categories' , list_notpresent_cat)
response_for_test('clean_subcategories' , list_notpresent_sub)
response_for_test('school_state' , list_notpresent_state)
response_for_test('teacher_prefix' , list_notpresent_state)
response_for_test('project_grade_category' , list_notpresent_state)
```

In [188]:

```
X_test.columns
```

Out[188]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', 'project_title',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'price', 'quantity',
       'clean_categories', 'preprocessed_grade', 'clean_subcategories',
       'essay', 'clean_essays', 'clean_resource_summary', 'clean_titles',
       'clean_categories_pos', 'clean_categories_neg',
       'clean_subcategories_pos', 'clean_subcategories_neg',
       'school_state_pos', 'school_state_neg', 'teacher_prefix_pos',
       'teacher_prefix_neg', 'project_grade_category_pos',
       'project_grade_category_neg'],
      dtype='object')
```

In [43]:

```
X_train.columns
```

Out[43]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', 'project_title',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'price', 'quantity',
       'clean_categories', 'preprocessed_grade', 'clean_subcategories',
       'essay', 'clean_essays', 'clean_resource_summary', 'clean_titles',
       'y_train', 'teacher_prefix_pos', 'teacher_prefix_neg',
       'clean_categories_pos', 'clean_categories_neg', 'school_state_pos',
       'school_state_neg', 'clean_subcategories_pos',
       'clean_subcategories_neg', 'project_grade_category_pos',
       'project_grade_category_neg'],
      dtype='object')
```

2.5 Normalizing the numerical features: Price

In [44]:

```
X_train.head(2)
```

Out[44]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_cat

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_cate
48370	33684	p193053	b17d1dc276ae0898c4df4fac3638a613	Mrs.	MD	2017-04-10 20:23:59	Grades PreK-2
30549	75509	p258509	a2aae781124890b31083fa0833509b9a	Ms.	NJ	2016-11-15 13:27:54	Grades PreK-2

2 rows × 30 columns

In [45]:

```
from sklearn.preprocessing import StandardScaler
standard_vec = StandardScaler(with_mean = False)
# 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.
standard_vec.fit(X_train['price'].values.reshape(-1,1))

X_train_price_std = standard_vec.transform(X_train['price'].values.reshape(-1,1))
X_cv_price_std = standard_vec.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_std = standard_vec.transform(X_test['price'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_price_std.shape, y_train.shape)
print(X_cv_price_std.shape, y_cv.shape)
print(X_test_price_std.shape, y_test.shape)



from sklearn.preprocessing import StandardScaler
standard_vec = StandardScaler(with_mean = False)
# 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.
standard_vec.fit(X_train_new['price'].values.reshape(-1,1))

X_train_price_std_new = standard_vec.transform(X_train_new['price'].values.reshape(-1,1))
X_cv_price_std_new = standard_vec.transform(X_cv_new['price'].values.reshape(-1,1))
X_test_price_std_new = standard_vec.transform(X_test_new['price'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_price_std_new.shape, y_train_new.shape)
print(X_cv_price_std_new.shape, y_cv_new.shape)
print(X_test_price_std_new.shape, y_test_new.shape)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
After vectorizations
(8978, 1) (8978,)
(4422, 1) (4422,)
(6600, 1) (6600,)
```

2.6 Vectorizing numerical features: teacher_number_of_previously _posted_projects"

In [46]:

```
from sklearn.preprocessing import StandardScaler
standard_vec = StandardScaler(with_mean = False)
# 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.
standard_vec.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
```

```
X_train_projects_std =
standard_vec.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1
))
X_cv_projects_std = standard_vec.transform(X_cv['teacher_number_of_previously_posted_projects'].va
lues.reshape(-1,1))
X_test_projects_std = standard_vec.transform(X_test['teacher_number_of_previously_posted_projects'
].values.reshape(-1,1))

print("After vectorizations")
print(X_train_projects_std.shape, y_train.shape)
print(X_cv_projects_std.shape, y_cv.shape)
print(X_test_projects_std.shape, y_test.shape)
print("="*100)

X_train_projects_std_new =
standard_vec.transform(X_train_new['teacher_number_of_previously_posted_projects'].values.reshape(
-1,1))
X_cv_projects_std_new =
standard_vec.transform(X_cv_new['teacher_number_of_previously_posted_projects'].values.reshape(-1,
1))
X_test_projects_std_new =
standard_vec.transform(X_test_new['teacher_number_of_previously_posted_projects'].values.reshape(-
1,1))

print("After vectorizations")
print(X_train_projects_std_new.shape, y_train_new.shape)
print(X_cv_projects_std_new.shape, y_cv_new.shape)
print(X_test_projects_std_new.shape, y_test_new.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================

After vectorizations
(8978, 1) (8978,)
(4422, 1) (4422,)
(6600, 1) (6600,)
====================================================================================================
```

In [47]:

```
from sklearn.preprocessing import StandardScaler
standard_vec = StandardScaler(with_mean = False)
# 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.
standard_vec.fit(X_train['quantity'].values.reshape(-1,1))

X_train_qty_std = standard_vec.transform(X_train['quantity'].values.reshape(-1,1))
X_cv_qty_std = standard_vec.transform(X_cv['quantity'].values.reshape(-1,1))
X_test_qty_std = standard_vec.transform(X_test['quantity'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_qty_std.shape, y_train.shape)
print(X_cv_qty_std.shape, y_cv.shape)
print(X_test_qty_std.shape, y_test.shape)
print("="*100)

standard_vec.fit(X_train_new['quantity'].values.reshape(-1,1))

X_train_qty_std_new = standard_vec.transform(X_train_new['quantity'].values.reshape(-1,1))
X_cv_qty_std_new = standard_vec.transform(X_cv_new['quantity'].values.reshape(-1,1))
X_test_qty_std_new = standard_vec.transform(X_test_new['quantity'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_qty_std_new.shape, y_train_new.shape)
print(X_cv_qty_std_new.shape, y_cv_new.shape)
print(X_test_qty_std_new.shape, y_test_new.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================

After vectorizations
(8978, 1) (8978,)
(4422, 1) (4422,)
(6600, 1) (6600,)
====================================================================================================
```

In [48]:

```
def optimal_hyper(X,y,parameters):
    k_scores = []
    for i in parameters['n_estimators']:
        for j in parameters['max_depth']:
            gbt = GradientBoostingClassifier(n_estimators=i , max_depth =j)
    # 3. obtain cross_val_score for KNeighborsClassifier with k neighbours
            scores = cross_val_score(gbt, X, y, cv=2, scoring='roc_auc',n_jobs=-1)
            #print(scores)
    # 4. append mean of scores for k neighbors to k_scores list
            k_scores.append(scores.mean())
        #print(k_scores)
    return k_scores
```

TFIDF vectorizer

In [51]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data
TFIDF_FeatureList=vectorizer.get_feature_names()
# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer.transform(X_train['clean_essays'].values)
X_cv_essay_tfidf = vectorizer.transform(X_cv['clean_essays'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['clean_essays'].values)
```

In [52]:

```
from sklearn.feature_extraction.text import TfidfVectorizer


vectorizer = TfidfVectorizer(min_df=5)
vectorizer.fit(X_train['clean_titles'].values) # fit has to happen only on train data
TFIDF_FeatureList=TFIDF_FeatureList + vectorizer.get_feature_names()
# we use the fitted CountVectorizer to convert the text to vector
X_train_titles_tfidf = vectorizer.transform(X_train['clean_titles'].values)
X_cv_titles_tfidf = vectorizer.transform(X_cv['clean_titles'].values)
X_test_titles_tfidf = vectorizer.transform(X_test['clean_titles'].values)
print("Train shape:",X_train_titles_tfidf.shape)
print("CV shape:",X_cv_titles_tfidf.shape)
print("Test shape:",X_test_titles_tfidf.shape)
```

```
Train shape: (22445, 2067)
CV shape: (11055, 2067)
Test shape: (16500, 2067)
```

In [53]:

```
vectorizer = TfidfVectorizer(min_df=5)
vectorizer.fit(X_train['clean_resource_summary'].values) # fit has to happen only on train
datadata
TFIDF_FeatureList=TFIDF_FeatureList + vectorizer.get_feature_names()
```

```
# we use the fitted CountVectorizer to convert the text to vector
X_train_summary_tfidf = vectorizer.transform(X_train['clean_resource_summary'].values)
X_cv_summary_tfidf = vectorizer.transform(X_cv['clean_resource_summary'].values)
X_test_summary_tfidf = vectorizer.transform(X_test['clean_resource_summary'].values)

print("After vectorizations")
print(X_train_summary_tfidf.shape, y_train.shape)
print(X_cv_summary_tfidf.shape, y_cv.shape)
print(X_test_summary_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 3875) (22445,)
(11055, 3875) (11055,)
(16500, 3875) (16500,)
====================================================================================================
```

In [183]:

```
X_test.columns
```

Out[183]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', 'project_title',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'price', 'quantity',
       'clean_categories', 'preprocessed_grade', 'clean_subcategories',
       'essay', 'clean_essays', 'clean_resource_summary', 'clean_titles'],
      dtype='object')
```

In [55]:

```
X_train_teacher_prefix_pos = X_train['teacher_prefix_pos'] . reshape((22445,1))
X_train_teacher_prefix_neg = X_train['teacher_prefix_neg'] . reshape((22445,1))
X_train_clean_categories_pos = X_train['clean_categories_pos'] . reshape((22445,1))
X_train_clean_categories_neg = X_train['clean_categories_neg'] . reshape((22445,1))
X_train_school_state_pos = X_train['school_state_pos'].reshape((22445,1))
X_train_school_state_neg = X_train['school_state_neg'].reshape((22445,1))
X_train_clean_subcategories_pos = X_train['clean_subcategories_pos'].reshape((22445,1))
X_train_clean_subcategories_neg = X_train['clean_subcategories_neg'].reshape((22445,1))
X_train_project_grade_category_pos = X_train['project_grade_category_pos'].reshape((22445,1))
X_train_project_grade_category_neg = X_train['project_grade_category_neg'].reshape((22445,1))
```

In [56]:

```
len(TFIDF_FeatureList)
```

Out[56]:

```
15120
```

In [57]:

```
TFIDF_FeatureList.append('price')
TFIDF_FeatureList.append('teacher_number_of_previously_posted_projects')
TFIDF_FeatureList.append('quantity')
TFIDF_FeatureList.append('X_train_teacher_prefix_pos')
TFIDF_FeatureList.append('X_train_clean_categories_neg')
TFIDF_FeatureList.append('X_train_teacher_prefix_neg')
TFIDF_FeatureList.append('X_train_clean_categories_pos')
TFIDF_FeatureList.append('X_train_school_state_pos')
TFIDF_FeatureList.append('X_train_school_state_neg')
TFIDF_FeatureList.append('X_train_clean_subcategories_pos')
TFIDF_FeatureList.append('X_train_clean_subcategories_neg')
TFIDF_FeatureList.append('X_train_project_grade_category_pos')
TFIDF_FeatureList.append('X_train_project_grade_category_neg')
```

In [58]:

```
type(X_train_project_grade_category_pos)
type(X_train_project_grade_category_pos)
```

Out[58]:

```
numpy.ndarray
```

In [145]:

```
X_train_teacher_prefix_pos.shape
```

Out[145]:

```
(22445, 1)
```

In [215]:

```
X_te_teacher_prefix_pos = X_test['teacher_prefix_pos'] . reshape((16500,1))
X_te_teacher_prefix_neg = X_test['teacher_prefix_neg'] . reshape((16500,1))
X_te_clean_categories_pos = X_test['clean_categories_pos'] . reshape((16500,1))
X_te_clean_categories_neg = X_test['clean_categories_neg'] . reshape((16500,1))
X_te_school_state_pos = X_test['school_state_pos'].reshape((16500,1))
X_te_school_state_neg = X_test['school_state_neg'].reshape((16500,1))
X_te_clean_subcategories_pos = X_test['clean_subcategories_pos'].reshape((16500,1))
X_te_clean_subcategories_neg = X_test['clean_subcategories_neg'].reshape((16500,1))
X_te_project_grade_category_pos = X_test['project_grade_category_pos'].reshape((16500,1))
X_te_project_grade_category_neg = X_test['project_grade_category_neg'].reshape((16500,1))
```

Concatinating all features (TFIDF)

In [216]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr =
hstack((X_train_essay_tfidf,X_train_titles_tfidf,X_train_summary_tfidf,X_train_teacher_prefix_pos,
X_train_teacher_prefix_neg,X_train_clean_categories_pos,X_train_clean_categories_neg,X_train_school
_state_pos
,X_train_school_state_neg,X_train_clean_subcategories_pos,X_train_clean_subcategories_neg,X_train_p
roject_grade_category_neg,X_train_project_grade_category_pos,
X_train_price_std,X_train_projects_std,X_train_qty_std)).tocsr()
#X_cv = hstack((X_cv_essay_tfidf,X_cv_titles_tfidf,X_cv_summary_tfidf,X_cv_teacher_prefix_pos, X_c
v_teacher_prefix_neg,X_cv_clean_categories_pos,X_cv_clean_categories_neg,X_cv_school_state_pos ,X_
cv_school_state_neg,X_cv_clean_subcategories_pos,X_cv_clean_subcategories_neg,X_cv_project_grade_ca
ry_neg,X_cv_project_grade_category_pos, X_cv_price_std,X_cv_projects_std,X_cv_qty_std)).tocsr()
X_te = hstack((X_test_essay_tfidf,X_test_titles_tfidf,X_test_summary_tfidf,X_te_teacher_prefix_pos
,
X_te_teacher_prefix_neg,X_te_clean_categories_pos,X_te_clean_categories_neg,X_te_school_state_pos
,X_te_school_state_neg,X_te_clean_subcategories_pos,X_te_clean_subcategories_neg,X_te_project_grade
_category_neg,X_te_project_grade_category_pos, X_test_price_std,X_test_projects_std,X_test_qty_std
)).tocsr()

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 15133) (22445,)
(11055, 8773) (11055,)
(16500, 15133) (16500,)
====================================================================================================
```

In [56]:

```
%%time
import warnings
warnings.filterwarnings("ignore")
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
```

```
import math
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from sklearn import metrics
from sklearn import cross_validation
```

```
Wall time: 179 ms
```

In [192]:

```
def plotfpr_tpr(X_train,X_test,y_train,y_test,best_tune_parameters,type_algo):
    if type_algo == "rft":
        dt=RandomForestClassifier(class_weight = 'balanced')
    else:
        dt = GradientBoostingClassifier()
    clf = GridSearchCV(dt, best_tune_parameters, cv=3, scoring='roc_auc')
    clf.fit(X_train, y_train)
    y_train_pred= clf.predict_proba(X_train)[:,1]
    y_test_pred= clf.predict_proba(X_test)[:,1]
    train_fpr1, train_tpr1, tr_thresholds1 = roc_curve(y_train, y_train_pred)
    test_fpr1, test_tpr1, te_thresholds1 = roc_curve(y_test, y_test_pred)
    plt.plot(train_fpr1, train_tpr1, label="train AUC ="+str(auc(train_fpr1, train_tpr1)))
    plt.plot(test_fpr1, test_tpr1, label="test AUC ="+str(auc(test_fpr1, test_tpr1)))
    plt.legend()
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ERROR PLOTS")
    plt.grid(True)
    plt.show()
```

In [58]:

```
%%time
n_estimators = [10, 50, 100, 150, 200, 300, 500, 1000]
parameters = {'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth': [2, 3, 4, 5, 6
, 7, 8, 9, 10]}
dt1 = RandomForestClassifier(class_weight = 'balanced')
clf = GridSearchCV(dt1, parameters, cv=3, scoring='roc_auc',return_train_score=True)
set1=clf.fit(X_tr, y_train)
```

```
Wall time: 30min 3s
```

In [59]:

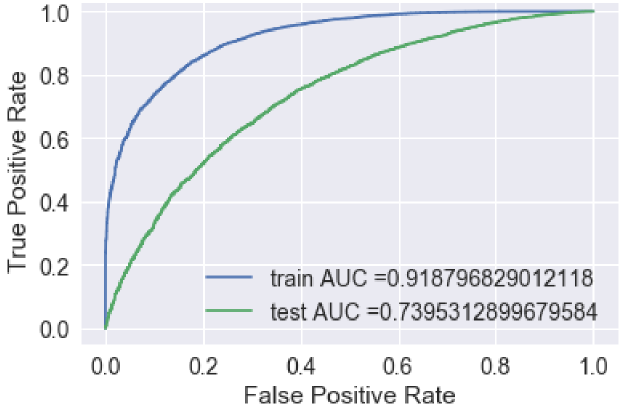
```
print(clf.best_estimator_)
#Mean cross-validated score of the best_estimator
print(clf.score(X_tr,y_train))
print(clf.score(X_te,y_test))
```

```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=10, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=1000, n_jobs=1, oob_score=False,
            random_state=None, verbose=0, warm_start=False)
0.9153365040953924
0.7569867178141335
```

Random Forest with Optimal Parameters

In [194]:

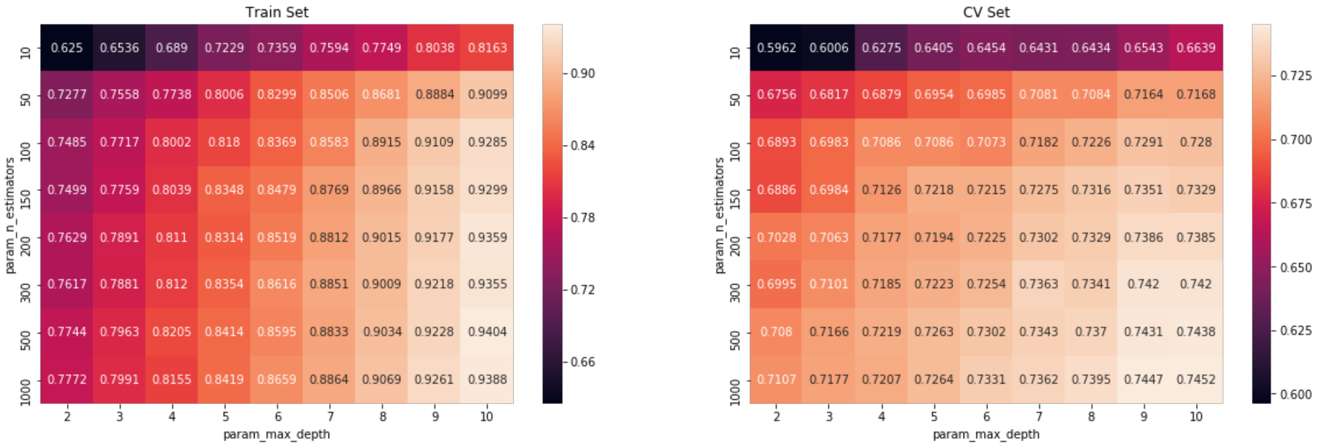
```
%%time
best_tune_parameters ={'n_estimators':[1000],'max_depth':[10]}
plotfpr_tpr(X_tr,X_te,y_train,y_test,best_tune_parameters,"rft")
```



```
Wall time: 5min 46s
```

In [61]:

```
import seaborn as sns
import pandas as pd
#reference https://seaborn.pydata.org/generated/seaborn.heatmap.html ,
https://blog.exploratory.io/quick-introduction-to-heatmap-c21a9f9e4644?gi=cbae67554962

max_scores1 = pd.DataFrame(clf.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max
().unstack()[['mean_test_score', 'mean_train_score']]
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



In [62]:

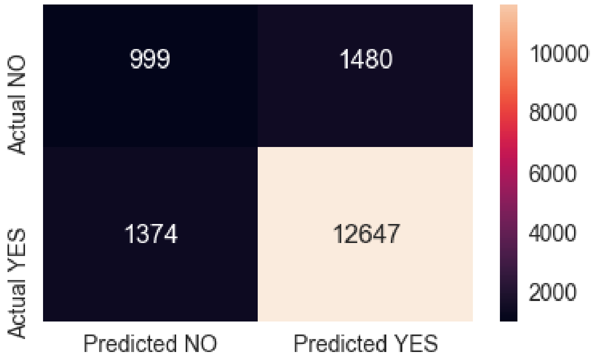
```
def get_confusion_matrix(y_test,y_pred):
    df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(2),range(2))
    df_cm.columns = ['Predicted NO','Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})
    sns.set(font_scale=1.4)#for label size
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

In [63]:

```
predbow = (clf.predict(X_te))
predbow_train=(clf.predict(X_tr))
```

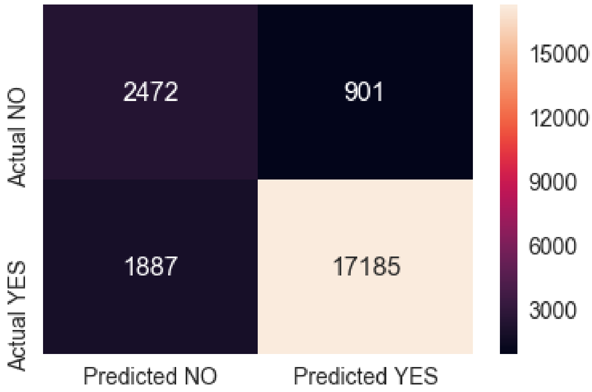
In [64]:

```
get_confusion_matrix(y_test,predbow)
```



In [65]:

```
get_confusion_matrix(y_train,predbow_train)
```



In [66]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test ,predbow))
```

```
             precision    recall  f1-score   support

          0       0.42      0.40      0.41      2479
          1       0.90      0.90      0.90     14021

avg / total       0.82      0.83      0.83     16500
```

GBDT on TFIDF

In [217]:

```
from sklearn.ensemble import GradientBoostingClassifier
```

In [ ]:

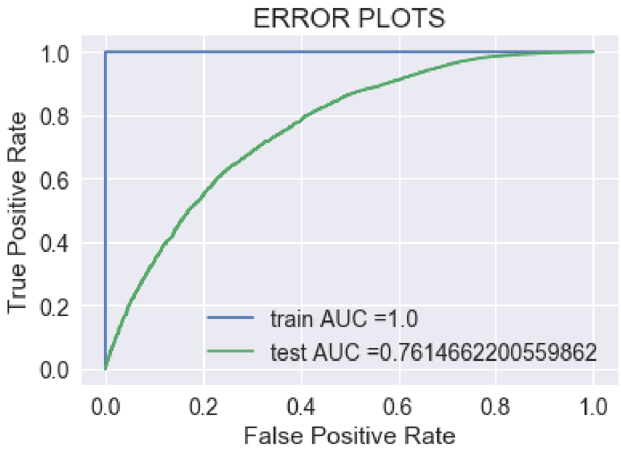
```
%%time

parameters = {'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth': [2, 3, 4, 5, 6
, 7, 8, 9, 10]}
List_param = optimal_hyper(X_tr,y_train,parameters)
max_ele = max(List_param)
for i,ele in enumerate(List_param):
    if ele == max_ele:
        max_index = i
        print(max_index)

max_row = int(max_index / 9)
max_col = max_index % 9
n_estimators = [10,50, 100, 150, 200, 300, 500, 1000]
max_depth = [2,3,4,5,6,7,8,9,10]
```

```
best_n_estimator = n_estimators[max_row]
best_max_depth = max_depth[max_col]
print(best_n_estimator,best_max_depth)
```

GBDT with Optimal Parameters

In [202]:

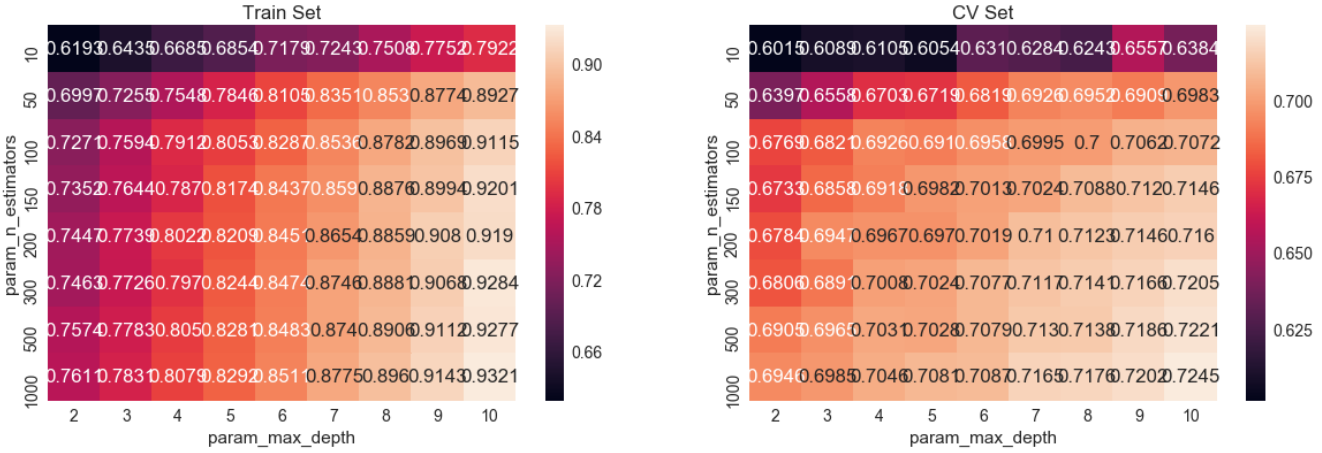
```
%%time
best_n_estimator =1000
best_max_depth = 10
best_tune_parameters ={'n_estimators':[best_n_estimator],'max_depth':[best_max_depth]}
plotfpr_tpr(X_tr,X_te,y_train,y_test,best_tune_parameters,"gbt")
```



```
Wall time: 3h 17min 21s
```

In [92]:

```
import seaborn as sns
import pandas as pd
#reference https://seaborn.pydata.org/generated/seaborn.heatmap.html ,
https://blog.exploratory.io/quick-introduction-to-heatmap-c21a9f9e4644?gi=cbae67554962

max_scores1 = pd.DataFrame(clf.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max
().unstack()[['mean_test_score', 'mean_train_score']]
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



In [93]:

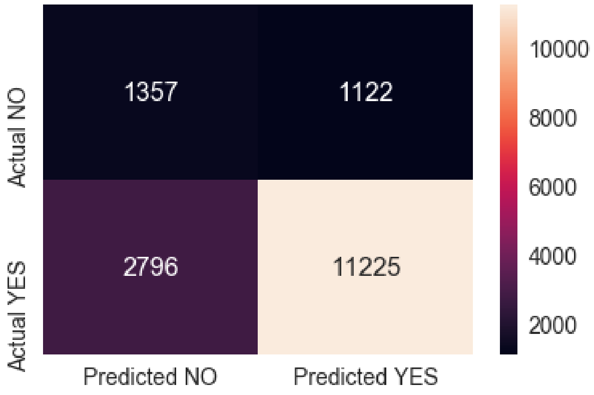
```
def get_confusion_matrix(y_test,y_pred):
    df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(2),range(2))
    df_cm.columns = ['Predicted NO','Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})
    sns.set(font_scale=1.4)#for label size
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

In [94]:

```
predbow = (clf.predict(X_te))
predbow_train=(clf.predict(X_tr))
```

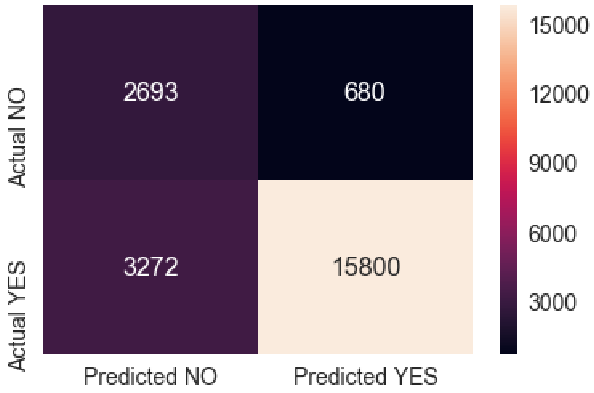
In [95]:

```
get_confusion_matrix(y_test,predbow)
```



In [96]:

```
get_confusion_matrix(y_train,predbow_train)
```



In [97]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test ,predbow))
```

```
             precision    recall  f1-score   support

          0       0.33      0.55      0.41      2479
          1       0.91      0.80      0.85     14021

avg / total       0.82      0.76      0.78     16500
```

In [98]:

```
print("AUC score for Decision Tree with TFIDF is ",round(metrics.roc_auc_score(y_test ,predbow),3)
)
```

```
AUC score for Decision Tree with TFIDF is  0.674
```

Random Forest on BOW

In [60]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10, max_features=5000)
vectorizer.fit(X_train['clean_essays'].values) # fit has to happen only on train data
Bow_FeatureList =vectorizer.get_feature_names()
# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['clean_essays'].values)
X_cv_essay_bow = vectorizer.transform(X_cv['clean_essays'].values)
X_test_essay_bow = vectorizer.transform(X_test['clean_essays'].values)

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
====================================================================================================
```

In [61]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10, max_features=5000)
vectorizer.fit(X_train['clean_titles'].values) # fit has to happen only on train data
Bow_FeatureList=Bow_FeatureList + (vectorizer.get_feature_names())
# we use the fitted CountVectorizer to convert the text to vector
X_train_titles_bow = vectorizer.transform(X_train['clean_titles'].values)
X_cv_titles_bow = vectorizer.transform(X_cv['clean_titles'].values)
X_test_titles_bow = vectorizer.transform(X_test['clean_titles'].values)

print("After vectorizations")
print(X_train_titles_bow.shape, y_train.shape)
print(X_cv_titles_bow.shape, y_cv.shape)
print(X_test_titles_bow.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1239) (22445,)
(11055, 1239) (11055,)
(16500, 1239) (16500,)
====================================================================================================
```

In [62]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10, max_features=5000)
vectorizer.fit(X_train['clean_resource_summary'].values) # fit has to happen only on train data
Bow_FeatureList=Bow_FeatureList + (vectorizer.get_feature_names())
# we use the fitted CountVectorizer to convert the text to vector
X_train_summary_bow = vectorizer.transform(X_train['clean_resource_summary'].values)
X_cv_summary_bow = vectorizer.transform(X_cv['clean_resource_summary'].values)
X_test_summary_bow = vectorizer.transform(X_test['clean_resource_summary'].values)

print("After vectorizations")
print(X_train_summary_bow.shape, y_train.shape)
print(X_cv_summary_bow.shape, y_cv.shape)
print(X_test_summary_bow.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 2521) (22445,)
(11055, 2521) (11055,)
```

```
(11055, 2521) (11055,)
(16500, 2521) (16500,)
====================================================================================================
```

In [203]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_bow,X_train_titles_bow,X_train_summary_bow,X_train_teacher_prefix_pos
,
X_train_teacher_prefix_neg,X_train_clean_categories_pos,X_train_clean_categories_neg,X_train_school
_state_pos
,X_train_school_state_neg,X_train_clean_subcategories_pos,X_train_clean_subcategories_neg,X_train_p
roject_grade_category_neg,X_train_project_grade_category_pos,
X_train_price_std,X_train_projects_std,X_train_qty_std)).tocsr()
#X_cv = hstack((X_cv_essay_bow,X_cv_titles_bow,X_cv_summary_bow,X_cv_teacher_prefix_pos, X_cv_teac
her_prefix_neg,X_cv_clean_categories_pos,X_cv_clean_categories_neg,X_cv_school_state_pos
,X_cv_school_state_neg,X_cv_clean_subcategories_pos,X_cv_clean_subcategories_neg,X_cv_project_grade
egory_neg,X_cv_project_grade_category_pos, X_cv_price_std,X_cv_projects_std,X_cv_qty_std)).tocsr()
X_te = hstack((X_test_essay_bow,X_test_titles_bow,X_test_summary_bow,X_te_teacher_prefix_pos,
X_te_teacher_prefix_neg,X_te_clean_categories_pos,X_te_clean_categories_neg,X_te_school_state_pos
,X_te_school_state_neg,X_te_clean_subcategories_pos,X_te_clean_subcategories_neg,X_te_project_grade
_category_neg,X_te_project_grade_category_pos, X_test_price_std,X_test_projects_std,X_test_qty_std
)).tocsr()

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
#print(X_cv.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 8773) (22445,)
(16500, 8773) (16500,)
====================================================================================================
```

In [64]:

```
%%time
import warnings
warnings.filterwarnings("ignore")
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
import math
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from sklearn import metrics
from sklearn import cross_validation
```

```
Wall time: 129 ms
```

In [68]:

```
%%time
n_estimators = [10, 50, 100, 150, 200, 300, 500, 1000]
parameters = {'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth': [2, 3, 4, 5, 6
, 7, 8, 9, 10]}
dt1 = RandomForestClassifier(class_weight = 'balanced')
clf = GridSearchCV(dt1, parameters, cv=3, scoring='roc_auc',return_train_score=True)
set1=clf.fit(X_tr, y_train)
```

```
Wall time: 21min 2s
```

In [69]:

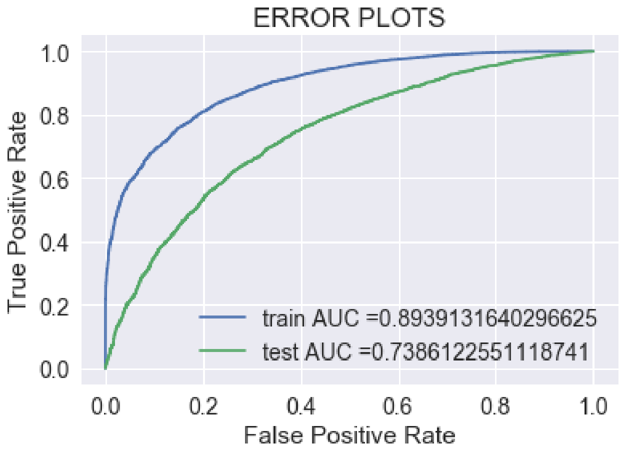
```
print(clf.best_estimator_)
#Mean cross-validated score of the best_estimator
print(clf.score(X_tr,y_train))
print(clf.score(X_te,y_test))
```

```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=10, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=1000, n_jobs=1, oob_score=False,
            random_state=None, verbose=0, warm_start=False)
0.8907239851824545
0.754302946964573
```

Random Forest with Optimal Parameters

In [200]:

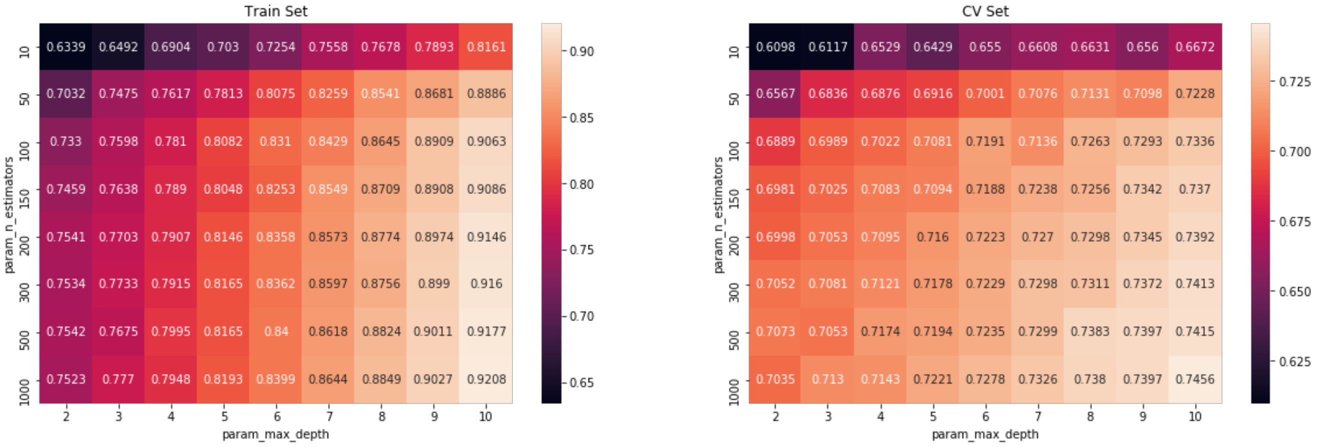
```
%%time
best_tune_parameters ={'n_estimators':[1000],'max_depth':[10]}
plotfpr_tpr(X_tr,X_te,y_train,y_test,best_tune_parameters,"rft")
```



```
Wall time: 1min 48s
```

In [71]:

```
import seaborn as sns
import pandas as pd
#reference https://seaborn.pydata.org/generated/seaborn.heatmap.html ,
https://blog.exploratory.io/quick-introduction-to-heatmap-c21a9f9e4644?gi=cbae67554962

max_scores1 = pd.DataFrame(clf.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max
().unstack()[['mean_test_score', 'mean_train_score']]
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



Train Set

param_n_estimators	2	3	4	5	6	7	8	9	10
10	0.6339	0.6492	0.6904	0.703	0.7254	0.7558	0.7678	0.7893	0.8161
50	0.7032	0.7475	0.7617	0.7813	0.8075	0.8259	0.8541	0.8681	0.8886
100	0.733	0.7598	0.781	0.8082	0.831	0.8429	0.8645	0.8909	0.9063
150	0.7459	0.7638	0.789	0.8048	0.8253	0.8549	0.8709	0.8908	0.9086
200	0.7541	0.7703	0.7907	0.8146	0.8358	0.8573	0.8774	0.8974	0.9146
300	0.7534	0.7733	0.7915	0.8165	0.8362	0.8597	0.8756	0.899	0.916
500	0.7542	0.7675	0.7995	0.8165	0.84	0.8618	0.8824	0.9011	0.9177
1000	0.7523	0.777	0.7948	0.8193	0.8399	0.8644	0.8849	0.9027	0.9208

CV Set

param_n_estimators	2	3	4	5	6	7	8	9	10
10	0.6098	0.6117	0.6529	0.6429	0.655	0.6608	0.6631	0.656	0.6672
50	0.6567	0.6836	0.6876	0.6916	0.7001	0.7076	0.7131	0.7098	0.7228
100	0.6889	0.6989	0.7022	0.7081	0.7191	0.7136	0.7263	0.7293	0.7336
150	0.6981	0.7025	0.7083	0.7094	0.7188	0.7238	0.7256	0.7342	0.737
200	0.6998	0.7053	0.7095	0.716	0.7223	0.727	0.7298	0.7345	0.7392
300	0.7052	0.7081	0.7121	0.7178	0.7229	0.7298	0.7311	0.7372	0.7413
500	0.7073	0.7053	0.7174	0.7194	0.7235	0.7299	0.7383	0.7397	0.7415
1000	0.7035	0.713	0.7143	0.7221	0.7278	0.7326	0.738	0.7397	0.7456

In [72]:

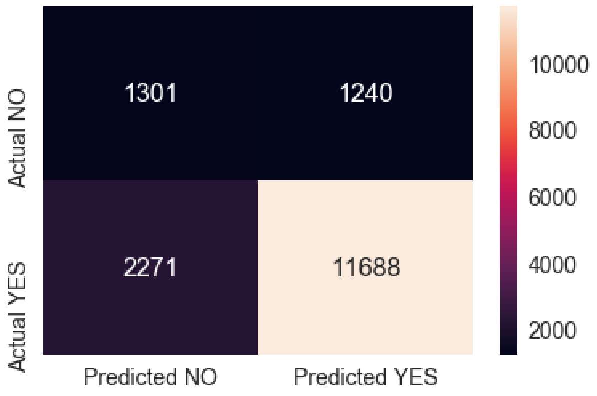
```
def get_confusion_matrix(y_test,y_pred):
    df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(2),range(2))
    df_cm.columns = ['Predicted NO','Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})
    sns.set(font_scale=1.4)#for label size
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

In [75]:

```
predbow = (clf.predict(X_te))
predbow_train=(clf.predict(X_tr))
```

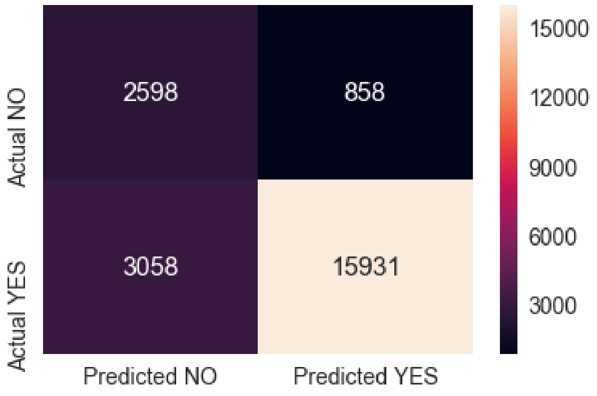
In [76]:

```
get_confusion_matrix(y_test,predbow)
```



In [77]:

```
get_confusion_matrix(y_train,predbow_train)
```



In [78]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test ,predbow))
```

```
             precision    recall  f1-score   support

          0       0.36      0.51      0.43      2541
          1       0.90      0.84      0.87     13959

avg / total       0.82      0.79      0.80     16500
```

In [79]:

```
print("AUC score for Decision Tree with TFIDF is ",round(metrics.roc_auc_score(y_test ,predbow),3)
)
```

```
AUC score for Decision Tree with TFIDF is  0.675
```

GBDT on BOW

In [84]:

```
from sklearn.ensemble import GradientBoostingClassifier
```

In [87]:

```
%%time
parameters = {'n_estimators': [10,50, 100, 150, 200, 300, 500, 1000], 'max_depth':
[2,3,4,5,6,7,8,9,10]}
List_param = optimal_hyper(X_tr,y_train,parameters)
```

```
Wall time: 4h 57min 33s
```

In [88]:

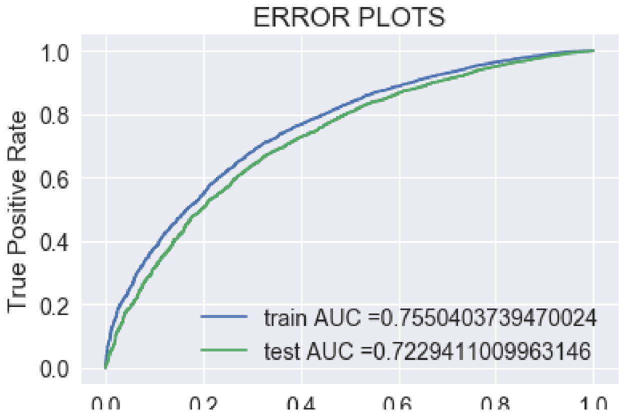
```
max_ele = max(List_param)
for i,ele in enumerate(List_param):
    if ele == max_ele:
        max_index = i
        print(max_index)

max_row = int(max_index / 9)
max_col = max_index % 9
n_estimators = [10,50, 100, 150, 200, 300, 500, 1000]
max_depth = [2,3,4,5,6,7,8,9,10]
best_n_estimator = n_estimators[max_row]
best_max_depth = max_depth[max_col]
print(best_n_estimator,best_max_depth)
```

```
64
1000 3
```

GBDT with Optimal Parameters

In [89]:

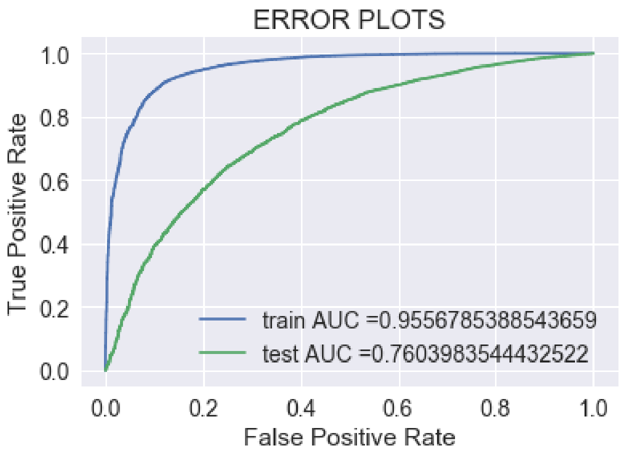
```
%%time

clf = GradientBoostingClassifier(n_estimators =best_n_estimator, max_depth = best_max_depth)
set1=clf.fit(X_tr, y_train)
# below is graph of previous run could not remove it as again GBDT function has to run and it will
take hours
```



```
Wall time: 7min 53s
```

In [204]:

```
best_tune_parameters = {'n_estimators': [1000], 'max_depth': [3]}
plotfpr_tpr(X_tr,X_te,y_train,y_test,best_tune_parameters,"gbt")
```



In [90]:

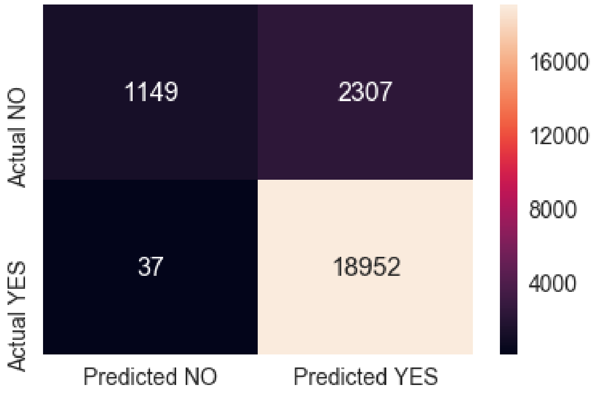
```
def get_confusion_matrix(y_test,y_pred):
    df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(2),range(2))
    df_cm.columns = ['Predicted NO','Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})
    sns.set(font_scale=1.4)#for label size
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

In [93]:

```
predbow = (clf.predict(X_te))
predbow_train=(clf.predict(X_tr))
```

In [94]:

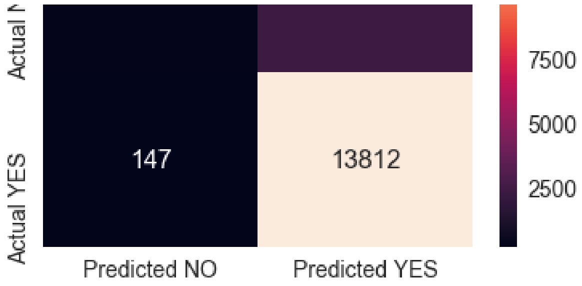
```
get_confusion_matrix(y_train,predbow_train)
```



In [95]:

```
get_confusion_matrix(y_test,predbow)
```

NO
225
2316
12500
10000



In [96]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test ,predbow))
```

```
             precision    recall  f1-score   support

          0       0.60      0.09      0.15      2541
          1       0.86      0.99      0.92     13959

avg / total       0.82      0.85      0.80     16500
```

In [97]:

```
print("AUC score for Decision Tree with TFIDF is ",round(metrics.roc_auc_score(y_test ,predbow),3)
)
```

```
AUC score for Decision Tree with TFIDF is  0.539
```

Random Forest on TFIDF - AVG W2V

In [134]:

```
(X_train_new['clean_essays'].values).shape
```

Out[134]:

```
(8978,)
```

In [135]:

```
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())

# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train_new['clean_essays'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [136]:

```
(X_train_new['clean_essays'].values).shape
```

Out[136]:

```
(8978,)
```

In [143]:

```
# average Word2Vec
# compute average word2vec for each review.
train_w2v_vectors_essays_new = []; # the avg-w2v for each essay is stored in this list
for sentence in (X_train_new['clean_essays'].values): # for each essay in training data
for sentence in (X_train_new['clean_essays'].values): # for each essay in training data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_w2v_vectors_essays_new.append(vector)
print("train  vector")
print(len(train_w2v_vectors_essays_new))
print(len(train_w2v_vectors_essays_new[0]))
print('='*50)

# average Word2Vec
# compute average word2vec for each review.
test_w2v_vectors_essays_new = []; # the avg-w2v for each essay is stored in this list
for sentence in (X_test['clean_essays'].values): # for each essay in training data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_w2v_vectors_essays_new.append(vector)

print("Test vec")
print(len(test_w2v_vectors_essays_new))
print(len(test_w2v_vectors_essays_new[0]))
print('='*50)

# average Word2Vec
# compute average word2vec for each review.
cv_w2v_vectors_essays_new = []; # the avg-w2v for each essay is stored in this list
for sentence in (X_cv['clean_essays'].values): # for each essay in training data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_w2v_vectors_essays_new.append(vector)

print("CV vec")
print(len(cv_w2v_vectors_essays_new))
print(len(cv_w2v_vectors_essays_new[0]))
print('='*50)
```

```
train  vector
8978
50
==================================================
Test vec
16500
50
==================================================
CV vec
11055
50
==================================================
```

In [148]:

```
# Changing list to numpy arrays


train_w2v_vectors_essays = np.array(train_tfidf_w2v_essays_new)
test_w2v_vectors_essays = np.array(test_tfidf_w2v_essays_new)
cv_w2v_vectors_essays= np.array(cv_tfidf_w2v_essays_new)
```

In [145]:

```
# average Word2Vec
# compute average word2vec for each review.
train_w2v_vectors_titles_new = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_train_new['clean_titles'].values): # for each essay in training data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_w2v_vectors_titles_new.append(vector)
print("train  vector")
print(len(train_w2v_vectors_titles_new))
print(len(train_w2v_vectors_titles_new[0]))
print('='*50)

# average Word2Vec
# compute average word2vec for each review.
test_w2v_vectors_titles_new = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_test_new['clean_titles'].values): # for each essay in training data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_w2v_vectors_titles_new.append(vector)

print("Test vec")
print(len(test_w2v_vectors_titles_new))
print(len(test_w2v_vectors_titles_new[0]))
print('='*50)

# average Word2Vec
# compute average word2vec for each review.
cv_w2v_vectors_titles_new = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_cv_new['clean_titles'].values): # for each essay in training data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_w2v_vectors_titles_new.append(vector)

print("CV vec")
print(len(cv_w2v_vectors_titles_new))
print(len(cv_w2v_vectors_titles_new[0]))
print('='*50)
```

```


  0%|                                                                                         | 0/8
[00:00<?, ?it/s]


 44%|█████████████████████████████████                                            | 3939/8978
[00:00<00:00, 39294.29it/s]


100%|████████████████████████████████████████████████████████████████████████████| 8978/8978
[00:00<00:00, 48007.93it/s]
```

```
train  vector
8978
50
==================================================
  0%|                                                                                    | 0/6
[00:00<?, ?it/s]


 54%|█████████████████████████████████████████▏                                  | 3581/6600 [00:00<
0:00, 35722.14it/s]


100%|████████████████████████████████████████████████████████████████████████████| 6600/6600
[00:00<00:00, 42553.55it/s]
```

```
Test vec
6600
50
==================================================
```

```
  0%|                                                                                    | 0/4
[00:00<?, ?it/s]


100%|████████████████████████████████████████████████████████████████████████████| 4422/4422
[00:00<00:00, 47168.84it/s]
```

```
CV vec
4422
50
==================================================
```

In [149]:

```
# Changing list to numpy arrays
train_w2v_vectors_titles = np.array(train_w2v_vectors_titles_new)
test_w2v_vectors_titles = np.array(test_w2v_vectors_titles_new)
cv_w2v_vectors_titles = np.array(cv_w2v_vectors_titles_new)
```

In [206]:

```
from scipy import sparse
train_tfidf_w2v_essays = sparse.csr_matrix(train_w2v_vectors_essays)
train_w2v_vectors_titles = sparse.csr_matrix(train_w2v_vectors_titles)
cv_tfidf_w2v_essays = sparse.csr_matrix(cv_w2v_vectors_essays)
cv_w2v_vectors_titles = sparse.csr_matrix(cv_w2v_vectors_titles)
test_tfidf_w2v_essays = sparse.csr_matrix(test_w2v_vectors_essays)
test_w2v_vectors_titles = sparse.csr_matrix(test_w2v_vectors_titles)
```

In [207]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((train_tfidf_w2v_essays,train_w2v_vectors_titles,X_train_teacher_prefix_pos[:8978],
X_train_teacher_prefix_neg[:8978],X_train_clean_categories_pos[:8978],X_train_clean_categories_neg
[:8978],X_train_school_state_pos[:8978]
,X_train_school_state_neg[:8978],X_train_clean_subcategories_pos[:8978],X_train_clean_subcategories
_neg[:8978],X_train_project_grade_category_neg[:8978],X_train_project_grade_category_pos[:8978], X
_train_price_std[:8978],X_train_projects_std[:8978],X_train_qty_std[:8978])).tocsr()
#X_cv = hstack((cv_tfidf_w2v_essays,cv_w2v_vectors_titles,X_cv_teacher_prefix_pos[:4422], X_cv_tea
cher_prefix_neg[:4422],X_cv_clean_categories_pos[:4422],X_cv_clean_categories_neg[:4422],X_cv_scho
ate_pos[:4422]
,X_cv_school_state_neg[:4422],X_cv_clean_subcategories_pos[:4422],X_cv_clean_subcategories_neg[:44
_cv_project_grade_category_neg[:4422],X_cv_project_grade_category_pos[:4422],
X_cv_price_std[:4422],X_cv_projects_std[:4422],X_cv_qty_std[:4422])).tocsr()
X_te = hstack((test_tfidf_w2v_essays,test_w2v_vectors_titles,X_te_teacher_prefix_pos[:6600], X_te_
teacher_prefix_neg[:6600],X_te_clean_categories_pos[:6600],X_te_clean_categories_neg[:6600],X_te_s
chool_state_pos[:6600] ,X_te_school_state_neg[:6600],X_te_clean_subcategories_pos[:6600],X_te_clea
n_subcategories_neg[:6600],X_te_project_grade_category_neg[:6600],X_te_project_grade_category_pos[
:6600], X_test_price_std[:6600],X_test_projects_std[:6600],X_test_qty_std[:6600])).tocsr()

print("Final Data matrix")
print("Final Data matrix")
print(X_tr.shape, y_train.shape)
#print(X_cv.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(8978, 113) (22445,)
(6600, 113) (16500,)
====================================================================================================
```

In [157]:

```
%%time
n_estimators = [10, 50, 100, 150, 200, 300, 500, 1000]
parameters = {'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth': [2, 3, 4, 5, 6
, 7, 8, 9, 10]}
dt1 = RandomForestClassifier(class_weight = 'balanced')
clf = GridSearchCV(dt1, parameters, cv=3, scoring='roc_auc',return_train_score=True)
set1=clf.fit(X_tr, y_train_new)
```

```
Wall time: 21h 16min 7s
```

In [159]:

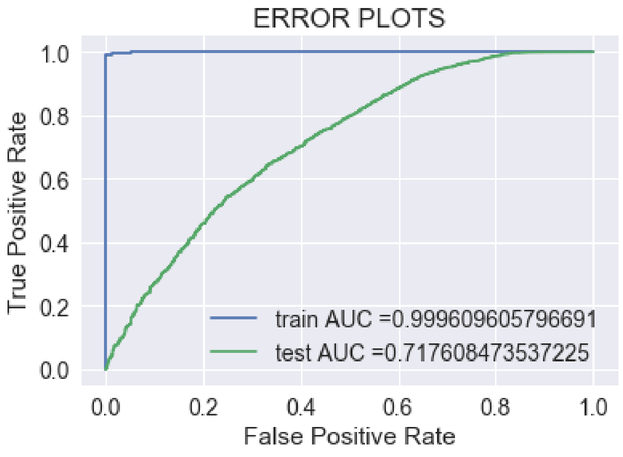
```
print(clf.best_estimator_)
#Mean cross-validated score of the best_estimator
print(clf.score(X_tr,y_train_new))
print(clf.score(X_te,y_test_new))
```

```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=10, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=500, n_jobs=1, oob_score=False, random_state=None,
            verbose=0, warm_start=False)
0.9987793737793738
0.7278327613006159
```

Random Forest with Optimal Parameters

In [209]:

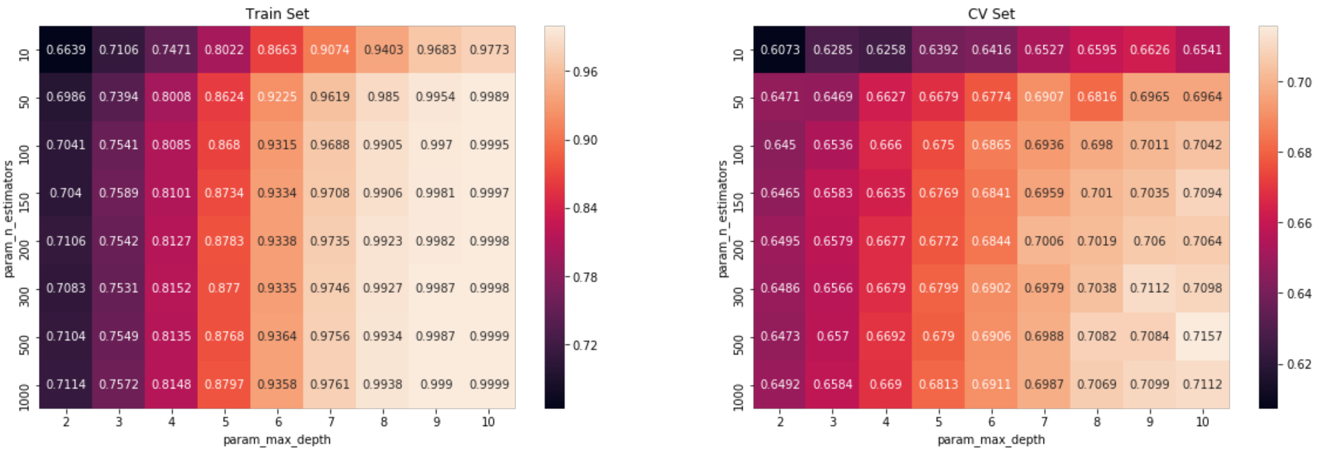
```
%%time
best_tune_parameters ={'n_estimators':[500],'max_depth':[10]}
plotfpr_tpr(X_tr,X_te,y_train_new,y_test_new,best_tune_parameters,"rft")
```



```
Wall time: 5min 9s
```

In [162]:

```
import seaborn as sns
import pandas as pd
#reference https://seaborn.pydata.org/generated/seaborn.heatmap.html ,
https://blog.exploratory.io/quick-introduction-to-heatmap-c21a9f9e4644?gi=cbae67554962

max_scores1 = pd.DataFrame(clf.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max
().unstack()[['mean_test_score', 'mean_train_score']]
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```

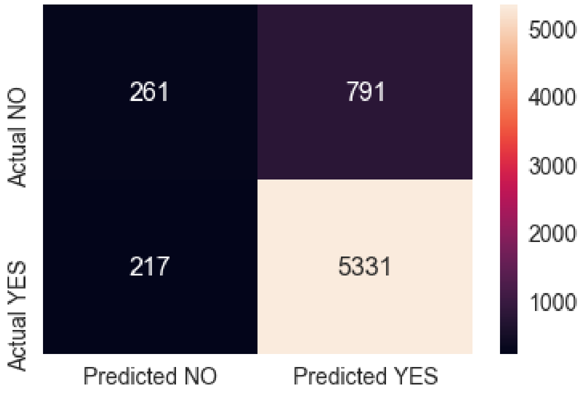


In [163]:

```
predbow = (clf.predict(X_te))
predbow_train=(clf.predict(X_tr))
```

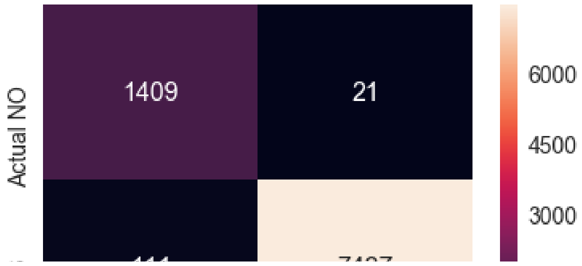
In [166]:

```
get_confusion_matrix(y_test_new,predbow)
```



In [167]:

```
get_confusion_matrix(y_train_new,predbow_train)
```



In [168]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test_new ,predbow))
```

```
             precision    recall  f1-score   support

          0       0.55      0.25      0.34      1052
          1       0.87      0.96      0.91      5548

avg / total       0.82      0.85      0.82      6600
```

In [169]:

```
print("AUC score for Decision Tree with TFIDF is ",round(metrics.roc_auc_score(y_test_new ,predbow
),3))
```

```
AUC score for Decision Tree with TFIDF is  0.604
```

GBDT on TFIDF - Avg W2V

In [170]:

```
from sklearn.ensemble import GradientBoostingClassifier
```

In [213]:

```
for i in parameters['n_estimators']:
    print(i)
```

```
500
```

In [214]:

```
%%time
from sklearn.cross_validation import cross_val_score
parameters = {'n_estimators': [10,50, 100, 150, 200, 300, 500, 1000], 'max_depth':
[2,3,4,5,6,7,8,9,10]}
optimal_hyper(X_tr,y_train_new,parameters)
```

```
[0.61737727 0.60538317]
[0.62160087 0.62200944]
[0.62746284 0.62142095]
[0.63467801 0.63572752]
[0.63241464 0.63395518]
[0.62960725 0.6378043 ]
[0.64317265 0.63311079]
[0.65777143 0.63130047]
[0.63498041 0.61455598]
[0.6113802202037496, 0.621805155628685, 0.6244418935595406, 0.6352027675557088,
0.6331849125966773, 0.6337057748822454, 0.6381417204946617, 0.6445359489477136,
0.6247681968270203]
[0.63641274 0.64469688]
[0.63739498 0.65383948]
[0.65180903 0.657797  ]
[0.65814091 0.66988968]
[0.67074018 0.66842844]
[0.67629864 0.66715955]
[0.68334686 0.68260679]
[0.67333837 0.67743523]
[0.67630271 0.65739046]
```

```
[0.6113802202037496, 0.621805155628685, 0.6244418935595406, 0.6352027675557088,
0.6331849125966773, 0.6337057748822454, 0.6381417204946617, 0.6445359489477136,
0.6247681968270203, 0.640554808201867, 0.645617233852528, 0.654803013626543, 0.6640152904858787,
0.6695843107607814, 0.6717290923173276, 0.6829768270944743, 0.6753868018573901,
0.6668465874348228]
[0.64226785 0.65405035]
[0.64381321 0.66743267]
[0.65711845 0.6648115 ]
[0.67896761 0.67319644]
[0.68098806 0.68930555]
[0.67759421 0.67701017]
[0.68470062 0.6870016 ]
[0.68313562 0.66919075]
[0.67138611 0.67366301]
[0.6113802202037496, 0.621805155628685, 0.6244418935595406, 0.6352027675557088,
0.6331849125966773, 0.6337057748822454, 0.6381417204946617, 0.6445359489477136,
0.6247681968270203, 0.640554808201867, 0.645617233852528, 0.654803013626543, 0.6640152904858787,
0.6695843107607814, 0.6717290923173276, 0.6829768270944743, 0.6753868018573901,
0.6668465874348228, 0.6481591011002776, 0.6556229409170586, 0.660964975670858, 0.6760820260820262,
0.6851468086762205, 0.6773021890668949, 0.6858511123217006, 0.6761631849867145,
0.6725245607598548]
[0.64471337 0.65754537]
[0.65441649 0.66482076]
[0.66903621 0.66845031]
[0.68604326 0.68189415]
[0.68654356 0.67646836]
[0.68238741 0.68630416]
[0.68209946 0.68202163]
[0.68419032 0.67831056]
[0.68571381 0.6774986 ]
[0.6113802202037496, 0.621805155628685, 0.6244418935595406, 0.6352027675557088,
0.6331849125966773, 0.6337057748822454, 0.6381417204946617, 0.6445359489477136,
0.6247681968270203, 0.640554808201867, 0.645617233852528, 0.654803013626543, 0.6640152904858787,
0.6695843107607814, 0.6717290923173276, 0.6829768270944743, 0.6753868018573901,
0.6668465874348228, 0.6481591011002776, 0.6556229409170586, 0.660964975670858, 0.6760820260820262,
0.6851468086762205, 0.6773021890668949, 0.6858511123217006, 0.6761631849867145,
0.6725245607598548, 0.6511293687764277, 0.6596186272656861, 0.6687432599197305,
0.6839687074981192, 0.6815059609177256, 0.6843457814046049, 0.6820605467664291,
0.6812504400739695, 0.6816062051356169]
[0.64773737 0.65934977]
[0.65797896 0.66606594]
[0.66911774 0.67287217]
[0.68990368 0.68063304]
[0.68270982 0.67347846]
[0.67600439 0.68928072]
[0.68709351 0.68148613]
[0.68810929 0.66877124]
[0.68558596 0.67487483]
[0.6113802202037496, 0.621805155628685, 0.6244418935595406, 0.6352027675557088,
0.6331849125966773, 0.6337057748822454, 0.6381417204946617, 0.6445359489477136,
0.6247681968270203, 0.640554808201867, 0.645617233852528, 0.654803013626543, 0.6640152904858787,
0.6695843107607814, 0.6717290923173276, 0.6829768270944743, 0.6753868018573901,
0.6668465874348228, 0.6481591011002776, 0.6556229409170586, 0.660964975670858, 0.6760820260820262,
0.6851468086762205, 0.6773021890668949, 0.6858511123217006, 0.6761631849867145,
0.6725245607598548, 0.6511293687764277, 0.6596186272656861, 0.6687432599197305,
0.6839687074981192, 0.6815059609177256, 0.6843457814046049, 0.6820605467664291,
0.6812504400739695, 0.6816062051356169, 0.6535435682494507, 0.6620224502577443, 0.670994956289074,
0.68526836173895, 0.6780941369176664, 0.682642556171968, 0.6842898225251166, 0.6784402666755608, 0
.6802303949362772]
[0.65494124 0.66038667]
[0.66987041 0.6694635 ]
[0.67342176 0.67542108]
[0.68827421 0.68436635]
[0.6905889  0.67544109]
[0.67782324 0.68616259]
[0.68580016 0.68535508]
[0.68722952 0.68224214]
[0.68502266 0.6785092 ]
[0.6113802202037496, 0.621805155628685, 0.6244418935595406, 0.6352027675557088,
0.6331849125966773, 0.6337057748822454, 0.6381417204946617, 0.6445359489477136,
0.6247681968270203, 0.640554808201867, 0.645617233852528, 0.654803013626543, 0.6640152904858787,
0.6695843107607814, 0.6717290923173276, 0.6829768270944743, 0.6753868018573901,
0.6668465874348228, 0.6481591011002776, 0.6556229409170586, 0.660964975670858, 0.6760820260820262,
0.6851468086762205, 0.6773021890668949, 0.6858511123217006, 0.6761631849867145,
0.6725245607598548, 0.6511293687764277, 0.6596186272656861, 0.6687432599197305,
0.6839687074981192, 0.6815059609177256, 0.6843457814046049, 0.6820605467664291,
0.6812504400739695, 0.6816062051356169, 0.6535435682494507, 0.6620224502577443, 0.670994956289074,
0.68526836173895, 0.6780941369176664, 0.682642556171968, 0.6842898225251166, 0.6784402666755608, 0
.6802303949362772, 0.6576639576639576, 0.6696669520198932, 0.6744214185390656, 0.6863202774967482,
0.68301499772088, 0.6819929143458556, 0.685577617930559, 0.6847358259122965, 0.6817659288247524]
[0.66069685 0.66218551]
[0.67109668 0.67248528]
[0.67957686 0.67587172]
[0.68826012 0.68091024]
[0.68693749 0.67690344]
[0.67685081 0.68179706]
[0.67951831 0.67896539]
[0.67972028 0.68154802]
[0.68228401 0.68266387]
[0.6113802202037496, 0.621805155628685, 0.6244418935595406, 0.6352027675557088,
0.6331849125966773, 0.6337057748822454, 0.6381417204946617, 0.6445359489477136,
0.6247681968270203, 0.640554808201867, 0.645617233852528, 0.654803013626543, 0.6640152904858787,
0.6695843107607814, 0.6717290923173276, 0.6829768270944743, 0.6753868018573901,
0.6668465874348228, 0.6481591011002776, 0.6556229409170586, 0.660964975670858, 0.6760820260820262,
0.6851468086762205, 0.6773021890668949, 0.6858511123217006, 0.6761631849867145,
0.6725245607598548, 0.6511293687764277, 0.6596186272656861, 0.6687432599197305,
0.6839687074981192, 0.6815059609177256, 0.6843457814046049, 0.6820605467664291,
0.6812504400739695, 0.6816062051356169, 0.6535435682494507, 0.6620224502577443, 0.670994956289074,
0.68526836173895, 0.6780941369176664, 0.682642556171968, 0.6842898225251166, 0.6784402666755608, 0
.6802303949362772, 0.6576639576639576, 0.6696669520198932, 0.6744214185390656, 0.6863202774967482,
0.68301499772088, 0.6819929143458556, 0.685577617930559, 0.6847358259122965, 0.6817659288247524, 0
.6614411820294173, 0.67179098061451, 0.6777242894889953, 0.6845851816440052, 0.6819204642734055, 0
.6793239352062882, 0.6792418498300851, 0.6806341512223866, 0.6824739383562912]
[0.66593216 0.66531179]
[0.67471252 0.67528508]
[0.67953906 0.67275692]
[0.68977954 0.67747896]
[0.68513384 0.67570866]
[0.6810392 0.6869916]
[0.68583796 0.68757972]
[0.68413251 0.68434856]
[0.69128561 0.68738183]
[0.6113802202037496, 0.621805155628685, 0.6244418935595406, 0.6352027675557088,
0.6331849125966773, 0.6337057748822454, 0.6381417204946617, 0.6445359489477136,
0.6247681968270203, 0.640554808201867, 0.645617233852528, 0.654803013626543, 0.6640152904858787,
0.6695843107607814, 0.6717290923173276, 0.6829768270944743, 0.6753868018573901,
0.6668465874348228, 0.6481591011002776, 0.6556229409170586, 0.660964975670858, 0.6760820260820262,
0.6851468086762205, 0.6773021890668949, 0.6858511123217006, 0.6761631849867145,
0.6725245607598548, 0.6511293687764277, 0.6596186272656861, 0.6687432599197305,
0.6839687074981192, 0.6815059609177256, 0.6843457814046049, 0.6820605467664291,
0.6812504400739695, 0.6816062051356169, 0.6535435682494507, 0.6620224502577443, 0.670994956289074,
0.68526836173895, 0.6780941369176664, 0.682642556171968, 0.6842898225251166, 0.6784402666755608, 0
.6802303949362772, 0.6576639576639576, 0.6696669520198932, 0.6744214185390656, 0.6863202774967482,
0.68301499772088, 0.6819929143458556, 0.685577617930559, 0.6847358259122965, 0.6817659288247524, 0
.6614411820294173, 0.67179098061451, 0.6777242894889953, 0.6845851816440052, 0.6819204642734055, 0
.6793239352062882, 0.6792418498300851, 0.6806341512223866, 0.6824739383562912, 0.6656219773866833,
0.674998795587031, 0.6761479908538732, 0.6836292483351307, 0.680421248068307, 0.6840154016624604,
0.6867088396500161, 0.6842405342405342, 0.6893337187454835]
Wall time: 3h 4min 29s
```

In [215]:

```
List_param = [0.6113802202037496, 0.621805155628685, 0.6244418935595406, 0.6352027675557088, 0.6331
849125966773, 0.6337057748822454, 0.6381417204946617, 0.6445359489477136, 0.6247681968270203,
0.640554808201867, 0.645617233852528, 0.654803013626543, 0.6640152904858787, 0.6695843107607814, 0
.6717290923173276, 0.6829768270944743, 0.6753868018573901, 0.6668465874348228, 0.6481591011002776,
0.6556229409170586, 0.660964975670858, 0.6760820260820262, 0.6851468086762205, 0.6773021890668949,
0.6858511123217006, 0.6761631849867145, 0.6725245607598548, 0.6511293687764277, 0.6596186272656861
, 0.6687432599197305, 0.6839687074981192, 0.6815059609177256, 0.6843457814046049,
0.6820605467664291, 0.6812504400739695, 0.6816062051356169, 0.6535435682494507, 0.6620224502577443
, 0.670994956289074, 0.68526836173895, 0.6780941369176664, 0.682642556171968, 0.6842898225251166, 0
.6784402666755608, 0.6802303949362772, 0.6576639576639576, 0.6696669520198932, 0.6744214185390656,
0.6863202774967482, 0.68301499772088, 0.6819929143458556, 0.685577617930559, 0.6847358259122965,
0.6817659288247524, 0.6614411820294173, 0.67179098061451, 0.6777242894889953, 0.6845851816440052, 0
.6819204642734055, 0.6793239352062882, 0.6792418498300851, 0.6806341512223866, 0.6824739383562912,
0.6656219773866833, 0.674998795587031, 0.6761479908538732, 0.6836292483351307, 0.680421248068307,
0.6840154016624604, 0.6867088396500161, 0.6842405342405342, 0.6893337187454835]
```

In [251]:

```
max_ele = max(List_param)
for i,ele in enumerate(List_param):
    if ele == max_ele:
        max_index = i
        print(max_index)

max_row = int(max_index / 9)
max_col = max_index % 9
n_estimators = [10,50, 100, 150, 200, 300, 500, 1000]
max_depth = [2,3,4,5,6,7,8,9,10]
best_n_estimator = n_estimators[max_row]
best_max_depth = max_depth[max_col]
print(best_n_estimator,best_max_depth)
```

```
71
1000 10
```

In [234]:

```
print(max_row, max_col)
```

```
7 8
```

In [252]:

```
%%time
gbt = GradientBoostingClassifier(n_estimators=best_n_estimator , max_depth =best_max_depth)
gbt.fit(X_tr,y_train_new)
```

```
Wall time: 21min 49s
```

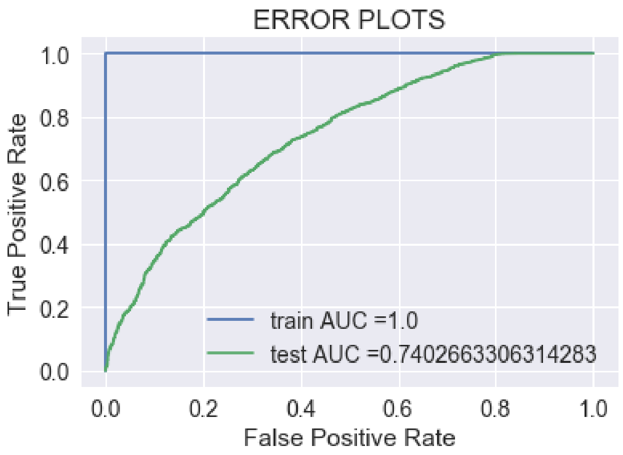
GBDT with Optimal Parameters

In [253]:

```
def get_confusion_matrix(y_test,y_pred):
    df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(2),range(2))
    df_cm.columns = ['Predicted NO','Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})
    sns.set(font_scale=1.4)#for label size
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

In [210]:

```
%%time
best_tune_parameters ={'n_estimators':[1000],'max_depth':[10]}
plotfpr_tpr(X_tr,X_te,y_train_new,y_test_new,best_tune_parameters,"gbt")
```



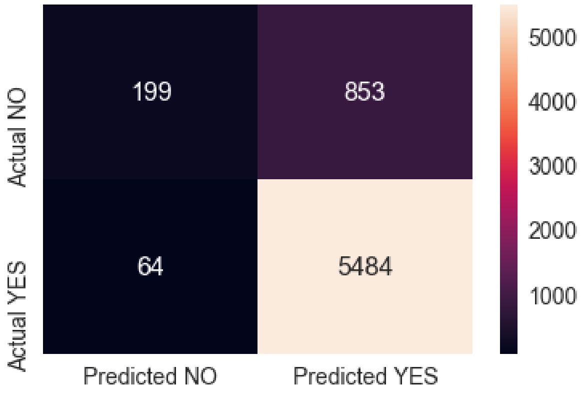
```
Wall time: 49min 5s
```

In [255]:

```
predbow = (gbt.predict(X_te))
predbow_train=(gbt.predict(X_tr))
```

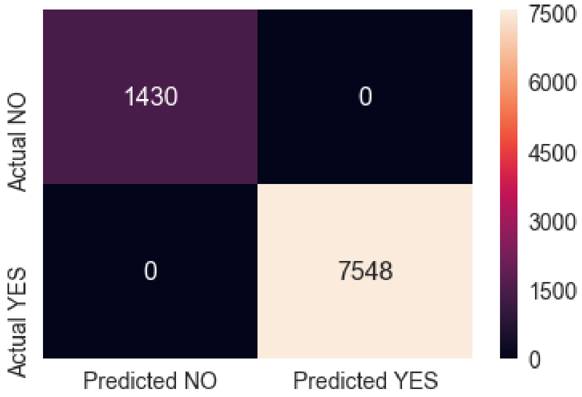
In [256]:

```
get_confusion_matrix(y_test_new,predbow)
```



In [257]:

```
get_confusion_matrix(y_train_new,predbow_train)
```



In [258]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test_new ,predbow))
```

```
             precision    recall  f1-score   support

          0       0.76      0.19      0.30      1052
          1       0.87      0.99      0.92      5548

avg / total       0.85      0.86      0.82      6600
```

AVG W2V

In [110]:

```
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [111]:

```python
# average Word2Vec
# compute average word2vec for each review.
train_w2v_vectors_essays = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm((X_train_new['clean_essays'].values)): # for each essay in training data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_w2v_vectors_essays.append(vector)
print("train  vector")
print(len(train_w2v_vectors_essays))
print(len(train_w2v_vectors_essays[0]))
print('='*50)

# average Word2Vec
# compute average word2vec for each review.
test_w2v_vectors_essays = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm((X_test_new['clean_essays'].values)): # for each essay in training data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_w2v_vectors_essays.append(vector)

print("Test vec")
print(len(test_w2v_vectors_essays))
print(len(test_w2v_vectors_essays[0]))
print('='*50)

# average Word2Vec
# compute average word2vec for each review.
cv_w2v_vectors_essays = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm((X_cv_new['clean_essays'].values[0:20000])): # for each essay in training
data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_w2v_vectors_essays.append(vector)

print("CV vec")
print(len(cv_w2v_vectors_essays))
print(len(cv_w2v_vectors_essays[0]))
print('='*50)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 8978/8978
[00:10<00:00, 857.94it/s]
```

```
train  vector
8978
50
==================================================
```

```
100%|████████████████████████████████████████████████████████████████████████████| 6600/6600
[00:07<00:00, 912.39it/s]
```

```
Test vec
6600
50
==================================================
```

```
100%|███████████████████████████████████████████████████████████████████████████| 4422/4422
```

```
[00:03<00:00, 1152.68it/s]
```

```
CV vec
4422
50
==================================================
```

In [112]:

```
train_w2v_vectors_essays = np.array(train_w2v_vectors_essays)
test_w2v_vectors_essays = np.array(test_w2v_vectors_essays)
cv_w2v_vectors_essays = np.array(cv_w2v_vectors_essays)
```

In [113]:

```
# average Word2Vec
# compute average word2vec for each review.
train_w2v_vectors_titles = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_train_new['clean_titles'].values): # for each essay in training data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_w2v_vectors_titles.append(vector)
print("train  vector")
print(len(train_w2v_vectors_titles))
print(len(train_w2v_vectors_titles[0]))
print('='*50)

# average Word2Vec
# compute average word2vec for each review.
test_w2v_vectors_titles = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_test_new['clean_titles'].values): # for each essay in training data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_w2v_vectors_titles.append(vector)

print("Test vec")
print(len(test_w2v_vectors_titles))
print(len(test_w2v_vectors_titles[0]))
print('='*50)

# average Word2Vec
# compute average word2vec for each review.
cv_w2v_vectors_titles = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_cv_new['clean_titles'].values): # for each essay in training data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_w2v_vectors_titles.append(vector)

print("CV vec")
print(len(cv_w2v_vectors_titles))
print(len(cv_w2v_vectors_titles[0]))
print('='*50)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 8978/8978
[00:00<00:00, 18734.19it/s]
```

```
train  vector
8978
50
==================================================
```

```
100%|████████████████████████████████████████████████████████████████████████████| 6600/6600
[00:00<00:00, 14705.91it/s]
```

```
Test vec
6600
50
==================================================
```

```
100%|████████████████████████████████████████████████████████████████████████████| 4422/4422
[00:00<00:00, 17185.36it/s]
```

```
CV vec
4422
50
==================================================
```

In [114]:

```
# Changing list to numpy arrays
train_w2v_vectors_titles = np.array(train_w2v_vectors_titles)
test_w2v_vectors_titles = np.array(test_w2v_vectors_titles)
cv_w2v_vectors_titles = np.array(cv_w2v_vectors_titles)
```

In [211]:

```
from scipy import sparse
train_w2v_vectors_essays = sparse.csr_matrix(train_w2v_vectors_essays)
train_w2v_vectors_titles = sparse.csr_matrix(train_w2v_vectors_titles)
cv_w2v_vectors_essays = sparse.csr_matrix(cv_w2v_vectors_essays)
cv_w2v_vectors_titles = sparse.csr_matrix(cv_w2v_vectors_titles)
test_w2v_vectors_essays = sparse.csr_matrix(test_w2v_vectors_essays)
test_w2v_vectors_titles = sparse.csr_matrix(test_w2v_vectors_titles)
```

In [212]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((train_w2v_vectors_essays,train_w2v_vectors_titles,X_train_teacher_prefix_pos[:8978]
,
X_train_teacher_prefix_neg[:8978],X_train_clean_categories_pos[:8978],X_train_clean_categories_neg
[:8978],X_train_school_state_pos[:8978]
,X_train_school_state_neg[:8978],X_train_clean_subcategories_pos[:8978],X_train_clean_subcategories
_neg[:8978],X_train_project_grade_category_neg[:8978],X_train_project_grade_category_pos[:8978], X
_train_price_std[:8978],X_train_projects_std[:8978],X_train_qty_std[:8978])).tocsr()
#X_cv = hstack((cv_w2v_vectors_essays,cv_w2v_vectors_titles,X_cv_teacher_prefix_pos[:4422], X_cv_t
eacher_prefix_neg[:4422],X_cv_clean_categories_pos[:4422],X_cv_clean_categories_neg[:4422],X_cv_sch
state_pos[:4422]
,X_cv_school_state_neg[:4422],X_cv_clean_subcategories_pos[:4422],X_cv_clean_subcategories_neg[:442
_cv_project_grade_category_neg[:4422],X_cv_project_grade_category_pos[:4422],
X_cv_price_std[:4422],X_cv_projects_std[:4422],X_cv_qty_std[:4422])).tocsr()
X_te = hstack((test_w2v_vectors_essays,test_w2v_vectors_titles,X_te_teacher_prefix_pos[:6600], X_t
e_teacher_prefix_neg[:6600],X_te_clean_categories_pos[:6600],X_te_clean_categories_neg[:6600],X_te
_school_state_pos[:6600] ,X_te_school_state_neg[:6600],X_te_clean_subcategories_pos[:6600],X_te_cl
ean_subcategories_neg[:6600],X_te_project_grade_category_neg[:6600],X_te_project_grade_category_pos
[:6600], X_test_price_std[:6600],X_test_projects_std[:6600],X_test_qty_std[:6600])).tocsr()

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
#print(X_cv.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(8978, 113) (22445,)
(6600, 113) (16500,)
====================================================================================================
```

In [120]:

```
%%time
n_estimators = [10, 50, 100, 150, 200, 300, 500, 1000]
parameters = {'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth': [2, 3, 4, 5, 6
, 7, 8, 9, 10]}
dt1 = RandomForestClassifier()
clf = GridSearchCV(dt1, parameters, cv=3, scoring='roc_auc',return_train_score=True ,n_jobs = -1)
set1=clf.fit(X_tr, y_train_new)
```

```
Wall time: 17min 10s
```

In [121]:

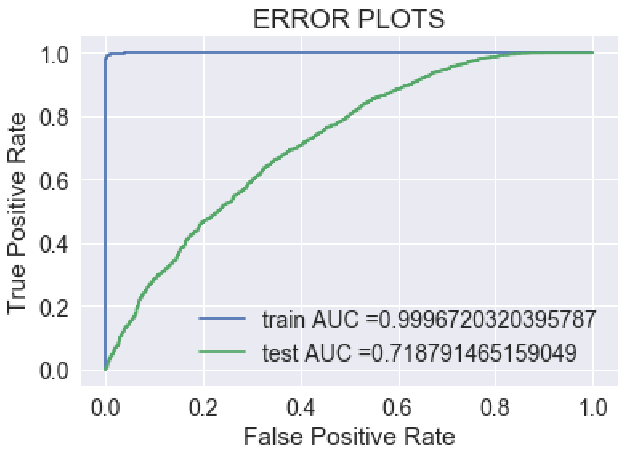
```
print(clf.best_estimator_)
#Mean cross-validated score of the best_estimator
print(clf.score(X_tr,y_train_new))
print(clf.score(X_te,y_test_new))
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=10, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=1000, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
0.9925234906361096
0.7164912131326204
```

Random Forest with Optimal Parameters

In [213]:

```
%%time
best_tune_parameters ={'n_estimators':[1000],'max_depth':[10]}
plotfpr_tpr(X_tr,X_te,y_train_new,y_test_new,best_tune_parameters,"rft")
```

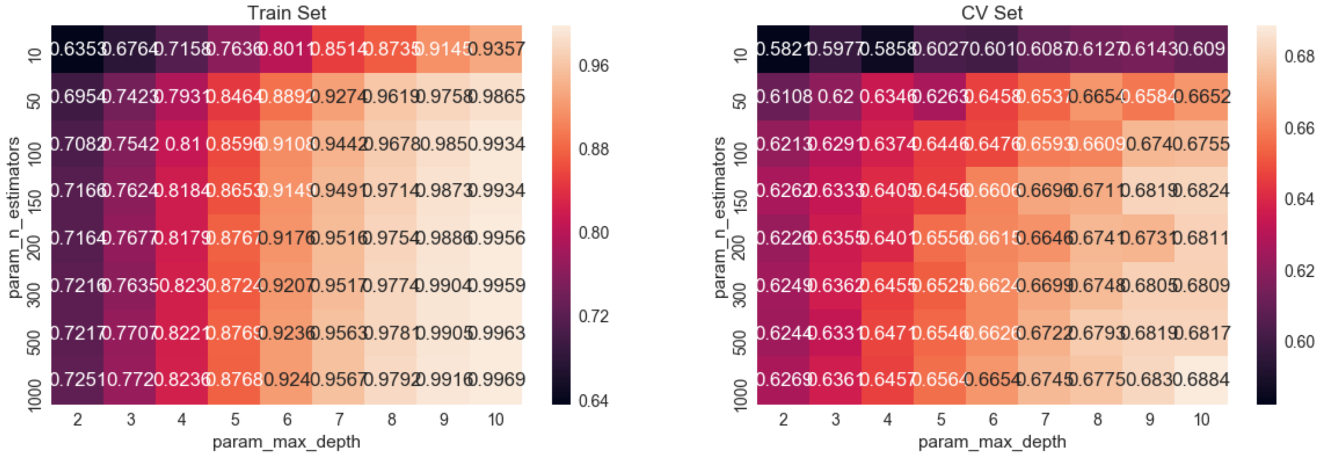


```
Wall time: 10min 27s
```

In [123]:

```
import seaborn as sns
import pandas as pd
#reference https://seaborn.pydata.org/generated/seaborn.heatmap.html ,
https://blog.exploratory.io/quick-introduction-to-heatmap-c21a9f9e4644?gi=cbae67554962

max_scores1 = pd.DataFrame(clf.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max
().unstack()[['mean_test_score', 'mean_train_score']]
fig, ax = plt.subplots(1,2, figsize=(20,6))
```

```
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



In [124]:

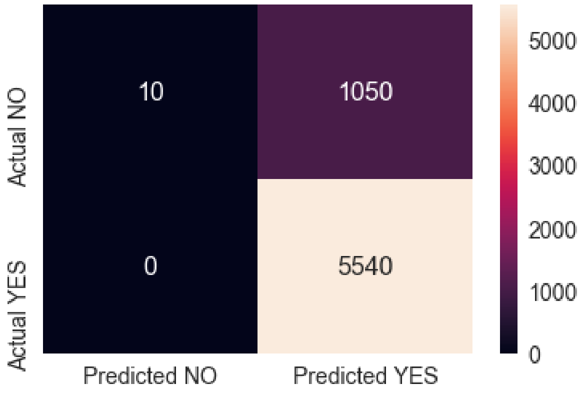
```
def get_confusion_matrix(y_test,y_pred):
    df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(2),range(2))
    df_cm.columns = ['Predicted NO','Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})
    sns.set(font_scale=1.4)#for label size
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

In [125]:

```
predbow = (clf.predict(X_te))
predbow_train=(clf.predict(X_tr))
```

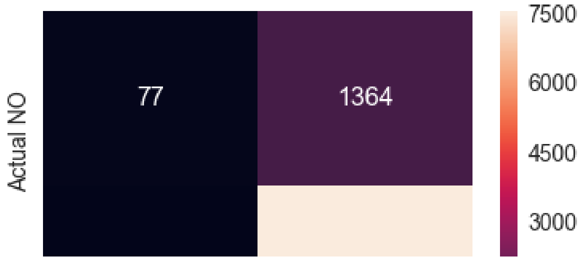
In [126]:

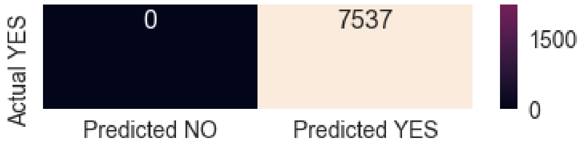
```
get_confusion_matrix(y_test_new,predbow)
```



In [127]:

```
get_confusion_matrix(y_train_new,predbow_train)
```





In [128]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test_new ,predbow))
```

```
             precision    recall  f1-score   support

          0       1.00      0.01      0.02      1060
          1       0.84      1.00      0.91      5540

avg / total       0.87      0.84      0.77      6600
```

GBDT on AVG W2V

In [130]:

```
from sklearn.ensemble import GradientBoostingClassifier
```

In [131]:

```
%%time
from sklearn.cross_validation import cross_val_score
parameters = {'n_estimators': [10,50, 100, 150, 200, 300, 500, 1000], 'max_depth':
[2,3,4,5,6,7,8,9,10]}
best_hyper = optimal_hyper(X_tr,y_train_new,parameters)
```

```
Wall time: 3h 12min 38s
```

In [142]:

```
max_ele = max(best_hyper)
for i,ele in enumerate(best_hyper):
    if ele == max_ele:
        max_index = i
        print(max_index)

max_row = int(max_index / 9)
max_col = max_index % 9
n_estimators = [10,50, 100, 150, 200, 300, 500, 1000]
max_depth = [2,3,4,5,6,7,8,9,10]
best_n_estimator = n_estimators[max_row]
best_max_depth = max_depth[max_col]
print(best_n_estimator,best_max_depth)
```

```
70
1000 9
```

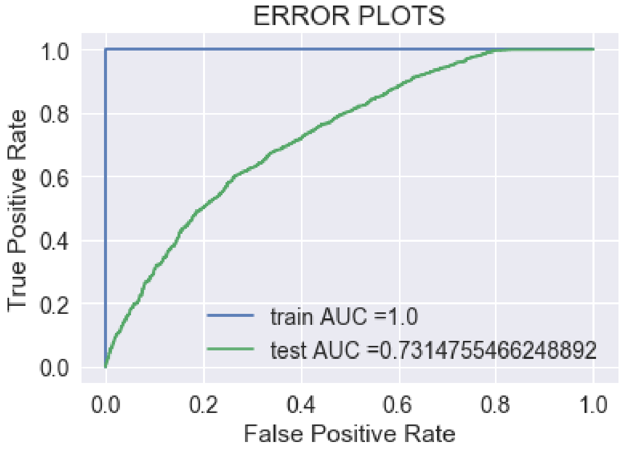
GBDT with Optimal Parameters

In [132]:

```
def get_confusion_matrix(y_test,y_pred):
    df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(2),range(2))
    df_cm.columns = ['Predicted NO','Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})
    sns.set(font_scale=1.4)#for label size
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

In [214]:

```
%%time
best_tune_parameters ={'n_estimators':[1000],'max_depth':[9]}
plotfpr_tpr(X_tr,X_te,y_train_new,y_test_new,best_tune_parameters,"gbt")
```



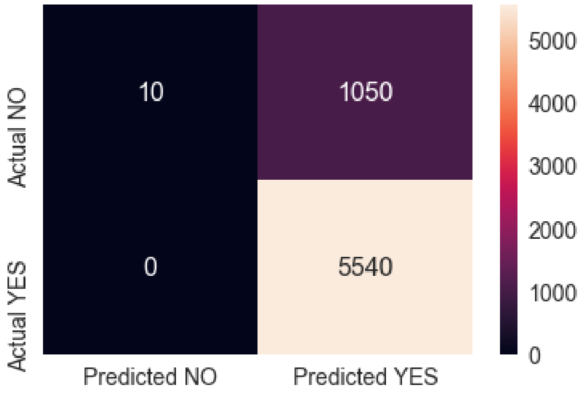
```
Wall time: 1h 56min 22s
```

In [134]:

```
predbow = (clf.predict(X_te))
predbow_train=(clf.predict(X_tr))
```

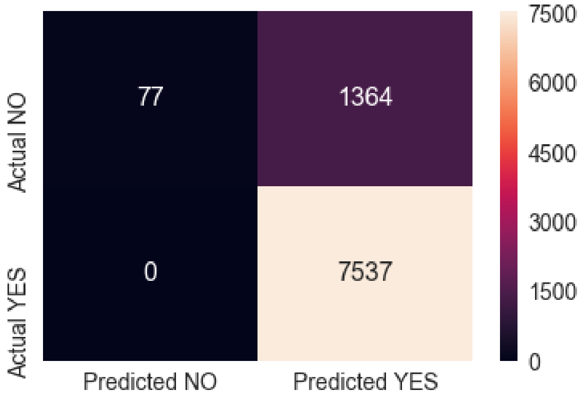
In [135]:

```
get_confusion_matrix(y_test_new,predbow)
```



In [136]:

```
get_confusion_matrix(y_train_new,predbow_train)
```



In [137]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test_new ,predbow))
```

```
             precision    recall  f1-score   support

          0       1.00      0.01      0.02      1060
          1       0.84      1.00      0.91      5540

avg / total       0.87      0.84      0.77      6600
```

In [143]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Algorithm","Vectorizer", "n_estimators","Max Depth", "Test AUC"]
x.add_row(["Random Forest","TFIDF",1000, 10, 0.757])
x.add_row(["Random Forest","BOW",1000, 10,0.756])
x.add_row(["Random Forest","TFIDF - AVG W2V", 1000, 10 ,0.722])
x.add_row(["Random Forest","AVG W2V", 1000, 10 ,0.719])
x.add_row(["GBDT","TFIDF",1000, 10,0.741])
x.add_row(["GBDT","BOW",1000, 3,0.723])
x.add_row(["GBDT","TFIDF - AVG W2V", 1000, 10,0.733])
x.add_row(["GBDT","AVG W2V", 1000, 9,0.718])

print(x)
```

```
+---------------+-----------------+--------------+-----------+----------+
|   Algorithm   |    Vectorizer   | n_estimators | Max Depth | Test AUC |
+---------------+-----------------+--------------+-----------+----------+
| Random Forest |      TFIDF      |     1000     |     10    |  0.757   |
| Random Forest |       BOW       |     1000     |     10    |  0.756   |
| Random Forest | TFIDF - AVG W2V |     1000     |     10    |  0.722   |
| Random Forest |     AVG W2V     |     1000     |     10    |  0.719   |
|      GBDT     |      TFIDF      |     1000     |     10    |  0.741   |
|      GBDT     |       BOW       |     1000     |     3     |  0.723   |
|      GBDT     | TFIDF - AVG W2V |     1000     |     10    |  0.733   |
|      GBDT     |     AVG W2V     |     1000     |     9     |  0.718   |
+---------------+-----------------+--------------+-----------+----------+
```