DonorsChoose

DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need of funding. Right now, a large number of volunteers is needed to manually screen each submission before it's approved to be posted on the DonorsChoose.org website.

Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result, there are three main problems they need to solve:

- How to scale current manual processes and resources to screen 500,000 projects so that they can be posted as quickly and as efficiently as possible
- How to increase the consistency of project vetting across different volunteers to improve the experience for teachers
- How to focus volunteer time on the applications that need the most assistance

The goal of the competition is to predict whether or not a DonorsChoose.org project proposal submitted by a teacher will be approved, using the text of project descriptions as well as additional metadata about the project, teacher, and school. DonorsChoose.org can then use this information to identify projects most likely to need further review before approval.

About the DonorsChoose Data Set

The `train.csv` data set provided by DonorsChoose contains the following features:

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located (Two-letter U.S. postal code). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy` • `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • `My students need hands on literacy materials to manage sensory needs!`
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*

Feature	Description
`project_essay_4`	Fourth application essay
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
`teacher_id`	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same teacher. **Example:** `2`

* See the section **Notes on the Essay Data** for more details about these features.

Additionally, the `resources.csv` data set provides more data about the resources required for each project. Each line in this file represents a resource required by a project:

Feature	Description
`id`	A `project_id` value from the `train.csv` file. **Example:** `p036502`
`description`	Desciption of the resource. **Example:** `Tenor Saxophone Reeds, Box of 25`
`quantity`	Quantity of the resource required. **Example:** `3`
`price`	Price of the resource required. **Example:** `9.95`

Note: Many projects require multiple resources. The `id` value corresponds to a `project_id` in train.csv, so you use it as a key to retrieve all resources needed for a project:

The data set contains the following label (the value you will attempt to predict):

Label	Description
`project_is_approved`	A binary flag indicating whether DonorsChoose approved the project. A value of `0` indicates the project was not approved, and a value of `1` indicates the project was approved.

Notes on the Essay Data

Prior to May 17, 2016, the prompts for the essays were as follows:

- __project_essay_1:__ "Introduce us to your classroom"
- __project_essay_2:__ "Tell us more about your students"
- __project_essay_3:__ "Describe how your students will use the materials you're requesting"
- __project_essay_3:__ "Close by sharing why your project will make a difference"

Starting on May 17, 2016, the number of essays was reduced from 4 to 2, and the prompts for the first 2 essays were changed to the following:

- __project_essay_1:__ "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."
- __project_essay_2:__ "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

For all projects with project_submitted_datetime of 2016-05-17 and later, the values of project_essay_3 and project_essay_4 will be NaN.

In [75]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
```

```
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

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

from tqdm import tqdm
import os

from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
from mpl_toolkits.mplot3d import Axes3D
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
```

1.1 Reading Data

In [2]:

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

In [3]:

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

```
Number of data points in train data (109248, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [4]:

```
print("Number of data points in train data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

```
Number of data points in train data (1541272, 4)
['id' 'description' 'quantity' 'price']
```

Out[4]:

	id	description	quantity	price
0	p233245	LC652 - Lakeshore Double-Space Mobile Drying	1	149.00

	id	description	quantity	price
0	[illegible]	... Rack	1	[illegible]
1	p069063	Bouncy Bands for Desks (Blue support pipes)	3	14.95

1.2 preprocessing of `project_subject_categories`

In [5]:

```
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]))
```

1.3 preprocessing of `project_subject_subcategories`

In [6]:

```
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]))
```

1.3 Text preprocessing

In [7]:

```
# 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 [8]:

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

Out[8]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_cate
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades P
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grade

In [9]:

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
price_data.head(2)
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

Assignment 9: RF and GBDT

Response Coding: Example

> The response tabel is built only on train dataset. For a category which is not there in train data and present in test data, we will encode them with default values Ex: in our test data if have State: D then we encode it as [0.5, 0.5]

1. **Apply both Random Forrest and GBDT on these feature sets**
    - Set 1: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(BOW) + preprocessed_eassay (BOW)
    - Set 2: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
    - Set 3: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V). Here for this set take **20K** datapoints only.

- Set 4: categorical(instead of one hot encoding, try [response coding](): use probability values), numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V). Here for this set take **20K** datapoints only.

2. **The hyper paramter tuning (Consider any two hyper parameters preferably n_estimators, max_depth)**

   - Consider the following range for hyperparameters **n_estimators** = [10, 50, 100, 150, 200, 300, 500, 1000], **max_depth** = [2, 3, 4, 5, 6, 7, 8, 9, 10]
   - Find the best hyper parameter which will give the maximum [AUC]() value
   - Find the best hyper paramter using simple cross validation data
   - You can write your own for loops to do this task

3. **Representation of results**

   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure
     with X-axis as **n_estimators**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

or

   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure
     [seaborn heat maps]() with rows as **n_estimators**, columns as **max_depth**, and values inside the cell representing **AUC Score**
   - You can choose either of the plotting techniques: 3d plot or heat map
   - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
   - Along with plotting ROC curve, you need to print the [confusion matrix]() with predicted and original labels of test data points

4. **Conclusion**

   - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library [link]()

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this [link.]()

2. Random Forest and GBDT

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

In [10]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
sample_data=project_data.sample(50000)
sample_data.shape
from sklearn.model_selection import train_test_split
data=sample_data
data.head()
y=data["project_is_approved"].values
x=data
```

```
x=data
x_train,x_test,y_train1,y_test1=train_test_split(x,y,train_size=0.8,test_size=0.2,stratify=y)
x_train,x_cv,y_train1,y_cv1=train_test_split(x_train,y_train1,train_size=0.75,test_size=0.25,strati
fy=y_train1)


print("shape of train data ")
print(x_train.shape)
print(y_train1.shape)
print("shape of test data ")
print(x_test.shape)
print(y_test1.shape)
print("shape of crossvalidation data ")
print(x_cv.shape)
print(y_cv1.shape)
```

```
shape of train data
(30000, 20)
(30000,)
shape of test data
(10000, 20)
(10000,)
shape of crossvalidation data
(10000, 20)
(10000,)
```

In [11]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
sample_data=project_data.sample(20000)
sample_data.shape
from sklearn.model_selection import train_test_split
data=sample_data
data.head()
y=data["project_is_approved"].values
x=data
x_train2,x_test2,y_train2,y_test2=train_test_split(x,y,train_size=0.8,test_size=0.2,stratify=y)
x_train2,x_cv2,y_train2,y_cv2=train_test_split(x_train2,y_train2,train_size=0.75,test_size=0.25,str
atify=y_train2)


print("shape of train data ")
print(x_train2.shape)
print(y_train2.shape)
print("shape of test data ")
print(x_test2.shape)
print(y_test2.shape)
print("shape of crossvalidation data ")
print(x_cv2.shape)
print(y_cv2.shape)
```

```
shape of train data
(12000, 20)
(12000,)
shape of test data
(4000, 20)
(4000,)
shape of crossvalidation data
(4000, 20)
(4000,)
```

In [ ]:

2.2 Make Data Model Ready: encoding numerical, categorical features

In [12]:

```
def get_fea_dict(alpha, feature ):
    fea_dict=dict()
    value_count=x_train[feature].value_counts()
    for i , denominator in value_count.items():
        vec=[]
        for k in range(0,2):
            cnt=x_train[(x_train["project_is_approved"]==k) & (x_train[feature]==i)]
            vec.append((cnt.shape[0]+alpha*10)/(denominator+alpha*20))
        fea_dict[i]=vec
    return fea_dict

def get_fea(alpha,feature, df):
    fea_dict=get_fea_dict(alpha,feature)
    value_count=x_train[feature].value_counts()
    gv_fea = []
    # for every feature values in the given data frame we will check if it is there in the train
data then we will add the feature to gv_fea
    # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gv_fea
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(fea_dict[row[feature]])
        else:
            gv_fea.append([1/2,1/2])
            #gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea
```

In [20]:

```
def get_fea_dict2(alpha, feature ):
    fea_dict=dict()
    value_count=x_train2[feature].value_counts()
    for i , denominator in value_count.items():
        vec=[]
        for k in range(0,2):
            cnt=x_train2[(x_train2["project_is_approved"]==k) & (x_train2[feature]==i)]
            vec.append((cnt.shape[0]+alpha*10)/(denominator+alpha*20))
        fea_dict[i]=vec
    return fea_dict

def get_fea2(alpha,feature, df):
    fea_dict=get_fea_dict(alpha,feature)
    value_count=x_train2[feature].value_counts()
    gv_fea = []
    # for every feature values in the given data frame we will check if it is there in the train
data then we will add the feature to gv_fea
    # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gv_fea
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(fea_dict[row[feature]])
        else:
            gv_fea.append([1/2,1/2])
            #gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea
```

In [13]:

```
alpha = 1
# train gene feature
train_cleancat_feature_responseCoding = np.array(get_fea(alpha, "clean_categories", x_train))
# test gene feature
test_cleancat_feature_responseCoding = np.array(get_fea(alpha, "clean_categories", x_test))
# cross validation gene feature
cv_cleancat_feature_responseCoding = np.array(get_fea(alpha, "clean_categories", x_cv))
```

In [18]:

```
alpha = 1
# train gene feature
train_cleansub_feature_responseCoding = np.array(get_fea(alpha, "clean_subcategories", x_train))
# test gene feature
test_cleansub_feature_responseCoding = np.array(get_fea(alpha, "clean_subcategories", x_test))
# cross validation gene feature
cv_cleansub_feature_responseCoding = np.array(get_fea(alpha, "clean_subcategories", x_cv))

alpha = 1
# train gene feature
train_state_feature_responseCoding = np.array(get_fea(alpha, "school_state", x_train))
# test gene feature
test_state_feature_responseCoding = np.array(get_fea(alpha, "school_state", x_test))
# cross validation gene feature
cv_state_feature_responseCoding = np.array(get_fea(alpha, "school_state", x_cv))

alpha = 1
# train gene feature
train_teach_feature_responseCoding = np.array(get_fea(alpha, "teacher_prefix", x_train))
# test gene feature
test_teach_feature_responseCoding = np.array(get_fea(alpha, "teacher_prefix", x_test))
# cross validation gene feature
cv_teach_feature_responseCoding = np.array(get_fea(alpha, "teacher_prefix", x_cv))

alpha = 1
# train gene feature
train_proj_feature_responseCoding = np.array(get_fea(alpha, "project_grade_category", x_train))
# test gene feature
test_proj_feature_responseCoding = np.array(get_fea(alpha, "project_grade_category", x_test))
# cross validation gene feature
cv_proj_feature_responseCoding = np.array(get_fea(alpha, "project_grade_category", x_cv))
```

In [21]:

```
alpha = 1
# train gene feature
train_cleancat_feature_responseCoding2 = np.array(get_fea2(alpha, "clean_categories", x_train2))
# test gene feature
test_cleancat_feature_responseCoding2 = np.array(get_fea2(alpha, "clean_categories", x_test2))
# cross validation gene feature
cv_cleancat_feature_responseCoding2 = np.array(get_fea2(alpha, "clean_categories", x_cv2))


alpha = 1
# train gene feature
train_cleansub_feature_responseCoding2 = np.array(get_fea2(alpha, "clean_subcategories",
x_train2))
# test gene feature
test_cleansub_feature_responseCoding2 = np.array(get_fea2(alpha, "clean_subcategories", x_test2))
# cross validation gene feature
cv_cleansub_feature_responseCoding2 = np.array(get_fea2(alpha, "clean_subcategories", x_cv2))

alpha = 1
# train gene feature
train_state_feature_responseCoding2 = np.array(get_fea2(alpha, "school_state", x_train2))
# test gene feature
test_state_feature_responseCoding2 = np.array(get_fea2(alpha, "school_state", x_test2))
# cross validation gene feature
cv_state_feature_responseCoding2 = np.array(get_fea2(alpha, "school_state", x_cv2))

alpha = 1
# train gene feature
train_teach_feature_responseCoding2 = np.array(get_fea2(alpha, "teacher_prefix", x_train2))
# test gene feature
test_teach_feature_responseCoding2 = np.array(get_fea2(alpha, "teacher_prefix", x_test2))
# cross validation gene feature
cv_teach_feature_responseCoding2 = np.array(get_fea2(alpha, "teacher_prefix", x_cv2))

alpha = 1
# train gene feature
train_proj_feature_responseCoding2 = np.array(get_fea2(alpha, "project_grade_category", x_train2))
# test gene feature
test_proj_feature_responseCoding2 = np.array(get_fea2(alpha, "project_grade_category", x_test2))
# cross validation gene feature
```

```
cv_proj_feature_responseCoding2 = np.array(get_fea2(alpha, "project_grade_category", x_cv2))
```

In [40]:

```
def veccat(x,cat,subcat,state,teach,proj):
    from sklearn.feature_extraction.text import CountVectorizer
    from sklearn.preprocessing import Normalizer
    from scipy.sparse import hstack

    price_scalar =  Normalizer(copy=False,norm='l2')
    price_scalar.fit(x['price'].values.reshape(1,-1)) # finding the mean and standard deviation of
this data



    # Now standardize the data with above maen and variance.
    price_standardized = price_scalar.transform(x['price'].values.reshape(1, -1))
    price_standardized=np.transpose(price_standardized)

    projects_scalar = Normalizer(copy=False,norm='l2')
    projects_scalar.fit(x['teacher_number_of_previously_posted_projects'].values.reshape(1,-1)) # f
inding the mean and standard deviation of this data

    # Now standardize the data with above maen and variance.
    projects_standardized =
projects_scalar.transform(x['teacher_number_of_previously_posted_projects'].values.reshape(1, -1))
    projects_standardized =np.transpose(projects_standardized)

    qty_scalar= Normalizer(copy=False,norm='l2')
    qty_scalar.fit(x['quantity'].values.reshape(1,-1)) # finding the mean and standard deviation of
this data


    # Now standardize the data with above maen and variance.
    qty_standardized = qty_scalar.transform(x['quantity'].values.reshape(1, -1))
    qty_standardized=np.transpose(qty_standardized)

    X1 = np.concatenate((cat, subcat,state,teach,proj,price_standardized,projects_standardized,qty_
standardized),axis=1 )

    print(X1.shape)
    return(X1)
```

In [41]:

```
x=veccat(x_train,train_cleancat_feature_responseCoding,train_cleansub_feature_responseCoding,train_
state_feature_responseCoding,train_teach_feature_responseCoding,train_proj_feature_responseCoding)
t=veccat(x_test,test_cleancat_feature_responseCoding,test_cleansub_feature_responseCoding,test_stat
e_feature_responseCoding,test_teach_feature_responseCoding,test_proj_feature_responseCoding)
cv=veccat(x_cv,cv_cleancat_feature_responseCoding,cv_cleansub_feature_responseCoding,cv_state_featu
re_responseCoding,cv_teach_feature_responseCoding,cv_proj_feature_responseCoding)
```

```
(30000, 13)
(10000, 13)
(10000, 13)
```

In [37]:

```
(30000, 2)
```

In [42]:

```
x2=veccat(x_train2,train_cleancat_feature_responseCoding2,train_cleansub_feature_responseCoding2,t
rain_state_feature_responseCoding2,train_teach_feature_responseCoding2,train_proj_feature_response(
oding2)
t2=veccat(x_test2,test_cleancat_feature_responseCoding2,test_cleansub_feature_responseCoding2,test_
state_feature_responseCoding2,test_teach_feature_responseCoding2,test_proj_feature_responseCoding2
)
cv2=veccat(x_cv2,cv_cleancat_feature_responseCoding2,cv_cleansub_feature_responseCoding2,cv_state_f
eature_responseCoding2,cv_teach_feature_responseCoding2,cv_proj_feature_responseCoding2)
```

```
(12000, 13)
(4000, 13)
(4000, 13)
```

In [0]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

2.3 Make Data Model Ready: encoding eassay, and project_title

In [43]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
import re

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

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

In [44]:

```
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 [45]:

```
def preprocessing(x):
    import nltk
    nltk.download('stopwords')
    from tqdm import tqdm
    preprocessed_essays = []
    # tqdm is for printing the status bar
    for sentence in tqdm(x.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.lower() not in stopwords)
        preprocessed_essays.append(sent.strip())
    return preprocessed_essays
```

In [46]:

```
train_essay=[]
test_essay=[]
cv_essay=[]
train_title=[]
test_title=[]
cv_title=[]
train_essay=preprocessing(x_train['essay'])
test_essay=preprocessing(x_test['essay'])
cv_essay=preprocessing(x_cv['essay'])

train_title=preprocessing(x_train['project_title'])
test_title=preprocessing(x_test['project_title'])
cv_title=preprocessing(x_cv['project_title'])
train_essay2=[]
test_essay2=[]
cv_essay2=[]
train_title2=[]
test_title2=[]
cv_title2=[]
train_essay2=preprocessing(x_train2['essay'])
test_essay2=preprocessing(x_test2['essay'])
cv_essay2=preprocessing(x_cv2['essay'])

train_title2=preprocessing(x_train2['project_title'])
test_title2=preprocessing(x_test2['project_title'])
cv_title2=preprocessing(x_cv2['project_title'])
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
100%|████████████████████████████████████████████████████████████████████████████| 30000/30000
[00:28<00:00, 1058.07it/s]
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
100%|████████████████████████████████████████████████████████████████████████████| 10000/10000
[00:08<00:00, 1248.12it/s]
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
100%|████████████████████████████████████████████████████████████████████████████| 10000/10000
[00:08<00:00, 1249.18it/s]
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
100%|████████████████████████████████████████████████████████████████████████| 30000/30000
[00:01<00:00, 24445.15it/s]
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
100%|████████████████████████████████████████████████████████████████████████| 10000/10000
[00:00<00:00, 24464.90it/s]
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
100%|████████████████████████████████████████████████████████████████████████| 10000/10000
[00:00<00:00, 25078.26it/s]
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
100%|█████████████████████████████████████████████████████████████████████████| 12000/12000
[00:10<00:00, 1199.40it/s]
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
100%|███████████████████████████████████████████████████████████████████████████| 4000/4000
[00:03<00:00, 1232.29it/s]
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
100%|███████████████████████████████████████████████████████████████████████████| 4000/4000
[00:03<00:00, 1158.46it/s]
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
100%|████████████████████████████████████████████████████████████████████████| 12000/12000
[00:00<00:00, 20992.09it/s]
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
100%|██████████████████████████████████████████████████████████████████████████| 4000/4000
[00:00<00:00, 23002.65it/s]
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
100%|██████████████████████████████████████████████████████████████████████████| 4000/4000
[00:00<00:00, 21871.65it/s]
```

In [49]:

```python
from scipy.sparse import hstack
from sklearn.preprocessing import StandardScaler
def bow_text(train_essay, test_essay, cv_essay,train_title,test_title,cv_title,x1,t1,cv1):
    from scipy.sparse import hstack

    vectorizer = CountVectorizer(ngram_range=(1,2),min_df=10,max_features=5000)

    vectorizer.fit(train_essay)

    text_bow=vectorizer.transform(train_essay)

    text_bow1 = vectorizer.transform(test_essay)

    text_bow2 = vectorizer.transform(cv_essay)
    #feature_bow.extend(vectorizer.get_feature_names())
    vectorizer= CountVectorizer()
    vectorizer.fit(train_title)
    title_bow=vectorizer.transform(train_title)

    title_bow1 = vectorizer.transform(test_title)

    title_bow2 = vectorizer.transform(cv_title)

    x1 = hstack((x1,text_bow,title_bow )).tocsr()
    t1= hstack((t1,text_bow1,title_bow1 )).tocsr()
    cv1 = hstack((cv1,text_bow2,title_bow2 )).tocsr()
```

```
    return x1,t1,cv1
```

In [50]:

```
xbow,tbow,cvbow=bow_text(train_essay, test_essay, cv_essay,train_title,test_title,cv_title,x,t,cv)
print(xbow.shape)
print(tbow.shape)
print(cvbow.shape)
```

```
(30000, 14161)
(10000, 14161)
(10000, 14161)
```

In [53]:

```
def tfidf_text(train_essay, test_essay, cv_essay,train_title,test_title,cv_title,x1,t1,cv1):
    from sklearn.feature_extraction.text import TfidfVectorizer
    vectorizer = TfidfVectorizer(ngram_range=(1,2),min_df=10,max_features=5000)
    vectorizer.fit(train_essay)
    #feature_tfidf.extend(vectorizer.get_feature_names())
    text_tfidf=vectorizer.transform(train_essay)
    text_tfidf1 = vectorizer.transform(test_essay)
    text_tfidf2 = vectorizer.transform(cv_essay)
    vectorizer = TfidfVectorizer()
    vectorizer.fit(train_title)

    title_tfidf=vectorizer.transform(train_title)
    title_tfidf1 = vectorizer.transform(test_title)
    title_tfidf2 = vectorizer.transform(cv_title)
    x1 = hstack((x1,text_tfidf,title_tfidf ))
    t1= hstack((t1,text_tfidf1,title_tfidf1 ))
    cv1 = hstack((cv1,text_tfidf2,title_tfidf2 ))



    return x1,t1,cv1
```

In [54]:

```
xtf,ttf,cvtf=tfidf_text(train_essay, test_essay, cv_essay,train_title,test_title,cv_title,x,t,cv)
print(xtf.shape)
print(ttf.shape)
print(cvtf.shape)
```

```
(30000, 14161)
(10000, 14161)
(10000, 14161)
```

In [55]:

```
def avgword2vec(text):

    with open('glove_vectors', 'rb') as f:
        model = pickle.load(f)
        glove_words =  set(model.keys())
    avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    for sentence in tqdm(text):# for each review/sentence
        vector = np.zeros(300) # as word vectors are of zero length
        cnt_words =0; # num of words with a valid vector in the sentence/review
        for word in sentence.split(): # for each word in a review/sentence
            if word in glove_words:
                vector += model[word]
                cnt_words += 1
        if cnt_words != 0:
            vector /= cnt_words
        avg_w2v_vectors.append(vector)
    return avg_w2v_vectors
```

In [60]:

```
def w2v(train_essay, test_essay, cv_essay,train_title,test_title,cv_title,x,t,cv):
```

```
def w2v(train_essay, test_essay, cv_essay,train_title,test_title,cv_title,x,t,cv):

    text_w2v=avgword2vec(train_essay2)
    text_w2v1 = avgword2vec(test_essay2)
    text_w2v2 = avgword2vec(cv_essay2)

    title_w2v=avgword2vec(train_title2)
    title_w2v1 = avgword2vec(test_title2)
    title_w2v2 = avgword2vec(cv_title2)
    x1 = np.concatenate((x,text_w2v,title_w2v ),axis=1)
    t1= np.concatenate((t,text_w2v1,title_w2v1 ),axis=1)
    cv1 = np.concatenate((cv,text_w2v2,title_w2v2 ),axis=1)
    return x1,t1,cv1
```

In [61]:

```
xaw2v,taw2v,cvaw2v=w2v(train_essay2, test_essay2, cv_essay2,train_title2,test_title2,cv_title2,x2,t
2,cv2)
print(xaw2v.shape)
print(taw2v.shape)
print(cvaw2v.shape)
```

```
100%|██████████████████████████████████████████████████| 12000/12000
[00:04<00:00, 2436.59it/s]
100%|████████████████████████████████████████████████████| 4000/4000
[00:01<00:00, 2445.01it/s]
100%|████████████████████████████████████████████████████| 4000/4000
[00:01<00:00, 2451.00it/s]
100%|█████████████████████████████████████████████████| 12000/12000
[00:00<00:00, 36058.27it/s]
100%|███████████████████████████████████████████████████| 4000/4000
[00:00<00:00, 42579.82it/s]
100%|███████████████████████████████████████████████████| 4000/4000
[00:00<00:00, 33634.14it/s]
```

```
(12000, 613)
(4000, 613)
(4000, 613)
```

In [63]:

```
def tfidfw2v(text,dictionary,tfidf_words):

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

    tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    for sentence in tqdm(text): # for each review/sentence
        vector = np.zeros(300) # as word vectors are of zero length
        tf_idf_weight =0; # num of words with a valid vector in the sentence/review
        for word in sentence.split(): # for each word in a review/sentence
            if (word in glove_words) and (word in tfidf_words):
                vec = model[word] # getting the vector for each word
                # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
                tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting
the tfidf value for each word
                vector += (vec * tf_idf) # calculating tfidf weighted w2v
                tf_idf_weight += tf_idf
        if tf_idf_weight != 0:
            vector /= tf_idf_weight
        tfidf_w2v_vectors.append(vector)
    return tfidf_w2v_vectors
```

In [64]:

```
def tfiw2v(train_essay, test_essay, cv_essay,train_title,test_title,cv_title,x,t,cv):

    tfidf_model = TfidfVectorizer()
    tfidf_model.fit(train_essay2)
    # 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())

    text_tfw2v=tfidfw2v(train_essay2,dictionary,tfidf_words)
    text_tfw2v1=tfidfw2v(test_essay2,dictionary,tfidf_words)
    text_tfw2v2=tfidfw2v(cv_essay2,dictionary,tfidf_words)

    tfidf_model.fit(train_title2)
    # 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())
    title_tfw2v=tfidfw2v(train_title2,dictionary,tfidf_words)
    title_tfw2v1=tfidfw2v(test_title2,dictionary,tfidf_words)
    title_tfw2v2=tfidfw2v(cv_title2,dictionary,tfidf_words)
    x1 = np.concatenate((x,text_tfw2v,title_tfw2v ),axis=1)
    t1= np.concatenate((t,text_tfw2v1,title_tfw2v1 ),axis=1)
    cv1 = np.concatenate((cv,text_tfw2v2,title_tfw2v2 ),axis=1)
    return x1,t1,cv1
```

In [65]:

```
xtw2v,ttw2v,cvtw2v=tfiw2v(train_essay2, test_essay2, cv_essay2,train_title2,test_title2,cv_title2,x
2,t2,cv2)
print(xtw2v.shape)
print(ttw2v.shape)
print(cvtw2v.shape)
```

```
100%|████████████████████████████████████████| 12000/12000 [00:
43<00:00, 273.85it/s]
100%|████████████████████████████████████████| 4000/4000
[00:12<00:00, 322.60it/s]
100%|████████████████████████████████████████| 4000/4000
[00:13<00:00, 303.78it/s]
100%|████████████████████████████████████████| 12000/12000
[00:00<00:00, 18761.62it/s]
100%|████████████████████████████████████████| 4000/4000
[00:00<00:00, 22236.03it/s]
100%|████████████████████████████████████████| 4000/4000
[00:00<00:00, 22113.20it/s]
```

```
(12000, 613)
(4000, 613)
(4000, 613)
```

2.4 Applying Random Forest

Apply Random Forest on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instrucations

2.4.1 Applying Random Forests on BOW, SET 1

In [83]:

```
# Please write all the code with proper documentation
def hp(x_train,y_train,x_cv,y_cv):
    alpha = [10, 50, 100, 150, 200, 300, 500, 1000]
    max_depth = [2,3,4,5,6,7,8,9,10]
    scores_auc=[]
    cv_log_error_array = []
    for i in alpha:
        for j in max_depth:
            #print("for n_estimators =", i,"and max depth = ", j)
            clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j,
random_state=42, n_jobs=-1)
            clf.fit(x_train, y_train)
            prob= clf.predict_proba(x_cv)
            prob=prob[:,1]
            scores_auc.append(roc_auc_score(y_cv,prob))
        # Chosen optimal k with the highest AUC score
    index=scores_auc.index(max(scores_auc))
    optimal_alpha = alpha[int(index/10)]
```

```
    optimal_alpha= alpha[int(index/10)]
    optimal_depth= max_depth[index%10]
    clf = RandomForestClassifier(n_estimators=optimal_alpha, criterion='gini', max_depth=optimal_de
pth, random_state=42, n_jobs=-1)
    clf.fit(x_train,y_train)
    # predict probabilities
    prob = clf.predict_proba(x_cv)
    # keep probabilities for the positive outcome only
    prob = prob[:, 1]
    prob1=clf.predict_proba(x_train)
    prob1=prob1[:, 1]
    # calculate roc curve
    fpr, tpr, thresholds = roc_curve(y_cv, prob)
    fpr1, tpr1, thresholds1 = roc_curve(y_train, prob1)
    # plot no skill
    plt.plot([0, 1], [0, 1], linestyle='--')
    # plot the roc curve for the model
    plt.plot(fpr, tpr, marker='.',label="CrossValidated or test ROC")
    plt.plot(fpr1, tpr1, marker='.' , label='Train ROC')
    plt.legend()
    # show the plot
    plt.title("Receiver Operating Characteristics")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.show()
    print("Optimal alpha ",optimal_alpha)
    print("Optimal depth ",optimal_depth)
    print("AUC: ", max(scores_auc))
    fig=plt.figure()
    alpha_plot=[10,10,10,10,10,10,10,10,10, 50,50,50,50,50,50,50,50,50, 100,100,100,100,100,100,100
,100,100, 150,150,150,150,150,150,150,150,150, 200,200,200,200,200,200,200,200,200, 300,300,300,300,
300,300,300,300,300, 500,500,500,500,500,500,500,500,500, 1000,1000,1000,1000,1000,1000,1000,1000,1
000]
    depth_plot=[2,3,4,5,6,7,8,9,10]*8
    ax=fig.add_subplot(111,projection='3d')
    ax.scatter(alpha_plot,depth_plot,scores_auc)
    ax.set_xlabel("n_estimators")
    ax.set_ylabel("max_depth")
    ax.set_zlabel("AUC Score")
    plt.show()
```

In [92]:

```
def test_auc(x_train,y_train,x_test,y_test,est,depth):
    clf = RandomForestClassifier(n_estimators=est, criterion='gini', max_depth=depth, random_state=
42, n_jobs=-1)
    clf.fit(x_train,y_train)
    # predict probabilities
    prob = clf.predict_proba(x_test)
    # keep probabilities for the positive outcome only
    prob = prob[:, 1]
    prob1=clf.predict_proba(x_train)
    prob1=prob1[:, 1]
    # calculate roc curve
    fpr, tpr, thresholds = roc_curve(y_test, prob)
    fpr1, tpr1, thresholds1 = roc_curve(y_train, prob1)
    # plot no skill
    plt.plot([0, 1], [0, 1], linestyle='--')
    # plot the roc curve for the model
    plt.plot(fpr, tpr, marker='.',label=" test ROC")
    plt.plot(fpr1, tpr1, marker='.' , label='Train ROC')
    plt.legend()
    # show the plot
    plt.title("Receiver Operating Characteristics")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.show()
    print(roc_auc_score(y_test,prob))
```

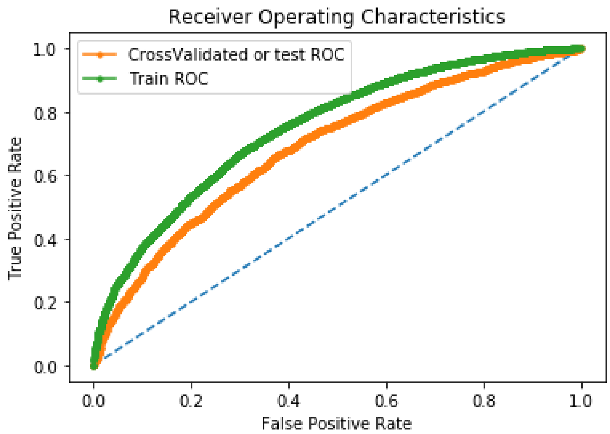
In [99]:

```
# Predictions  on test data

def predict(x_train,y_train,x_test,y_test,alpha,depth):
```

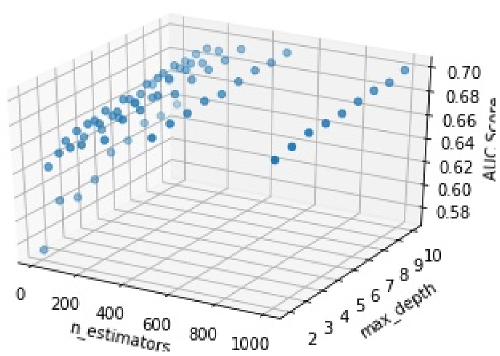
```
def predict(x_train,y_train,x_test,y_test,alpha,depth):
    #from sklearn.naive_bayes import  Multinomialrange
    clf=RandomForestClassifier(n_estimators=alpha, criterion='gini', max_depth=depth, random_state=
42, n_jobs=-1)
    clf.fit(x_train,y_train)
    prob1=clf.predict_proba(x_train)
    prob1=prob1[:,1]
    fpr, tpr, threshould = roc_curve(y_train, prob1)
    t = threshould[np.argmax(tpr*(1-fpr))]
    prob= clf.predict_proba(x_test)
    prob=prob[:,1]


    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    print("train AUC =",str(auc(fpr, tpr)))
    predictions = []
    for i in prob:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [84]:

```
hp(xbow,y_train1,cvbow,y_cv1)
```

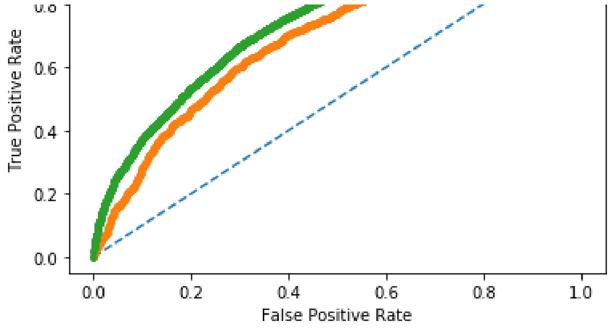



```
Optimal alpha   1000
Optimal depth   3
AUC:   0.6990887043919477
```



In [93]:

```
test_auc(xbow,y_train1,tbow,y_test1,1000,3)
```




```
0.6968920109453491
```

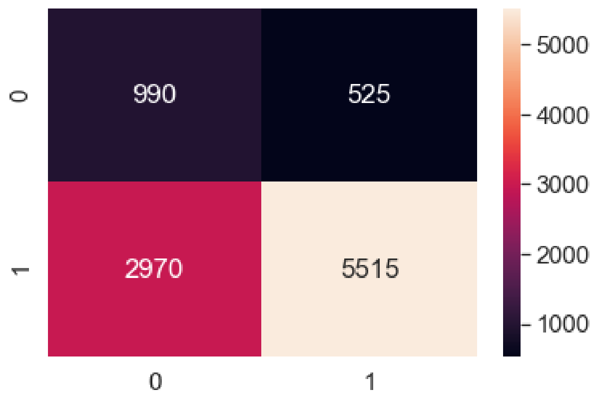
In [100]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
#from sklearn.naive_bayes import  MultinomialNB
labels=[0,1]
print("Test confusion matrix")
cm=confusion_matrix(y_test1, predict(xbow, y_train1, tbow, y_test1,1000,3),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.4678694330205779 for threshold 0.847
train AUC = 0.7510668954606015
```

Out[100]:

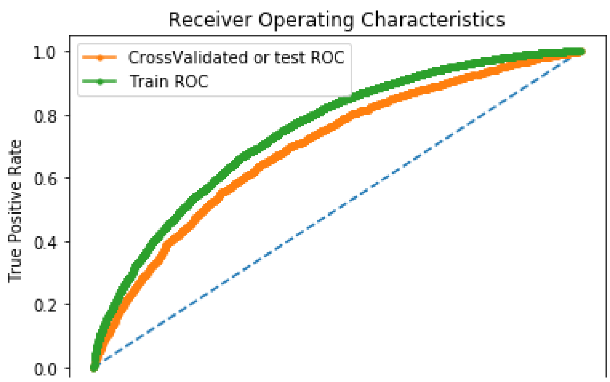
```
<matplotlib.axes._subplots.AxesSubplot at 0x1b0ddc538d0>
```

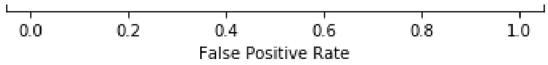


2.4.2 Applying Random Forests on TFIDF, SET 2

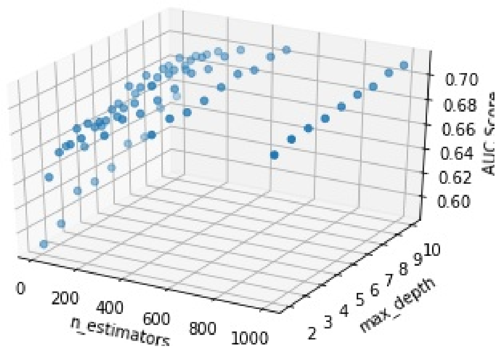
In [85]:

```
# Please write all the code with proper documentation
hp(xtf,y_train1,cvtf,y_cv1)
```



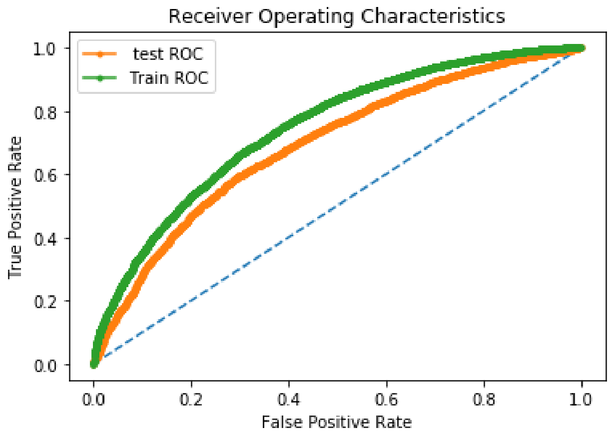



```
Optimal alpha  1000
Optimal depth  2
AUC:  0.7097005587417905
```



In [94]:

```
test_auc(xtf,y_train1,ttf,y_test1,1000,2)
```




```
0.6936569484880132
```

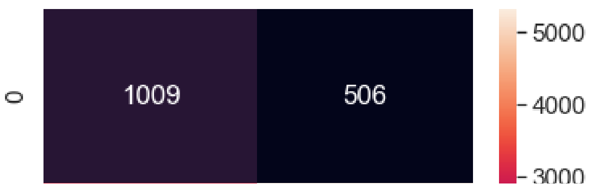
In [101]:

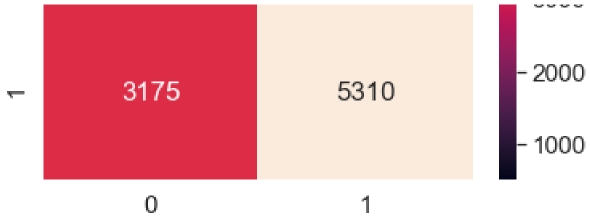
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
#from sklearn.naive_bayes import  MultinomialNB
labels=[0,1]
print("Test confusion matrix")
cm=confusion_matrix(y_test1, predict(xtf, y_train1, ttf, y_test1,1000,2),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.46404058286709576 for threshold 0.848
train AUC = 0.7474849545349169
```

Out[101]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b08130f550>
```

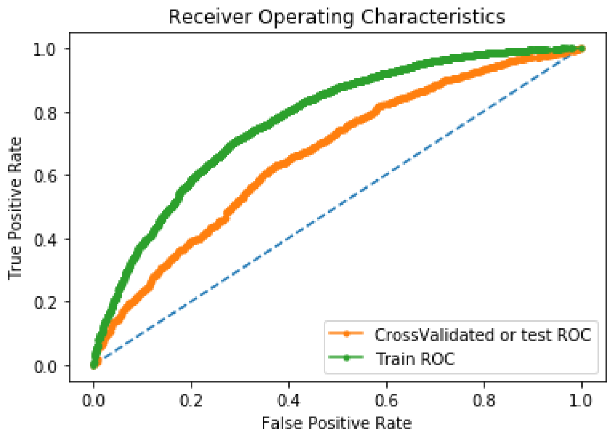




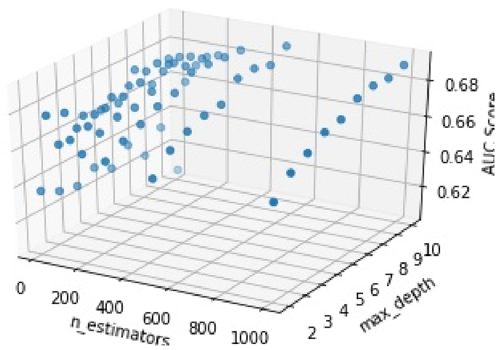
2.4.3 Applying Random Forests on AVG W2V, SET 3

In [87]:

```
# Please write all the code with proper documentation
hp(xaw2v,y_train2,cvaw2v,y_cv2)
```

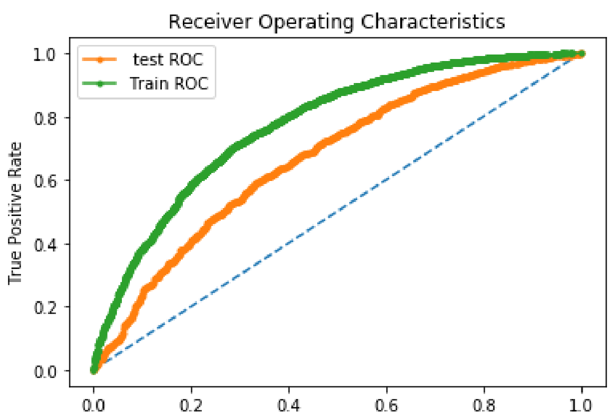



```
Optimal alpha  1000
Optimal depth  3
AUC:  0.6894918025237173
```



In [95]:

```
test_auc(xaw2v,y_train2,taw2v,y_test2,1000,3)
```



False Positive Rate

```
0.6720202711628298
```

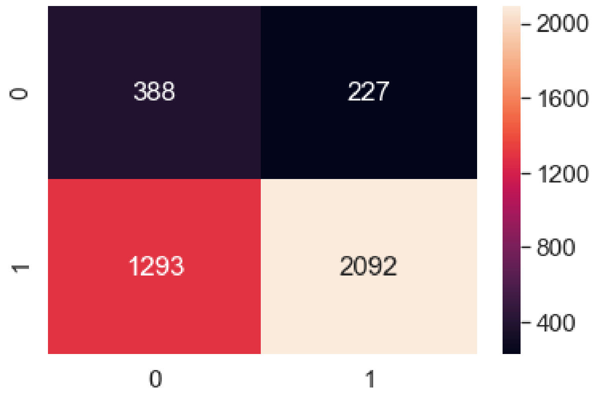
In [102]:

```python
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
#from sklearn.naive_bayes import  MultinomialNB
labels=[0,1]
print("Test confusion matrix")
cm=confusion_matrix(y_test2, predict(xaw2v, y_train2, taw2v, y_test2,1000,3),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.50192581375741 for threshold 0.842
train AUC = 0.7751250461207266
```

Out[102]:

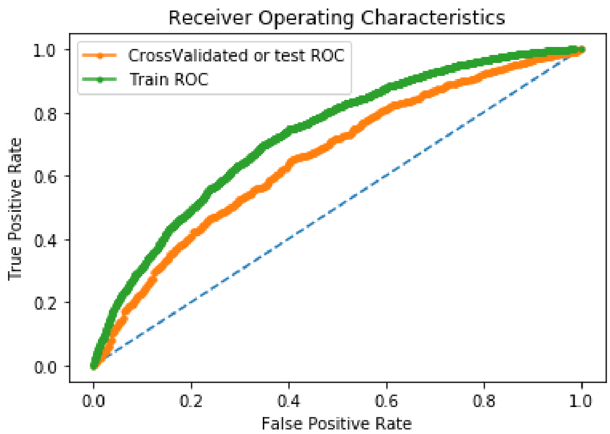
```
<matplotlib.axes._subplots.AxesSubplot at 0x1b08303c588>
```



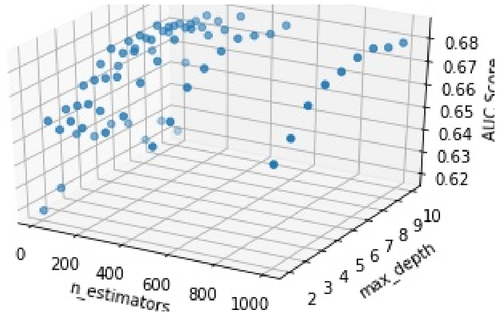
2.4.4 Applying Random Forests on TFIDF W2V, SET 4

In [88]:

```python
# Please write all the code with proper documentation
hp(xtw2v,y_train2,cvtw2v,y_cv2)
```

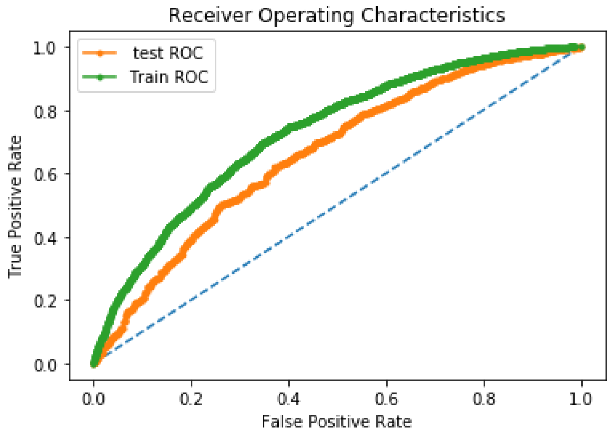



```
Optimal alpha  300
Optimal depth  2
AUC:  0.6839462251696294
```



In [96]:

```
test_auc(xtw2v,y_train2,ttw2v,y_test2,300,2)
```



0.6633574713885987

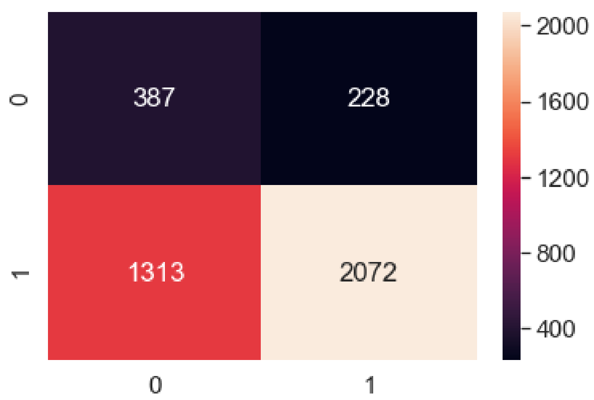
In [103]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
#from sklearn.naive_bayes import  MultinomialNB
labels=[0,1]
print("Test confusion matrix")
cm=confusion_matrix(y_test2, predict(xtw2v, y_train2, ttw2v, y_test2,300,2),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

Test confusion matrix
the maximum value of tpr*(1-fpr) 0.45436379431724366 for threshold 0.841
train AUC = 0.729660786189539

Out[103]:

<matplotlib.axes._subplots.AxesSubplot at 0x1b091bc0ef0>



2.5 Applying GBDT

Apply GBDT on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instrucations

In [110]:

```
def gbdt(x_train,y_train,x_cv,y_cv):
    import xgboost as xgb
    from sklearn.model_selection import GridSearchCV
    from sklearn.model_selection import RandomizedSearchCV
    from sklearn.metrics import roc_curve
    from sklearn.metrics import roc_auc_score
    from sklearn import preprocessing
    scores_auc=[]
    alpha=[10,50,100,150,200,300,500,1000]
    max_depth=[2,3,4,5,6,7,8,9,10]
    gbdt = xgb.XGBClassifier(n_jobs=-1,class_weight='balanced')
    parameters = {'n_estimators': [10,50,100,150,200,300,500,1000], 'max_depth':[2,3,4,5,6,7,8,9,10]
}
    # Fitted the model on train data
    clf = GridSearchCV(gbdt, parameters, cv=2,verbose=5, scoring='roc_auc')
    clf.fit(x_train, y_train)
    prob = clf.predict_proba(x_cv)
    prob = prob[:, 1]
    scores_auc.append(roc_auc_score(y_cv,prob))
    #train_auc= clf.cv_results_['mean_train_score']
    #cv_auc = clf.cv_results_['mean_test_score']
    #max_score= clf.best_score_
    #print("Train AUC:",train_auc)
    #print("CV AUC:",cv_auc)
    index=scores_auc.index(max(scores_auc))
    optimal_alpha = alpha[int(index/10)]
    optimal_depth= max_depth[index%10]
    clf = xgb.XGBClassifier(n_jobs=-1,class_weight='balanced',n_estimators=optimal_alpha,max_depth=
optimal_depth)
#clf = RandomForestClassifier(n_estimators=optimal_alpha, criterion='gini',
max_depth=optimal_depth, random_state=42, n_jobs=-1)
    clf.fit(x_train,y_train)
    # predict probabilities
    prob = clf.predict_proba(x_cv)
    # keep probabilities for the positive outcome only
    prob = prob[:, 1]
    prob1=clf.predict_proba(x_train)
    prob1=prob1[:, 1]
    # calculate roc curve
    fpr, tpr, thresholds = roc_curve(y_cv, prob)
    fpr1, tpr1, thresholds1 = roc_curve(y_train, prob1)
    # plot no skill
    plt.plot([0, 1], [0, 1], linestyle='--')
    # plot the roc curve for the model
    plt.plot(fpr, tpr, marker='.',label="CrossValidated or test ROC")
    plt.plot(fpr1, tpr1, marker='.' , label='Train ROC')
    plt.legend()
    # show the plot
    plt.title("Receiver Operating Characteristics")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.show()
    print("Optimal alpha ",optimal_alpha)
    print("Optimal depth ",optimal_depth)
    print("AUC: ", max(scores_auc))
    fig=plt.figure()
    alpha_plot=[10,10,10,10,10,10,10,10,10, 50,50,50,50,50,50,50,50,50, 100,100,100,100,100,100,100
,100,100, 150,150,150,150,150,150,150,150,150, 200,200,200,200,200,200,200,200,200, 300,300,300,300,
300,300,300,300,300, 500,500,500,500,500,500,500,500,500, 1000,1000,1000,1000,1000,1000,1000,1000,1
000]
    depth_plot=[2,3,4,5,6,7,8,9,10]*8
    ax=fig.add_subplot(111,projection='3d')
    ax.scatter(alpha_plot,depth_plot,scores_auc)
    ax.set_xlabel("n_estimators")
    ax.set_ylabel("max_depth")
    ax.set_zlabel("AUC Score")
    plt.show()
    print("Max Score",max(scores_auc))
```

```
    # Found out the score for crossvalidated data
    #print("Accuracy on crossvalidated data: " , clf.score(x_cv,y_cv))
    #return train_auc,cv_auc
```

In [120]:

```
import xgboost as xgb
def test_auc1(x_train,y_train,x_test,y_test,est,depth):
    clf = xgb.XGBClassifier(n_estimators=est, criterion='gini', max_depth=depth, random_state=42, n
_jobs=-1)
    clf.fit(x_train,y_train)
    # predict probabilities
    prob = clf.predict_proba(x_test)
    # keep probabilities for the positive outcome only
    prob = prob[:, 1]
    prob1=clf.predict_proba(x_train)
    prob1=prob1[:, 1]
    # calculate roc curve
    fpr, tpr, thresholds = roc_curve(y_test, prob)
    fpr1, tpr1, thresholds1 = roc_curve(y_train, prob1)
    # plot no skill
    plt.plot([0, 1], [0, 1], linestyle='--')
    # plot the roc curve for the model
    plt.plot(fpr, tpr, marker='.',label=" test ROC")
    plt.plot(fpr1, tpr1, marker='.' , label='Train ROC')
    plt.legend()
    # show the plot
    plt.title("Receiver Operating Characteristics")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.show()
    print(roc_auc_score(y_test,prob))
```

In [125]:

```
# Predictions  on test data

def predict1(x_train,y_train,x_test,y_test,alpha,depth):
    #from sklearn.naive_bayes import  Multinomialrange
    clf=xgb.XGBClassifier(n_estimators=alpha, criterion='gini', max_depth=depth, random_state=42, n
_jobs=-1)
    clf.fit(x_train,y_train)
    prob1=clf.predict_proba(x_train)
    prob1=prob1[:,1]
    fpr, tpr, threshould = roc_curve(y_train, prob1)
    t = threshould[np.argmax(tpr*(1-fpr))]
    prob= clf.predict_proba(x_test)
    prob=prob[:,1]


    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    print("train AUC =",str(auc(fpr, tpr)))
    predictions = []
    for i in prob:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

2.5.1 Applying XGBOOST on BOW, SET 1

In [111]:

```
# Please write all the code with proper documentation
gbdt(xbow,y_train1,cvbow,y_cv1)
```

```
Fitting 2 folds for each of 72 candidates, totalling 144 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[CV] max_depth=2, n_estimators=10 ....................................
[CV]  max_depth=2, n_estimators=10, score=0.6444332488093235, total=   2.1s
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    2.9s remaining:    0.0s
```

```
[CV] max_depth=2, n_estimators=10 ....................................
[CV]  max_depth=2, n_estimators=10, score=0.6599547320258762, total=   2.0s
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    5.7s remaining:    0.0s
```

```
[CV] max_depth=2, n_estimators=50 ....................................
[CV]  max_depth=2, n_estimators=50, score=0.6974026503063003, total=   4.4s
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:   10.9s remaining:    0.0s
```

```
[CV] max_depth=2, n_estimators=50 ....................................
[CV]  max_depth=2, n_estimators=50, score=0.7034562467909279, total=   4.2s
```

```
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:   15.9s remaining:    0.0s
```

```
[CV] max_depth=2, n_estimators=100 ...................................
[CV]  max_depth=2, n_estimators=100, score=0.713982885271244, total=   7.2s
[CV] max_depth=2, n_estimators=100 ...................................
[CV]  max_depth=2, n_estimators=100, score=0.718522840571082, total=   7.4s
[CV] max_depth=2, n_estimators=150 ...................................
[CV]  max_depth=2, n_estimators=150, score=0.7218827529523464, total=  10.3s
[CV] max_depth=2, n_estimators=150 ...................................
[CV]  max_depth=2, n_estimators=150, score=0.7252595406268536, total=  10.5s
[CV] max_depth=2, n_estimators=200 ...................................
[CV]  max_depth=2, n_estimators=200, score=0.7261185898783651, total=  13.0s
[CV] max_depth=2, n_estimators=200 ...................................
[CV]  max_depth=2, n_estimators=200, score=0.7274911598361381, total=  13.1s
[CV] max_depth=2, n_estimators=300 ...................................
[CV]  max_depth=2, n_estimators=300, score=0.7288655106906367, total=  19.3s
[CV] max_depth=2, n_estimators=300 ...................................
[CV]  max_depth=2, n_estimators=300, score=0.7325969561673942, total=  19.9s
[CV] max_depth=2, n_estimators=500 ...................................
[CV]  max_depth=2, n_estimators=500, score=0.7308195347841291, total=  31.5s
[CV] max_depth=2, n_estimators=500 ...................................
[CV]  max_depth=2, n_estimators=500, score=0.7346893887879444, total=  33.1s
[CV] max_depth=2, n_estimators=1000 ..................................
[CV]  max_depth=2, n_estimators=1000, score=0.7288006895078833, total= 1.1min
[CV] max_depth=2, n_estimators=1000 ..................................
[CV]  max_depth=2, n_estimators=1000, score=0.7343073777952126, total= 1.0min
[CV] max_depth=3, n_estimators=10 ....................................
[CV]  max_depth=3, n_estimators=10, score=0.6619554571101973, total=   2.4s
[CV] max_depth=3, n_estimators=10 ....................................
[CV]  max_depth=3, n_estimators=10, score=0.6735051602433583, total=   2.3s
[CV] max_depth=3, n_estimators=50 ....................................
[CV]  max_depth=3, n_estimators=50, score=0.7054782907651757, total=   5.5s
[CV] max_depth=3, n_estimators=50 ....................................
[CV]  max_depth=3, n_estimators=50, score=0.7130511996396987, total=   5.6s
[CV] max_depth=3, n_estimators=100 ...................................
[CV]  max_depth=3, n_estimators=100, score=0.7225058074523509, total=  11.2s
[CV] max_depth=3, n_estimators=100 ...................................
[CV]  max_depth=3, n_estimators=100, score=0.7239573593153832, total=  12.4s
[CV] max_depth=3, n_estimators=150 ...................................
[CV]  max_depth=3, n_estimators=150, score=0.7273719262068326, total=  16.5s
[CV] max_depth=3, n_estimators=150 ...................................
[CV]  max_depth=3, n_estimators=150, score=0.7307257697070227, total=  16.3s
[CV] max_depth=3, n_estimators=200 ...................................
[CV]  max_depth=3, n_estimators=200, score=0.728201322663353, total=  20.7s
[CV] max_depth=3, n_estimators=200 ...................................
[CV]  max_depth=3, n_estimators=200, score=0.7322480560215472, total=  21.2s
[CV] max_depth=3, n_estimators=300 ...................................
```

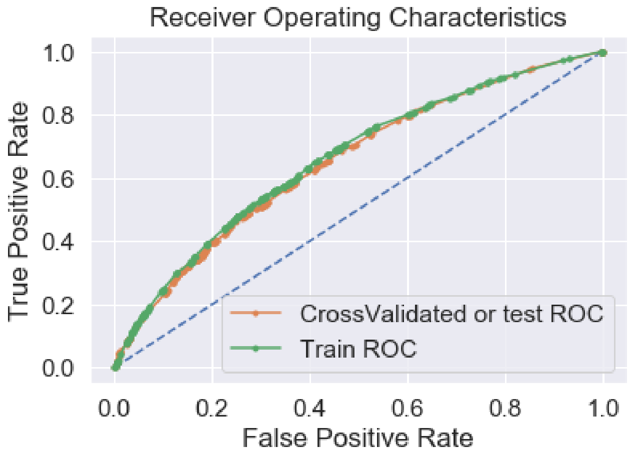
```
[CV]  max_depth=3, n_estimators=300, score=0.7306189850645356, total=  31.4s
[CV] max_depth=3, n_estimators=300 ...................................
[CV]  max_depth=3, n_estimators=300, score=0.7336573332001752, total=  31.7s
[CV] max_depth=3, n_estimators=500 ...................................
[CV]  max_depth=3, n_estimators=500, score=0.7294581343339737, total=  56.0s
[CV] max_depth=3, n_estimators=500 ...................................
[CV]  max_depth=3, n_estimators=500, score=0.7340076338570392, total=  49.3s
[CV] max_depth=3, n_estimators=1000 ..................................
[CV]  max_depth=3, n_estimators=1000, score=0.7281028719259303, total= 1.6min
[CV] max_depth=3, n_estimators=1000 ..................................
[CV]  max_depth=3, n_estimators=1000, score=0.7273816433326548, total= 1.7min
[CV] max_depth=4, n_estimators=10 ....................................
[CV]  max_depth=4, n_estimators=10, score=0.6685881735455157, total=   4.2s
[CV] max_depth=4, n_estimators=10 ....................................
[CV]  max_depth=4, n_estimators=10, score=0.6798794391703774, total=   4.6s
[CV] max_depth=4, n_estimators=50 ....................................
[CV]  max_depth=4, n_estimators=50, score=0.7116688261048061, total=  12.4s
[CV] max_depth=4, n_estimators=50 ....................................
[CV]  max_depth=4, n_estimators=50, score=0.7168134909393509, total=   9.7s
[CV] max_depth=4, n_estimators=100 ...................................
[CV]  max_depth=4, n_estimators=100, score=0.722571060891591, total=  17.4s
[CV] max_depth=4, n_estimators=100 ...................................
[CV]  max_depth=4, n_estimators=100, score=0.7258013654878676, total=  15.5s
[CV] max_depth=4, n_estimators=150 ...................................
[CV]  max_depth=4, n_estimators=150, score=0.7264524820789918, total=  24.6s
[CV] max_depth=4, n_estimators=150 ...................................
[CV]  max_depth=4, n_estimators=150, score=0.7303987244491461, total=  18.2s
[CV] max_depth=4, n_estimators=200 ...................................
[CV]  max_depth=4, n_estimators=200, score=0.7289865252166677, total=  23.2s
[CV] max_depth=4, n_estimators=200 ...................................
[CV]  max_depth=4, n_estimators=200, score=0.7314915207184337, total=  28.5s
[CV] max_depth=4, n_estimators=300 ...................................
[CV]  max_depth=4, n_estimators=300, score=0.7281881820661555, total=  45.0s
[CV] max_depth=4, n_estimators=300 ...................................
[CV]  max_depth=4, n_estimators=300, score=0.7335826046987457, total=  36.8s
[CV] max_depth=4, n_estimators=500 ...................................
[CV]  max_depth=4, n_estimators=500, score=0.7290199818687423, total=  59.6s
[CV] max_depth=4, n_estimators=500 ...................................
[CV]  max_depth=4, n_estimators=500, score=0.729586220576128, total= 1.2min
[CV] max_depth=4, n_estimators=1000 ..................................
[CV]  max_depth=4, n_estimators=1000, score=0.7272410562329035, total= 2.5min
[CV] max_depth=4, n_estimators=1000 ..................................
[CV]  max_depth=4, n_estimators=1000, score=0.722286636123308, total= 2.5min
[CV] max_depth=5, n_estimators=10 ....................................
[CV]  max_depth=5, n_estimators=10, score=0.6765387362673843, total=   2.9s
[CV] max_depth=5, n_estimators=10 ....................................
[CV]  max_depth=5, n_estimators=10, score=0.6805850892398704, total=   3.2s
[CV] max_depth=5, n_estimators=50 ....................................
[CV]  max_depth=5, n_estimators=50, score=0.7118961584363188, total=   9.9s
[CV] max_depth=5, n_estimators=50 ....................................
[CV]  max_depth=5, n_estimators=50, score=0.7219530378570922, total=   8.9s
[CV] max_depth=5, n_estimators=100 ...................................
[CV]  max_depth=5, n_estimators=100, score=0.7217776108845089, total=  14.8s
[CV] max_depth=5, n_estimators=100 ...................................
[CV]  max_depth=5, n_estimators=100, score=0.729248489937899, total=  20.3s
[CV] max_depth=5, n_estimators=150 ...................................
[CV]  max_depth=5, n_estimators=150, score=0.7257105224646117, total=  22.7s
[CV] max_depth=5, n_estimators=150 ...................................
[CV]  max_depth=5, n_estimators=150, score=0.7328070328199556, total=  24.1s
[CV] max_depth=5, n_estimators=200 ...................................
[CV]  max_depth=5, n_estimators=200, score=0.727618087630908, total=  28.3s
[CV] max_depth=5, n_estimators=200 ...................................
[CV]  max_depth=5, n_estimators=200, score=0.7323334353228105, total=  27.2s
[CV] max_depth=5, n_estimators=300 ...................................
[CV]  max_depth=5, n_estimators=300, score=0.7268241707868202, total=  40.1s
[CV] max_depth=5, n_estimators=300 ...................................
[CV]  max_depth=5, n_estimators=300, score=0.7315175944297146, total=  39.9s
[CV] max_depth=5, n_estimators=500 ...................................
[CV]  max_depth=5, n_estimators=500, score=0.7262695165532794, total= 1.1min
[CV] max_depth=5, n_estimators=500 ...................................
[CV]  max_depth=5, n_estimators=500, score=0.7274008182304069, total= 1.1min
[CV] max_depth=5, n_estimators=1000 ..................................
[CV]  max_depth=5, n_estimators=1000, score=0.7245341105005267, total= 2.2min
[CV] max_depth=5, n_estimators=1000 ..................................
[CV]  max_depth=5, n_estimators=1000, score=0.7201378545471446, total= 2.8min
[CV] max_depth=6, n_estimators=10 ....................................
[CV]  max_depth=6, n_estimators=10, score=0.6789066545920717, total=   3.6s
```

```
[CV] max_depth=6, n_estimators=10 ...................................
[CV]  max_depth=6, n_estimators=10, score=0.6818674040432096, total=   3.3s
[CV] max_depth=6, n_estimators=50 ...................................
[CV]  max_depth=6, n_estimators=50, score=0.7168565436854313, total=  12.6s
[CV] max_depth=6, n_estimators=50 ...................................
[CV]  max_depth=6, n_estimators=50, score=0.7181597451222104, total=  10.9s
[CV] max_depth=6, n_estimators=100 ..................................
[CV]  max_depth=6, n_estimators=100, score=0.7247643475956302, total=  24.8s
[CV] max_depth=6, n_estimators=100 ..................................
[CV]  max_depth=6, n_estimators=100, score=0.7249618888100761, total=  22.7s
[CV] max_depth=6, n_estimators=150 ..................................
[CV]  max_depth=6, n_estimators=150, score=0.7267244059896778, total=  36.3s
[CV] max_depth=6, n_estimators=150 ..................................
[CV]  max_depth=6, n_estimators=150, score=0.7267071503107267, total=  31.9s
[CV] max_depth=6, n_estimators=200 ..................................
[CV]  max_depth=6, n_estimators=200, score=0.7264780197922291, total=  44.3s
[CV] max_depth=6, n_estimators=200 ..................................
[CV]  max_depth=6, n_estimators=200, score=0.7260821420114022, total=  44.6s
[CV] max_depth=6, n_estimators=300 ..................................
[CV]  max_depth=6, n_estimators=300, score=0.728343085500748, total=  57.4s
[CV] max_depth=6, n_estimators=300 ..................................
[CV]  max_depth=6, n_estimators=300, score=0.7274358655863529, total=  57.6s
[CV] max_depth=6, n_estimators=500 ..................................
[CV]  max_depth=6, n_estimators=500, score=0.7262281409623674, total= 1.7min
[CV] max_depth=6, n_estimators=500 ..................................
[CV]  max_depth=6, n_estimators=500, score=0.723976776276768, total= 1.6min
[CV] max_depth=6, n_estimators=1000 .................................
[CV]  max_depth=6, n_estimators=1000, score=0.7237848371063907, total= 3.3min
[CV] max_depth=6, n_estimators=1000 .................................
[CV]  max_depth=6, n_estimators=1000, score=0.7195830101207497, total= 3.1min
[CV] max_depth=7, n_estimators=10 ...................................
[CV]  max_depth=7, n_estimators=10, score=0.6795907091274865, total=   3.8s
[CV] max_depth=7, n_estimators=10 ...................................
[CV]  max_depth=7, n_estimators=10, score=0.6830480521208646, total=   3.4s
[CV] max_depth=7, n_estimators=50 ...................................
[CV]  max_depth=7, n_estimators=50, score=0.7173805768694506, total=  15.0s
[CV] max_depth=7, n_estimators=50 ...................................
[CV]  max_depth=7, n_estimators=50, score=0.7203474816529599, total=  12.4s
[CV] max_depth=7, n_estimators=100 ..................................
[CV]  max_depth=7, n_estimators=100, score=0.7256045158838007, total=  33.9s
[CV] max_depth=7, n_estimators=100 ..................................
[CV]  max_depth=7, n_estimators=100, score=0.7272402954614867, total=  30.1s
[CV] max_depth=7, n_estimators=150 ..................................
[CV]  max_depth=7, n_estimators=150, score=0.7273560191681199, total=  28.8s
[CV] max_depth=7, n_estimators=150 ..................................
[CV]  max_depth=7, n_estimators=150, score=0.7303987763199247, total=  28.6s
[CV] max_depth=7, n_estimators=200 ..................................
[CV]  max_depth=7, n_estimators=200, score=0.7274745266065279, total=  50.4s
[CV] max_depth=7, n_estimators=200 ..................................
[CV]  max_depth=7, n_estimators=200, score=0.7308984993991288, total=  52.9s
[CV] max_depth=7, n_estimators=300 ..................................
[CV]  max_depth=7, n_estimators=300, score=0.7274804052947477, total= 1.3min
[CV] max_depth=7, n_estimators=300 ..................................
[CV]  max_depth=7, n_estimators=300, score=0.7270150725416293, total= 1.1min
[CV] max_depth=7, n_estimators=500 ..................................
[CV]  max_depth=7, n_estimators=500, score=0.7279492306802791, total= 1.5min
[CV] max_depth=7, n_estimators=500 ..................................
[CV]  max_depth=7, n_estimators=500, score=0.7236371610002567, total= 1.5min
[CV] max_depth=7, n_estimators=1000 .................................
[CV]  max_depth=7, n_estimators=1000, score=0.7254026693947468, total= 2.9min
[CV] max_depth=7, n_estimators=1000 .................................
[CV]  max_depth=7, n_estimators=1000, score=0.7197656125510132, total= 3.0min
[CV] max_depth=8, n_estimators=10 ...................................
[CV]  max_depth=8, n_estimators=10, score=0.6821410915603616, total=   3.4s
[CV] max_depth=8, n_estimators=10 ...................................
[CV]  max_depth=8, n_estimators=10, score=0.6825275115692584, total=   3.4s
[CV] max_depth=8, n_estimators=50 ...................................
[CV]  max_depth=8, n_estimators=50, score=0.716202487750197, total=  11.8s
[CV] max_depth=8, n_estimators=50 ...................................
[CV]  max_depth=8, n_estimators=50, score=0.7227464187031366, total=  11.6s
[CV] max_depth=8, n_estimators=100 ..................................
[CV]  max_depth=8, n_estimators=100, score=0.7251413444131161, total=  21.7s
[CV] max_depth=8, n_estimators=100 ..................................
[CV]  max_depth=8, n_estimators=100, score=0.7314866448652633, total=  21.4s
[CV] max_depth=8, n_estimators=150 ..................................
[CV]  max_depth=8, n_estimators=150, score=0.7267889505282796, total=  31.9s
[CV] max_depth=8, n_estimators=150 ..................................
```

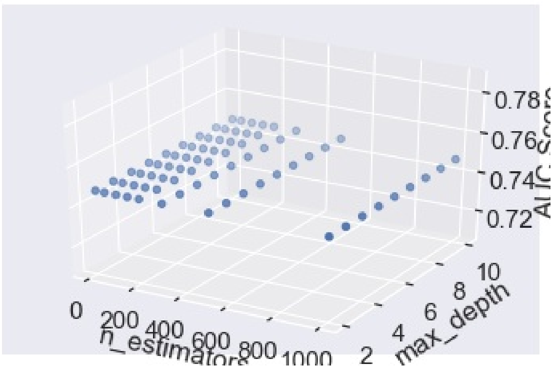
```
[CV]  max_depth=8, n_estimators=150, score=0.7337090310759908, total=  31.2s
[CV] max_depth=8, n_estimators=200 ...................................
[CV]  max_depth=8, n_estimators=200, score=0.7264643431969884, total=  42.5s
[CV] max_depth=8, n_estimators=200 ...................................
[CV]  max_depth=8, n_estimators=200, score=0.7324126247111835, total=  41.2s
[CV] max_depth=8, n_estimators=300 ...................................
[CV]  max_depth=8, n_estimators=300, score=0.7275843716249413, total= 1.0min
[CV] max_depth=8, n_estimators=300 ...................................
[CV]  max_depth=8, n_estimators=300, score=0.7312085483319465, total= 1.0min
[CV] max_depth=8, n_estimators=500 ...................................
[CV]  max_depth=8, n_estimators=500, score=0.7274370759045157, total= 1.7min
[CV] max_depth=8, n_estimators=500 ...................................
[CV]  max_depth=8, n_estimators=500, score=0.7274323383734209, total= 1.7min
[CV] max_depth=8, n_estimators=1000 ..................................
[CV]  max_depth=8, n_estimators=1000, score=0.7234105029888636, total= 3.4min
[CV] max_depth=8, n_estimators=1000 ..................................
[CV]  max_depth=8, n_estimators=1000, score=0.724894768022813, total= 3.4min
[CV] max_depth=9, n_estimators=10 ....................................
[CV]  max_depth=9, n_estimators=10, score=0.6779849281499809, total=   3.8s
[CV] max_depth=9, n_estimators=10 ....................................
[CV]  max_depth=9, n_estimators=10, score=0.6760053144724728, total=   3.7s
[CV] max_depth=9, n_estimators=50 ....................................
[CV]  max_depth=9, n_estimators=50, score=0.7154371862855322, total=  12.8s
[CV] max_depth=9, n_estimators=50 ....................................
[CV]  max_depth=9, n_estimators=50, score=0.7192240643341508, total=  12.8s
[CV] max_depth=9, n_estimators=100 ...................................
[CV]  max_depth=9, n_estimators=100, score=0.7247218827183719, total=  24.4s
[CV] max_depth=9, n_estimators=100 ...................................
[CV]  max_depth=9, n_estimators=100, score=0.7268270409698923, total=  24.7s
[CV] max_depth=9, n_estimators=150 ...................................
[CV]  max_depth=9, n_estimators=150, score=0.7260301156206567, total=  36.1s
[CV] max_depth=9, n_estimators=150 ...................................
[CV]  max_depth=9, n_estimators=150, score=0.7275039546281461, total=  35.4s
[CV] max_depth=9, n_estimators=200 ...................................
[CV]  max_depth=9, n_estimators=200, score=0.7262722484142757, total=  46.8s
[CV] max_depth=9, n_estimators=200 ...................................
[CV]  max_depth=9, n_estimators=200, score=0.7264754435435682, total=  46.2s
[CV] max_depth=9, n_estimators=300 ...................................
[CV]  max_depth=9, n_estimators=300, score=0.7274009392622233, total= 1.1min
[CV] max_depth=9, n_estimators=300 ...................................
[CV]  max_depth=9, n_estimators=300, score=0.7255419251445189, total= 1.1min
[CV] max_depth=9, n_estimators=500 ...................................
[CV]  max_depth=9, n_estimators=500, score=0.7260479072976513, total= 1.9min
[CV] max_depth=9, n_estimators=500 ...................................
[CV]  max_depth=9, n_estimators=500, score=0.7229027053584866, total= 1.9min
[CV] max_depth=9, n_estimators=1000 ..................................
[CV]  max_depth=9, n_estimators=1000, score=0.7252363716791637, total= 3.7min
[CV] max_depth=9, n_estimators=1000 ..................................
[CV]  max_depth=9, n_estimators=1000, score=0.7212123058511344, total= 3.8min
[CV] max_depth=10, n_estimators=10 ...................................
[CV]  max_depth=10, n_estimators=10, score=0.67967216353985, total=   3.9s
[CV] max_depth=10, n_estimators=10 ...................................
[CV]  max_depth=10, n_estimators=10, score=0.6794741554883987, total=   4.0s
[CV] max_depth=10, n_estimators=50 ...................................
[CV]  max_depth=10, n_estimators=50, score=0.7158319574897531, total=  14.2s
[CV] max_depth=10, n_estimators=50 ...................................
[CV]  max_depth=10, n_estimators=50, score=0.7199487163988013, total=  14.4s
[CV] max_depth=10, n_estimators=100 ..................................
[CV]  max_depth=10, n_estimators=100, score=0.7228162194806173, total=  27.0s
[CV] max_depth=10, n_estimators=100 ..................................
[CV]  max_depth=10, n_estimators=100, score=0.7262560993119307, total=  26.6s
[CV] max_depth=10, n_estimators=150 ..................................
[CV]  max_depth=10, n_estimators=150, score=0.7245608412416674, total=  39.6s
[CV] max_depth=10, n_estimators=150 ..................................
[CV]  max_depth=10, n_estimators=150, score=0.7267654184851409, total=  38.8s
[CV] max_depth=10, n_estimators=200 ..................................
[CV]  max_depth=10, n_estimators=200, score=0.7264366442013173, total=  51.6s
[CV] max_depth=10, n_estimators=200 ..................................
[CV]  max_depth=10, n_estimators=200, score=0.7263565557194519, total=  50.9s
[CV] max_depth=10, n_estimators=300 ..................................
[CV]  max_depth=10, n_estimators=300, score=0.7287720049674224, total= 1.3min
[CV] max_depth=10, n_estimators=300 ..................................
[CV]  max_depth=10, n_estimators=300, score=0.7265878129398642, total= 1.3min
[CV] max_depth=10, n_estimators=500 ..................................
[CV]  max_depth=10, n_estimators=500, score=0.7281091482901179, total= 2.1min
[CV] max_depth=10, n_estimators=500 ..................................
[CV]  max_depth=10, n_estimators=500, score=0.7234239893912501, total= 2.1min
```

```
[CV] max_depth=10, n_estimators=1000 ................................
[CV]  max_depth=10, n_estimators=1000, score=0.7267276219779393, total= 4.1min
[CV] max_depth=10, n_estimators=1000 ................................
[CV]  max_depth=10, n_estimators=1000, score=0.7226768599892883, total= 4.1min
```

```
[Parallel(n_jobs=1)]: Done 144 out of 144 | elapsed: 121.8min finished
```



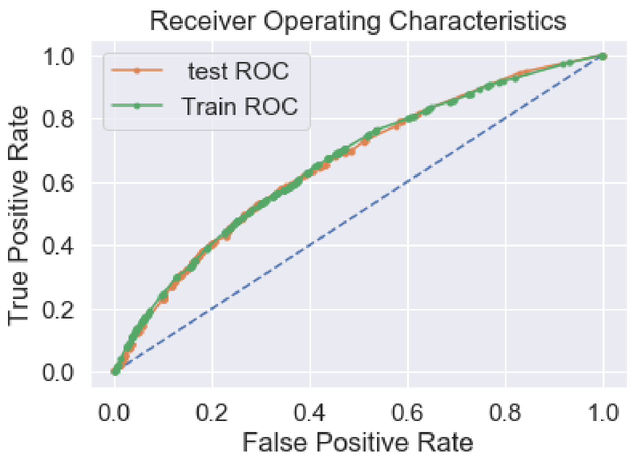

```
Optimal alpha  10
Optimal depth  2
AUC:  0.7476937169262006
```




```
Max_Score 0.7476937169262006
```

In [121]:

```
test_auc1(xbow,y_train1,tbow,y_test1,10,2)
```




```
0.6596134510327876
```

In [127]:

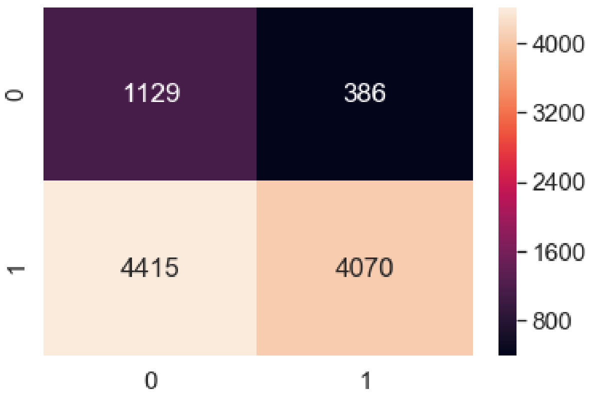
In [127]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
#from sklearn.naive_bayes import  MultinomialNB
labels=[0,1]
print("Test confusion matrix")
cm=confusion_matrix(y_test1, predict1(xbow, y_train1, tbow, y_test1,10,2),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.38378253542705 for threshold 0.717
train AUC = 0.6617544837792468
```

Out[127]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b085118cc0>
```



2.5.2 Applying XGBOOST on TFIDF, SET 2

In [112]:

```
# Please write all the code with proper documentation
gbdt(xtf,y_train1,cvtf,y_cv1)
```

```
Fitting 2 folds for each of 72 candidates, totalling 144 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[CV] max_depth=2, n_estimators=10 ...................................
[CV]  max_depth=2, n_estimators=10, score=0.6429139537096874, total=   2.6s
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    3.3s remaining:    0.0s
```

```
[CV] max_depth=2, n_estimators=10 ...................................
[CV]  max_depth=2, n_estimators=10, score=0.6675699155848036, total=   2.6s
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    6.7s remaining:    0.0s
```

```
[CV] max_depth=2, n_estimators=50 ...................................
[CV]  max_depth=2, n_estimators=50, score=0.6991767692500067, total=   7.7s
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:   15.2s remaining:    0.0s
```

```
[CV] max_depth=2, n_estimators=50 ...................................
[CV]  max_depth=2, n_estimators=50, score=0.7100345853602128, total=   7.7s
```

```
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:   23.6s remaining:    0.0s
```

```
[CV] max_depth=2, n_estimators=100 ..................................
```

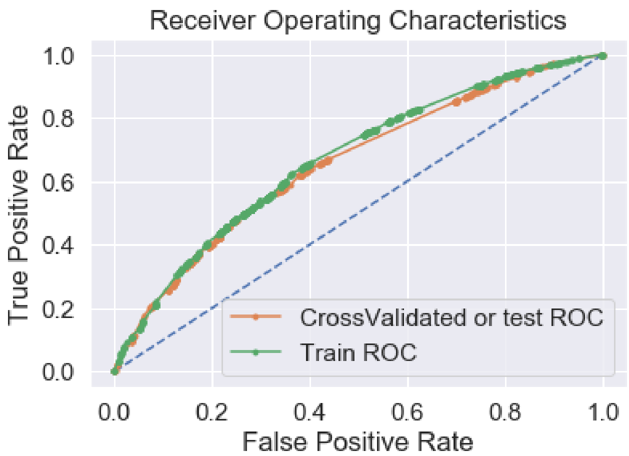
```
[CV]  max_depth=2, n_estimators=100, score=0.7147461810658102, total=  14.2s
[CV] max_depth=2, n_estimators=100 ...................................
[CV]  max_depth=2, n_estimators=100, score=0.72167577125623, total=  14.3s
[CV] max_depth=2, n_estimators=150 ...................................
[CV]  max_depth=2, n_estimators=150, score=0.7223154762761042, total=  20.4s
[CV] max_depth=2, n_estimators=150 ...................................
[CV]  max_depth=2, n_estimators=150, score=0.7258847736995511, total=  20.3s
[CV] max_depth=2, n_estimators=200 ...................................
[CV]  max_depth=2, n_estimators=200, score=0.7252415933375236, total=  26.5s
[CV] max_depth=2, n_estimators=200 ...................................
[CV]  max_depth=2, n_estimators=200, score=0.7271832894760138, total=  26.7s
[CV] max_depth=2, n_estimators=300 ...................................
[CV]  max_depth=2, n_estimators=300, score=0.728853426182489, total=  39.0s
[CV] max_depth=2, n_estimators=300 ...................................
[CV]  max_depth=2, n_estimators=300, score=0.7282158464813077, total=  39.0s
[CV] max_depth=2, n_estimators=500 ...................................
[CV]  max_depth=2, n_estimators=500, score=0.727963132048893, total= 1.1min
[CV] max_depth=2, n_estimators=500 ...................................
[CV]  max_depth=2, n_estimators=500, score=0.7294438352893919, total= 1.1min
[CV] max_depth=2, n_estimators=1000 ..................................
[CV]  max_depth=2, n_estimators=1000, score=0.724134532604173, total=156.4min
[CV] max_depth=2, n_estimators=1000 ..................................
[CV]  max_depth=2, n_estimators=1000, score=0.7245466632289019, total= 2.3min
[CV] max_depth=3, n_estimators=10 ....................................
[CV]  max_depth=3, n_estimators=10, score=0.6668791178481954, total=   3.5s
[CV] max_depth=3, n_estimators=10 ....................................
[CV]  max_depth=3, n_estimators=10, score=0.6824549443502625, total=   3.6s
[CV] max_depth=3, n_estimators=50 ....................................
[CV]  max_depth=3, n_estimators=50, score=0.7096995347122016, total=  10.5s
[CV] max_depth=3, n_estimators=50 ....................................
[CV]  max_depth=3, n_estimators=50, score=0.7177153162927913, total=  10.6s
[CV] max_depth=3, n_estimators=100 ...................................
[CV]  max_depth=3, n_estimators=100, score=0.7229731631658272, total=  21.0s
[CV] max_depth=3, n_estimators=100 ...................................
[CV]  max_depth=3, n_estimators=100, score=0.7254503559303653, total=  21.4s
[CV] max_depth=3, n_estimators=150 ...................................
[CV]  max_depth=3, n_estimators=150, score=0.7252966801041952, total=  38.0s
[CV] max_depth=3, n_estimators=150 ...................................
[CV]  max_depth=3, n_estimators=150, score=0.7263232719699719, total=  34.3s
[CV] max_depth=3, n_estimators=200 ...................................
[CV]  max_depth=3, n_estimators=200, score=0.7249320631124903, total=  45.3s
[CV] max_depth=3, n_estimators=200 ...................................
[CV]  max_depth=3, n_estimators=200, score=0.7283787207255159, total=  45.7s
[CV] max_depth=3, n_estimators=300 ...................................
[CV]  max_depth=3, n_estimators=300, score=0.7258839783476155, total= 1.3min
[CV] max_depth=3, n_estimators=300 ...................................
[CV]  max_depth=3, n_estimators=300, score=0.7282409000672798, total= 1.2min
[CV] max_depth=3, n_estimators=500 ...................................
[CV]  max_depth=3, n_estimators=500, score=0.7236674189543295, total= 2.3min
[CV] max_depth=3, n_estimators=500 ...................................
[CV]  max_depth=3, n_estimators=500, score=0.7270799110146423, total= 2.2min
[CV] max_depth=3, n_estimators=1000 ..................................
[CV]  max_depth=3, n_estimators=1000, score=0.7201978171669869, total= 3.8min
[CV] max_depth=3, n_estimators=1000 ..................................
[CV]  max_depth=3, n_estimators=1000, score=0.7197204331030179, total= 3.6min
[CV] max_depth=4, n_estimators=10 ....................................
[CV]  max_depth=4, n_estimators=10, score=0.6764529938706723, total=   4.6s
[CV] max_depth=4, n_estimators=10 ....................................
[CV]  max_depth=4, n_estimators=10, score=0.6838601410276556, total=   4.0s
[CV] max_depth=4, n_estimators=50 ....................................
[CV]  max_depth=4, n_estimators=50, score=0.7171776065135312, total=  16.4s
[CV] max_depth=4, n_estimators=50 ....................................
[CV]  max_depth=4, n_estimators=50, score=0.7217789941052664, total=  16.5s
[CV] max_depth=4, n_estimators=100 ...................................
[CV]  max_depth=4, n_estimators=100, score=0.7233136429553121, total=  29.7s
[CV] max_depth=4, n_estimators=100 ...................................
[CV]  max_depth=4, n_estimators=100, score=0.7271011780337905, total=  28.8s
[CV] max_depth=4, n_estimators=150 ...................................
[CV]  max_depth=4, n_estimators=150, score=0.7223299309330211, total=  43.5s
[CV] max_depth=4, n_estimators=150 ...................................
[CV]  max_depth=4, n_estimators=150, score=0.72706040760196, total=  38.2s
[CV] max_depth=4, n_estimators=200 ...................................
[CV]  max_depth=4, n_estimators=200, score=0.7225214032663928, total= 1.1min
[CV] max_depth=4, n_estimators=200 ...................................
[CV]  max_depth=4, n_estimators=200, score=0.7278538576090421, total=  56.6s
[CV] max_depth=4, n_estimators=300 ...................................
[CV]  max_depth=4, n_estimators=300, score=0.719484300029435, total= 1.4min
```

```
[CV] max_depth=4, n_estimators=300 ..................................
[CV]  max_depth=4, n_estimators=300, score=0.7278330920074185, total= 1.3min
[CV] max_depth=4, n_estimators=500 ..................................
[CV]  max_depth=4, n_estimators=500, score=0.7192358217105903, total= 2.1min
[CV] max_depth=4, n_estimators=500 ..................................
[CV]  max_depth=4, n_estimators=500, score=0.7234657972386488, total= 2.2min
[CV] max_depth=4, n_estimators=1000 .................................
[CV]  max_depth=4, n_estimators=1000, score=0.7144587996631582, total= 4.6min
[CV] max_depth=4, n_estimators=1000 .................................
[CV]  max_depth=4, n_estimators=1000, score=0.7157820750911819, total= 4.2min
[CV] max_depth=5, n_estimators=10 ...................................
[CV]  max_depth=5, n_estimators=10, score=0.6770526373593541, total=   7.3s
[CV] max_depth=5, n_estimators=10 ...................................
[CV]  max_depth=5, n_estimators=10, score=0.6845148885732687, total=   4.5s
[CV] max_depth=5, n_estimators=50 ...................................
[CV]  max_depth=5, n_estimators=50, score=0.715190194929002, total=  16.9s
[CV] max_depth=5, n_estimators=50 ...................................
[CV]  max_depth=5, n_estimators=50, score=0.7236965703317959, total=  17.1s
[CV] max_depth=5, n_estimators=100 ..................................
[CV]  max_depth=5, n_estimators=100, score=0.7202320173002187, total=  38.6s
[CV] max_depth=5, n_estimators=100 ..................................
[CV]  max_depth=5, n_estimators=100, score=0.7291912591790529, total=  32.7s
[CV] max_depth=5, n_estimators=150 ..................................
[CV]  max_depth=5, n_estimators=150, score=0.7209347798963802, total=  55.1s
[CV] max_depth=5, n_estimators=150 ..................................
[CV]  max_depth=5, n_estimators=150, score=0.7304217377845009, total=  55.3s
[CV] max_depth=5, n_estimators=200 ..................................
[CV]  max_depth=5, n_estimators=200, score=0.7210006557849611, total= 1.2min
[CV] max_depth=5, n_estimators=200 ..................................
[CV]  max_depth=5, n_estimators=200, score=0.7314084928924585, total= 1.2min
[CV] max_depth=5, n_estimators=300 ..................................
[CV]  max_depth=5, n_estimators=300, score=0.7201090489748674, total= 1.9min
[CV] max_depth=5, n_estimators=300 ..................................
[CV]  max_depth=5, n_estimators=300, score=0.7308879696311116, total= 1.7min
[CV] max_depth=5, n_estimators=500 ..................................
[CV]  max_depth=5, n_estimators=500, score=0.7163620595548463, total= 2.9min
[CV] max_depth=5, n_estimators=500 ..................................
[CV]  max_depth=5, n_estimators=500, score=0.7257344176031993, total= 2.8min
[CV] max_depth=5, n_estimators=1000 .................................
[CV]  max_depth=5, n_estimators=1000, score=0.7130678155790493, total= 5.4min
[CV] max_depth=5, n_estimators=1000 .................................
[CV]  max_depth=5, n_estimators=1000, score=0.7172305319977692, total= 5.9min
[CV] max_depth=6, n_estimators=10 ...................................
[CV]  max_depth=6, n_estimators=10, score=0.6770871314269968, total=   5.8s
[CV] max_depth=6, n_estimators=10 ...................................
[CV]  max_depth=6, n_estimators=10, score=0.6818540213823797, total=   6.0s
[CV] max_depth=6, n_estimators=50 ...................................
[CV]  max_depth=6, n_estimators=50, score=0.7124245660559839, total=  23.3s
[CV] max_depth=6, n_estimators=50 ...................................
[CV]  max_depth=6, n_estimators=50, score=0.7234131483985624, total=  22.8s
[CV] max_depth=6, n_estimators=100 ..................................
[CV]  max_depth=6, n_estimators=100, score=0.7166100018756474, total=  44.4s
[CV] max_depth=6, n_estimators=100 ..................................
[CV]  max_depth=6, n_estimators=100, score=0.7270183922714476, total=  44.5s
[CV] max_depth=6, n_estimators=150 ..................................
[CV]  max_depth=6, n_estimators=150, score=0.7176740617336956, total= 1.1min
[CV] max_depth=6, n_estimators=150 ..................................
[CV]  max_depth=6, n_estimators=150, score=0.727463495420986, total= 1.1min
[CV] max_depth=6, n_estimators=200 ..................................
[CV]  max_depth=6, n_estimators=200, score=0.7168554889796036, total= 1.4min
[CV] max_depth=6, n_estimators=200 ..................................
[CV]  max_depth=6, n_estimators=200, score=0.7277989437449652, total= 1.4min
[CV] max_depth=6, n_estimators=300 ..................................
[CV]  max_depth=6, n_estimators=300, score=0.7154974947105639, total= 2.1min
[CV] max_depth=6, n_estimators=300 ..................................
[CV]  max_depth=6, n_estimators=300, score=0.7249864928493019, total= 2.1min
[CV] max_depth=6, n_estimators=500 ..................................
[CV]  max_depth=6, n_estimators=500, score=0.7150343232398793, total= 3.4min
[CV] max_depth=6, n_estimators=500 ..................................
[CV]  max_depth=6, n_estimators=500, score=0.7220186716820407, total= 3.6min
[CV] max_depth=6, n_estimators=1000 .................................
[CV]  max_depth=6, n_estimators=1000, score=0.7124975309509477, total= 6.7min
[CV] max_depth=6, n_estimators=1000 .................................
[CV]  max_depth=6, n_estimators=1000, score=0.715719034805154, total= 6.8min
[CV] max_depth=7, n_estimators=10 ...................................
[CV]  max_depth=7, n_estimators=10, score=0.6770594324313258, total=   7.8s
[CV] max_depth=7, n_estimators=10 ...................................
```

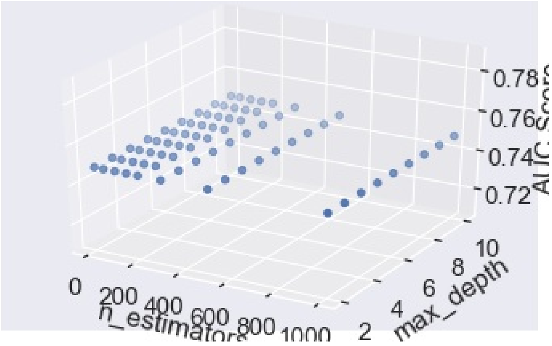
```
[CV]  max_depth=7, n_estimators=10, score=0.6821745827929551, total=   6.5s
[CV] max_depth=7, n_estimators=50 ....................................
[CV]  max_depth=7, n_estimators=50, score=0.7166265832344791, total=  25.3s
[CV] max_depth=7, n_estimators=50 ....................................
[CV]  max_depth=7, n_estimators=50, score=0.7170741934716407, total=  26.9s
[CV] max_depth=7, n_estimators=100 ...................................
[CV]  max_depth=7, n_estimators=100, score=0.720187339269748, total=  49.7s
[CV] max_depth=7, n_estimators=100 ...................................
[CV]  max_depth=7, n_estimators=100, score=0.7222064784804048, total=  50.5s
[CV] max_depth=7, n_estimators=150 ...................................
[CV]  max_depth=7, n_estimators=150, score=0.7196218613337789, total= 1.2min
[CV] max_depth=7, n_estimators=150 ...................................
[CV]  max_depth=7, n_estimators=150, score=0.7243053257872186, total= 1.2min
[CV] max_depth=7, n_estimators=200 ...................................
[CV]  max_depth=7, n_estimators=200, score=0.7178861440563558, total= 1.7min
[CV] max_depth=7, n_estimators=200 ...................................
[CV]  max_depth=7, n_estimators=200, score=0.7236754762152425, total= 1.6min
[CV] max_depth=7, n_estimators=300 ...................................
[CV]  max_depth=7, n_estimators=300, score=0.7161326869727163, total= 2.5min
[CV] max_depth=7, n_estimators=300 ...................................
[CV]  max_depth=7, n_estimators=300, score=0.7228435208003204, total= 2.4min
[CV] max_depth=7, n_estimators=500 ...................................
[CV]  max_depth=7, n_estimators=500, score=0.7165683496405839, total= 4.0min
[CV] max_depth=7, n_estimators=500 ...................................
[CV]  max_depth=7, n_estimators=500, score=0.7196933046859093, total= 4.0min
[CV] max_depth=7, n_estimators=1000 ..................................
[CV]  max_depth=7, n_estimators=1000, score=0.7138846938877134, total= 7.9min
[CV] max_depth=7, n_estimators=1000 ..................................
[CV]  max_depth=7, n_estimators=1000, score=0.7149855647081736, total= 7.3min
[CV] max_depth=8, n_estimators=10 ....................................
[CV]  max_depth=8, n_estimators=10, score=0.6796256873223945, total=   6.1s
[CV] max_depth=8, n_estimators=10 ....................................
[CV]  max_depth=8, n_estimators=10, score=0.679140228707253, total=   6.1s
[CV] max_depth=8, n_estimators=50 ....................................
[CV]  max_depth=8, n_estimators=50, score=0.7172896473948973, total=  25.6s
[CV] max_depth=8, n_estimators=50 ....................................
[CV]  max_depth=8, n_estimators=50, score=0.719772857169731, total=  25.1s
[CV] max_depth=8, n_estimators=100 ...................................
[CV]  max_depth=8, n_estimators=100, score=0.7229381849709192, total=  48.9s
[CV] max_depth=8, n_estimators=100 ...................................
[CV]  max_depth=8, n_estimators=100, score=0.724946621510964, total=  48.3s
[CV] max_depth=8, n_estimators=150 ...................................
[CV]  max_depth=8, n_estimators=150, score=0.7228717039232566, total= 1.2min
[CV] max_depth=8, n_estimators=150 ...................................
[CV]  max_depth=8, n_estimators=150, score=0.7252241301754588, total= 1.2min
[CV] max_depth=8, n_estimators=200 ...................................
[CV]  max_depth=8, n_estimators=200, score=0.7221312312711909, total= 1.6min
[CV] max_depth=8, n_estimators=200 ...................................
[CV]  max_depth=8, n_estimators=200, score=0.7230551362859747, total= 1.6min
[CV] max_depth=8, n_estimators=300 ...................................
[CV]  max_depth=8, n_estimators=300, score=0.7224578615628403, total= 2.3min
[CV] max_depth=8, n_estimators=300 ...................................
[CV]  max_depth=8, n_estimators=300, score=0.7208602070072857, total= 2.5min
[CV] max_depth=8, n_estimators=500 ...................................
[CV]  max_depth=8, n_estimators=500, score=0.7207799110423068, total= 4.6min
[CV] max_depth=8, n_estimators=500 ...................................
[CV]  max_depth=8, n_estimators=500, score=0.7155536534733227, total= 4.5min
[CV] max_depth=8, n_estimators=1000 ..................................
[CV]  max_depth=8, n_estimators=1000, score=0.7182640053868149, total= 8.9min
[CV] max_depth=8, n_estimators=1000 ..................................
[CV]  max_depth=8, n_estimators=1000, score=0.7127138666774374, total= 9.0min
[CV] max_depth=9, n_estimators=10 ....................................
[CV]  max_depth=9, n_estimators=10, score=0.6749466837558981, total=   7.6s
[CV] max_depth=9, n_estimators=10 ....................................
[CV]  max_depth=9, n_estimators=10, score=0.6809680684871328, total=   7.6s
[CV] max_depth=9, n_estimators=50 ....................................
[CV]  max_depth=9, n_estimators=50, score=0.7173405672090367, total=  33.2s
[CV] max_depth=9, n_estimators=50 ....................................
[CV]  max_depth=9, n_estimators=50, score=0.7209723862107276, total=  34.2s
[CV] max_depth=9, n_estimators=100 ...................................
[CV]  max_depth=9, n_estimators=100, score=0.7197445011442003, total= 1.1min
[CV] max_depth=9, n_estimators=100 ...................................
[CV]  max_depth=9, n_estimators=100, score=0.7236864728202654, total= 1.2min
[CV] max_depth=9, n_estimators=150 ...................................
[CV]  max_depth=9, n_estimators=150, score=0.7182910819331452, total= 1.7min
[CV] max_depth=9, n_estimators=150 ...................................
[CV]  max_depth=9, n_estimators=150, score=0.7249899854817149, total= 1.7min
```

```
[CV] max_depth=9, n_estimators=200 ..................................
[CV]  max_depth=9, n_estimators=200, score=0.7179723014192952, total= 2.2min
[CV] max_depth=9, n_estimators=200 ..................................
[CV]  max_depth=9, n_estimators=200, score=0.7244263057327307, total= 2.3min
[CV] max_depth=9, n_estimators=300 ..................................
[CV]  max_depth=9, n_estimators=300, score=0.7171739928493019, total= 3.3min
[CV] max_depth=9, n_estimators=300 ..................................
[CV]  max_depth=9, n_estimators=300, score=0.721969722957481, total= 3.0min
[CV] max_depth=9, n_estimators=500 ..................................
[CV]  max_depth=9, n_estimators=500, score=0.7165304666820851, total= 5.0min
[CV] max_depth=9, n_estimators=500 ..................................
[CV]  max_depth=9, n_estimators=500, score=0.7175199017802605, total= 5.0min
[CV] max_depth=9, n_estimators=1000 .................................
[CV]  max_depth=9, n_estimators=1000, score=0.7140489340624198, total= 9.8min
[CV] max_depth=9, n_estimators=1000 .................................
[CV]  max_depth=9, n_estimators=1000, score=0.7142733789206009, total= 9.0min
[CV] max_depth=10, n_estimators=10 ..................................
[CV]  max_depth=10, n_estimators=10, score=0.6720609221600817, total=  10.6s
[CV] max_depth=10, n_estimators=10 ..................................
[CV]  max_depth=10, n_estimators=10, score=0.6783740281490958, total=   8.7s
[CV] max_depth=10, n_estimators=50 ..................................
[CV]  max_depth=10, n_estimators=50, score=0.7168230524528376, total=  35.4s
[CV] max_depth=10, n_estimators=50 ..................................
[CV]  max_depth=10, n_estimators=50, score=0.7182012417449386, total=  34.8s
[CV] max_depth=10, n_estimators=100 .................................
[CV]  max_depth=10, n_estimators=100, score=0.7186217235649914, total= 1.1min
[CV] max_depth=10, n_estimators=100 .................................
[CV]  max_depth=10, n_estimators=100, score=0.725067013587654l, total= 1.1min
[CV] max_depth=10, n_estimators=150 .................................
[CV]  max_depth=10, n_estimators=150, score=0.7183524450640044, total= 1.5min
[CV] max_depth=10, n_estimators=150 .................................
[CV]  max_depth=10, n_estimators=150, score=0.7241914348480891, total= 1.8min
[CV] max_depth=10, n_estimators=200 .................................
[CV]  max_depth=10, n_estimators=200, score=0.7184754306796152, total= 2.2min
[CV] max_depth=10, n_estimators=200 .................................
[CV]  max_depth=10, n_estimators=200, score=0.724355035283195, total= 2.0min
[CV] max_depth=10, n_estimators=300 .................................
[CV]  max_depth=10, n_estimators=300, score=0.7192407321442799, total= 2.9min
[CV] max_depth=10, n_estimators=300 .................................
[CV]  max_depth=10, n_estimators=300, score=0.7223520797554024, total= 2.9min
[CV] max_depth=10, n_estimators=500 .................................
[CV]  max_depth=10, n_estimators=500, score=0.7178207695852993, total= 4.8min
[CV] max_depth=10, n_estimators=500 .................................
[CV]  max_depth=10, n_estimators=500, score=0.7206786592828498, total= 4.9min
[CV] max_depth=10, n_estimators=1000 ................................
[CV]  max_depth=10, n_estimators=1000, score=0.7155260582192084, total=10.4min
[CV] max_depth=10, n_estimators=1000 ................................
[CV]  max_depth=10, n_estimators=1000, score=0.7191826887432389, total=10.4min
```

```
[Parallel(n_jobs=1)]: Done 144 out of 144 | elapsed: 435.3min finished
```



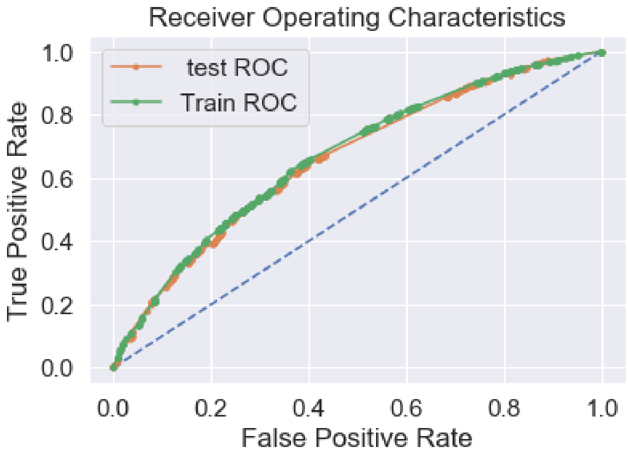

```
Optimal alpha  10
Optimal depth  2
AUC:  0.7484095598717209
```




```
Max_Score 0.7484095598717209
```

In [122]:

```
test_auc1(xtf,y_train1,ttf,y_test1,10,2)
```




```
0.6623374971557262
```

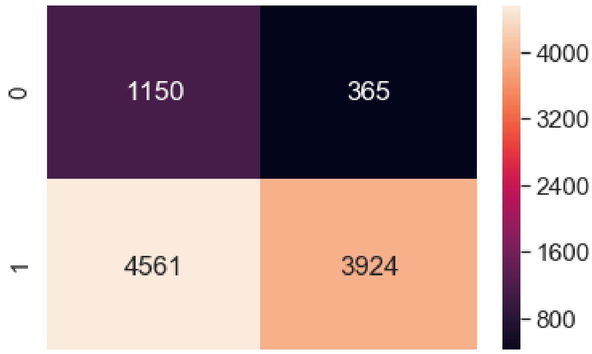
In [128]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
#from sklearn.naive_bayes import  MultinomialNB
labels=[0,1]
print("Test confusion matrix")
cm=confusion_matrix(y_test1, predict1(xtf, y_train1, ttf, y_test1,10,2),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.3964864584762661 for threshold 0.719
train AUC = 0.6707762169783709
```

Out[128]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b080133710>
```



2.5.3 Applying XGBOOST on AVG W2V, SET 3

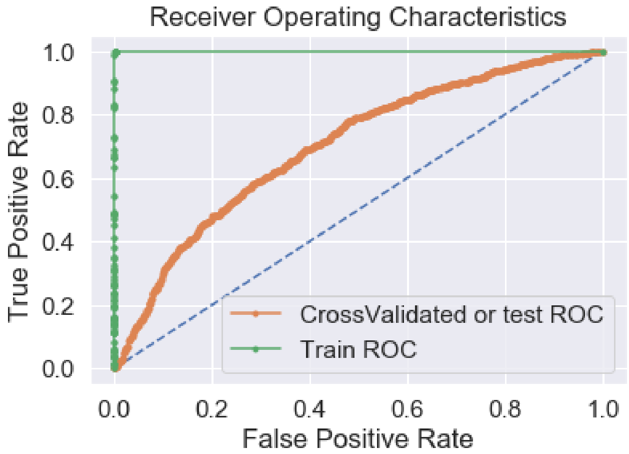
In [116]:

```
def gbdt2(x_train,y_train,x_cv,y_cv):
    import xgboost as xgb
    from sklearn.model_selection import GridSearchCV
    from sklearn.model_selection import RandomizedSearchCV
    from sklearn.metrics import roc_curve
    from sklearn.metrics import roc_auc_score
    from sklearn import preprocessing
    scores_auc=[]
    alpha=[10,50,100,150,200,300,500,1000]
    max_depth=[2,3,4,5,6,7,8,9,10]
    for i in alpha:
        for j in max_depth:

            gbdt = xgb.XGBClassifier(n_jobs=-1,class_weight='balanced',n_estimators=i,max_depth=j)
    #parameters = {'n_estimators': [10,50,100,150,200,300,500,1000], 'max_depth':
[2,3,4,5,6,7,8,9,10]}
    # Fitted the model on train data
    #clf = GridSearchCV(gbdt, parameters, cv=2,verbose=5, scoring='roc_auc')
            gbdt.fit(x_train, y_train)
            prob = gbdt.predict_proba(x_cv)
            prob = prob[:, 1]
            scores_auc.append(roc_auc_score(y_cv,prob))
    #train_auc= clf.cv_results_['mean_train_score']
    #cv_auc = clf.cv_results_['mean_test_score']
    #max_score= clf.best_score_
    #print("Train AUC:",train_auc)
    #print("CV AUC:",cv_auc)
    index=scores_auc.index(max(scores_auc))
    optimal_alpha = alpha[int(index/10)]
    optimal_depth= max_depth[index%10]
    clf = xgb.XGBClassifier(n_jobs=-1,class_weight='balanced',n_estimators=optimal_alpha,max_depth=
optimal_depth)
#clf = RandomForestClassifier(n_estimators=optimal_alpha, criterion='gini',
max_depth=optimal_depth, random_state=42, n_jobs=-1)
    clf.fit(x_train,y_train)
    # predict probabilities
    prob = clf.predict_proba(x_cv)
    # keep probabilities for the positive outcome only
    prob = prob[:, 1]
    prob1=clf.predict_proba(x_train)
    prob1=prob1[:, 1]
    # calculate roc curve
    fpr, tpr, thresholds = roc_curve(y_cv, prob)
    fpr1, tpr1, thresholds1 = roc_curve(y_train, prob1)
    # plot no skill
    plt.plot([0, 1], [0, 1], linestyle='--')
    # plot the roc curve for the model
    plt.plot(fpr, tpr, marker='.',label="CrossValidated or test ROC")
    plt.plot(fpr1, tpr1, marker='.' , label='Train ROC')
    plt.legend()
    # show the plot
    plt.title("Receiver Operating Characteristics")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.show()
    print("Optimal alpha ",optimal_alpha)
    print("Optimal depth ",optimal_depth)
    print("AUC: ", max(scores_auc))
    fig=plt.figure()
    alpha_plot=[10,10,10,10,10,10,10,10,10, 50,50,50,50,50,50,50,50,50, 100,100,100,100,100,100,100
,100,100, 150,150,150,150,150,150,150,150,150, 200,200,200,200,200,200,200,200,200, 300,300,300,300,
300,300,300,300,300, 500,500,500,500,500,500,500,500,500, 1000,1000,1000,1000,1000,1000,1000,1000,1
000]
    depth_plot=[2,3,4,5,6,7,8,9,10]*8
    ax=fig.add_subplot(111,projection='3d')
    ax.scatter(alpha_plot,depth_plot,scores_auc)
    ax.set_xlabel("n_estimators")
    ax.set_ylabel("max_depth")
```

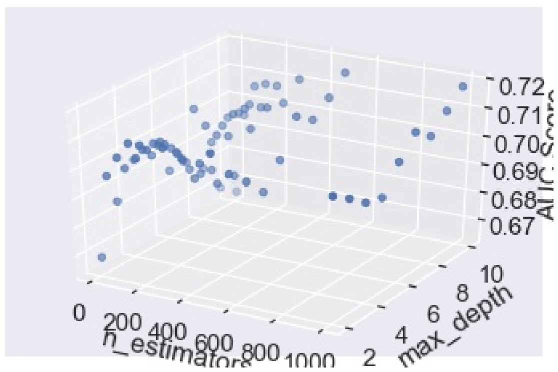
```
    ax.set_zlabel("AUC Score")
    plt.show()
    print("Max_Score",max(scores_auc))


    # Found out the score for crossvalidated data
    #print("Accuracy on crossvalidated data: " , clf.score(x_cv,y_cv))
    #return train_auc,cv_auc
```

In [117]:

```
# Please write all the code with proper documentation

gbdt2(xaw2v,y_train2,cvaw2v,y_cv2)
```



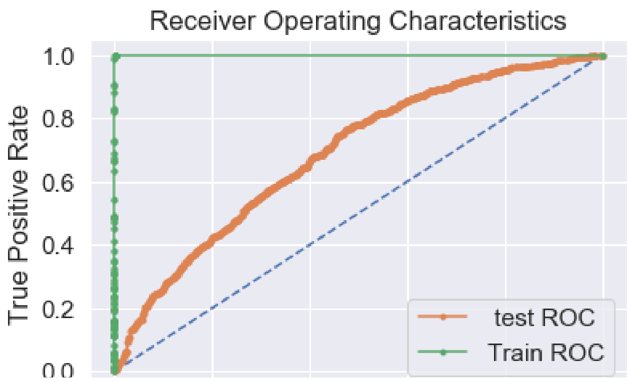

```
Optimal alpha  1000
Optimal depth  3
AUC:  0.717951743882595
```




```
Max_Score 0.717951743882595
```

In [123]:

```
test_auc1(xaw2v,y_train2,taw2v,y_test2,1000,3)
```




```
0.6896369684524024
```

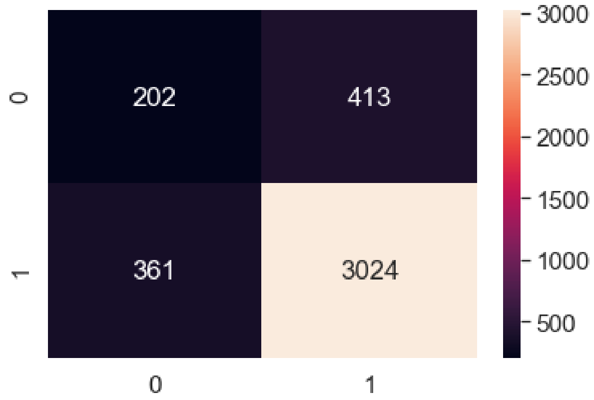
In [129]:

```python
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
#from sklearn.naive_bayes import  MultinomialNB
labels=[0,1]
print("Test confusion matrix")
cm=confusion_matrix(y_test2, predict1(xaw2v, y_train2, taw2v, y_test2,1000,3),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.9986708481369575 for threshold 0.693
train AUC = 0.9999922109588182
```

Out[129]:

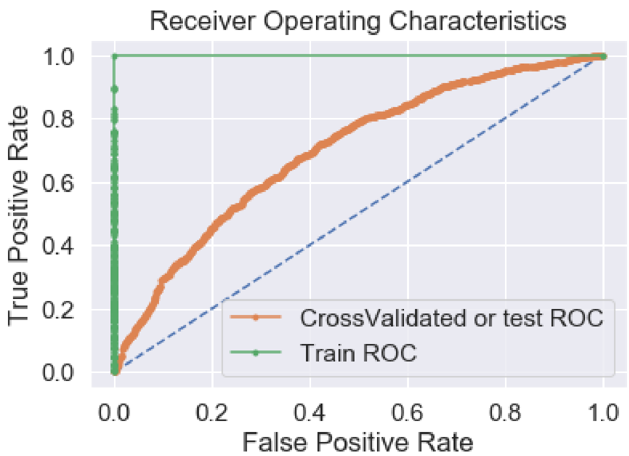
```
<matplotlib.axes._subplots.AxesSubplot at 0x1b0df553d68>
```



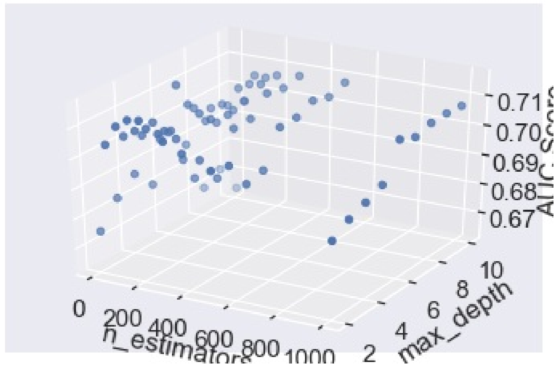
2.5.4 Applying XGBOOST on TFIDF W2V, SET 4

In [119]:

```python
# Please write all the code with proper documentation

gbdt2(xtw2v,y_train2,cvtw2v,y_cv2)
```




```
Optimal alpha  150
Optimal depth  8
AUC:  0.7146757276411533
```




```
Max_Score 0.7146757276411533
```

In [124]:

```
test_auc1(xtw2v,y_train2,ttw2v,y_test2,150,8)
```


```
0.6908316220532958
```

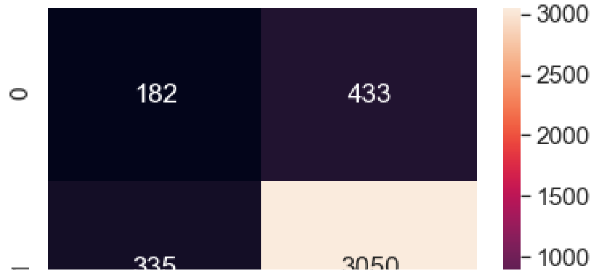
In [130]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
#from sklearn.naive_bayes import  MultinomialNB
labels=[0,1]
print("Test confusion matrix")
cm=confusion_matrix(y_test2, predict1(xtw2v, y_train2, ttw2v, y_test2,150,8),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 1.0 for threshold 0.813
train AUC = 1.0
```

Out[130]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b081f7a358>
```



3. Conclusion

In [1]:

```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
x=PrettyTable()
x.field_names=["Vectorizer","Model","estimator","Depth","Test AUC"]
x.add_row(["BOW","RandomForest","1000","3","0.75" ])
x.add_row(["TFIDF","RandomForest","1000","2","0.693"])
x.add_row(["AVG W2V","RandomForest","1000","3","0.672"])
x.add_row(["TFIDF W2V","RandomForest","300","2","0.663"])
print(x)
```

```
+------------+--------------+-----------+-------+----------+
| Vectorizer |    Model     | estimator | Depth | Test AUC |
+------------+--------------+-----------+-------+----------+
|    BOW     | RandomForest |    1000   |   3   |   0.75   |
|   TFIDF    | RandomForest |    1000   |   2   |  0.693   |
|  AVG W2V   | RandomForest |    1000   |   3   |  0.672   |
| TFIDF W2V  | RandomForest |    300    |   2   |  0.663   |
+------------+--------------+-----------+-------+----------+
```

In [3]:

```
x=PrettyTable()
x.field_names=["Vectorizer","Model","estimator","Depth","Test AUC"]
x.add_row(["BOW","XGBoost","10","2","0.65" ])
x.add_row(["TFIDF","XGBoost","10","2","0.662"])
x.add_row(["AVG W2V","XGBoost","1000","3","0.689"])
x.add_row(["TFIDF W2V","XGBoost","150","8","0.69"])
print(x)
```

```
+------------+---------+-----------+-------+----------+
| Vectorizer |  Model  | estimator | Depth | Test AUC |
+------------+---------+-----------+-------+----------+
|    BOW     | XGBoost |     10    |   2   |   0.65   |
|   TFIDF    | XGBoost |     10    |   2   |  0.662   |
|  AVG W2V   | XGBoost |    1000   |   3   |  0.689   |
| TFIDF W2V  | XGBoost |    150    |   8   |   0.69   |
+------------+---------+-----------+-------+----------+
```

In [ ]: