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*

Feature	Description
`project_essay_3`	Third application essay*
`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 [1]:

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

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

```
import 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
```

1.1 Reading Data

In [2]:

```
import json
from google.colab import files
files.upload()
```

Choose File No file selected

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
```

Out[2]:

```
{'kaggle.json': b'{"username":"sahiltinky","key":"7209ba7120a1763968c676c27cd7f6a2"}'}
```

In [0]:

```
os.environ['KAGGLE_USERNAME'] = 'sahiltinky'
os.environ['KAGGLE_KEY'] = '7209ba7120a1763968c676c27cd7f6a2'
```

In [4]:

```
!kaggle datasets download -d sahiltinky/donorchoose
```

```
Downloading donorchoose.zip to /content
 99% 179M/181M [00:01<00:00, 132MB/s]
100% 181M/181M [00:01<00:00, 107MB/s]
```

In [6]:

```
!unzip /content/donorchoose.zip
```

```
Archive:  /content/donorchoose.zip
  inflating: glove_vectors
  inflating: resources.csv
  inflating: train_data.csv
```

In [2]:

```
# project_data = pd.read_csv('/content/train_data.csv')
# resource_data = pd.read_csv('/content/resources.csv')
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 Rack	1	149.00
1	p069063	Bouncy Bands for Desks (Blue support pipes)	3	14.95

In [5]:

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

1.2 preprocessing of `project_subject_categories`

In [6]:

```
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 & Hunge
```

```
r"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=>
"Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e r
emoving '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 [7]:

```
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 & Hunge
r"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=>
"Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e r
emoving '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 [8]:

```
# 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 [9]:

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

Out[9]:

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

In [10]:

```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

In [11]:

```
# printing some random reviews
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
print(project_data['essay'].values[1000])
print("="*50)
print(project_data['essay'].values[20000])
print("="*50)
print(project_data['essay'].values[99999])
print("="*50)
```

```
My students are English learners that are working on English as their second or third languages. We are
a melting pot of refugees, immigrants, and native-born Americans bringing the gift of language to our s
chool. \r\n\r\n We have over 24 languages represented in our English Learner program with students at e
very level of mastery.  We also have over 40 countries represented with the families within our school.
Each student brings a wealth of knowledge and experiences to us that open our eyes to new cultures, bel
iefs, and respect.\"The limits of your language are the limits of your world.\"-Ludwig Wittgenstein  Ou
r English learner's have a strong support system at home that begs for more resources.  Many times our
parents are learning to read and speak English along side of their children.  Sometimes this creates ba
rriers for parents to be able to help their child learn phonetics, letter recognition, and other readin
g skills.\r\n\r\nBy providing these dvd's and players, students are able to continue their mastery of t
he English language even if no one at home is able to assist.  All families with students within the Le
vel 1 proficiency status, will be a offered to be a part of this program.  These educational videos wil
l be specially chosen by the English Learner Teacher and will be sent home regularly to watch.  The vid
eos are to help the child develop early reading skills.\r\n\r\nParents that do not have access to a dvd
player will have the opportunity to check out a dvd player to use for the year.  The plan is to use the
se videos and educational dvd's for the years to come for other EL students.\r\nnannan
==================================================
The 51 fifth grade students that will cycle through my classroom this year all love learning, at least
most of the time. At our school, 97.3% of the students receive free or reduced price lunch. Of the 560
students, 97.3% are minority students. \r\nThe school has a vibrant community that loves to get togethe
r and celebrate. Around Halloween there is a whole school parade to show off the beautiful costumes tha
t students wear. On Cinco de Mayo we put on a big festival with crafts made by the students, dances, an
d games. At the end of the year the school hosts a carnival to celebrate the hard work put in during th
e school year, with a dunk tank being the most popular activity.My students will use these five brightl
y colored Hokki stools in place of regular, stationary, 4-legged chairs. As I will only have a total of
ten in the classroom and not enough for each student to have an individual one, they will be used in a
variety of ways. During independent reading time they will be used as special chairs students will each
use on occasion. I will utilize them in place of chairs at my small group tables during math and readin
g times. The rest of the day they will be used by the students who need the highest amount of movement
in their life in order to stay focused on school.\r\n\r\nWhenever asked what the classroom is missing,
```

my students always say more Hokki Stools. They can't get their fill of the 5 stools we already have. Wh
en the students are sitting in group with me on the Hokki Stools, they are always moving, but at the sa
me time doing their work. Anytime the students get to pick where they can sit, the Hokki Stools are the
first to be taken. There are always students who head over to the kidney table to get one of the stools
who are disappointed as there are not enough of them. \r\n\r\nWe ask a lot of students to sit for 7 hou
rs a day. The Hokki stools will be a compromise that allow my students to do desk work and move at the
same time. These stools will help students to meet their 60 minutes a day of movement by allowing them
to activate their core muscles for balance while they sit. For many of my students, these chairs will t
ake away the barrier that exists in schools for a child who can't sit still.nannan
==========
How do you remember your days of school? Was it in a sterile environment with plain walls, rows of desk
s, and a teacher in front of the room? A typical day in our room is nothing like that. I work hard to c
reate a warm inviting themed room for my students look forward to coming to each day.\r\n\r\nMy class i
s made up of 28 wonderfully unique boys and girls of mixed races in Arkansas.\r\nThey attend a Title I
school, which means there is a high enough percentage of free and reduced-price lunch to qualify. Our s
chool is an \"open classroom\" concept, which is very unique as there are no walls separating the class
rooms. These 9 and 10 year-old students are very eager learners; they are like sponges, absorbing all t
he information and experiences and keep on wanting more.With these resources such as the comfy red thro
w pillows and the whimsical nautical hanging decor and the blue fish nets, I will be able to help creat
e the mood in our classroom setting to be one of a themed nautical environment. Creating a classroom en
vironment is very important in the success in each and every child's education. The nautical photo prop
s will be used with each child as they step foot into our classroom for the first time on Meet the Teac
her evening. I'll take pictures of each child with them, have them developed, and then hung in our clas
sroom ready for their first day of 4th grade. This kind gesture will set the tone before even the firs
t day of school! The nautical thank you cards will be used throughout the year by the students as they
create thank you cards to their team groups.\r\n\r\nYour generous donations will help me to help make o
ur classroom a fun, inviting, learning environment from day one.\r\n\r\nIt costs lost of money out of m
y own pocket on resources to get our classroom ready. Please consider helping with this project to make
our new school year a very successful one. Thank you!nannan
==========
My kindergarten students have varied disabilities ranging from speech and language delays, cognitive de
lays, gross/fine motor delays, to autism. They are eager beavers and always strive to work their hardes
t working past their limitations. \r\n\r\nThe materials we have are the ones I seek out for my students
. I teach in a Title I school where most of the students receive free or reduced price lunch. Despite
their disabilities and limitations, my students love coming to school and come eager to learn and explo
re.Have you ever felt like you had ants in your pants and you needed to groove and move as you were in
a meeting? This is how my kids feel all the time. The want to be able to move as they learn or so they
say.Wobble chairs are the answer and I love then because they develop their core, which enhances gross
motor and in Turn fine motor skills. \r\nThey also want to learn through games, my kids don't want to s
it and do worksheets. They want to learn to count by jumping and playing. Physical engagement is the ke
y to our success. The number toss and color and shape mats can make that happen. My students will forge
t they are doing work and just have the fun a 6 year old deserves.nannan
==========
The mediocre teacher tells. The good teacher explains. The superior teacher demonstrates. The great tea
cher inspires. -William A. Ward\r\n\r\nMy school has 803 students which is makeup is 97.6% African-Amer
ican, making up the largest segment of the student body. A typical school in Dallas is made up of 23.2%
African-American students. Most of the students are on free or reduced lunch. We aren't receiving docto
rs, lawyers, or engineers children from rich backgrounds or neighborhoods. As an educator I am inspirin
g minds of young children and we focus not only on academics but one smart, effective, efficient, and d
isciplined students with good character.In our classroom we can utilize the Bluetooth for swift transit
ions during class. I use a speaker which doesn't amplify the sound enough to receive the message. Due t
o the volume of my speaker my students can't hear videos or books clearly and it isn't making the lesso
ns as meaningful. But with the bluetooth speaker my students will be able to hear and I can stop, pause
and replay it at any time.\r\nThe cart will allow me to have more room for storage of things that are n
eeded for the day and has an extra part to it I can use. The table top chart has all of the letter, wo
rds and pictures for students to learn about different letters and it is more accessible.nannan
==========

In [12]:

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

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

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

```
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [13]:

```
sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

```
My kindergarten students have varied disabilities ranging from speech and language delays, cognitive de
lays, gross/fine motor delays, to autism. They are eager beavers and always strive to work their hardes
t working past their limitations. \r\n\r\nThe materials we have are the ones I seek out for my students
. I teach in a Title I school where most of the students receive free or reduced price lunch.  Despite
their disabilities and limitations, my students love coming to school and come eager to learn and explo
re.Have you ever felt like you had ants in your pants and you needed to groove and move as you were in
a meeting? This is how my kids feel all the time. The want to be able to move as they learn or so they
say.Wobble chairs are the answer and I love then because they develop their core, which enhances gross
motor and in Turn fine motor skills. \r\nThey also want to learn through games, my kids do not want to
sit and do worksheets. They want to learn to count by jumping and playing. Physical engagement is the k
ey to our success. The number toss and color and shape mats can make that happen. My students will forg
et they are doing work and just have the fun a 6 year old deserves.nannan
==================================================
```

In [14]:

```
# \r \n \t remove from string python: http://texthandler.com/info/remove-line-breaks-python/
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

```
My kindergarten students have varied disabilities ranging from speech and language delays, cognitive de
lays, gross/fine motor delays, to autism. They are eager beavers and always strive to work their hardes
t working past their limitations.     The materials we have are the ones I seek out for my students. I
teach in a Title I school where most of the students receive free or reduced price lunch.  Despite thei
r disabilities and limitations, my students love coming to school and come eager to learn and explore.H
ave you ever felt like you had ants in your pants and you needed to groove and move as you were in a me
eting? This is how my kids feel all the time. The want to be able to move as they learn or so they say.
Wobble chairs are the answer and I love then because they develop their core, which enhances gross moto
r and in Turn fine motor skills.   They also want to learn through games, my kids do not want to sit an
d do worksheets. They want to learn to count by jumping and playing. Physical engagement is the key to
our success. The number toss and color and shape mats can make that happen. My students will forget the
y are doing work and just have the fun a 6 year old deserves.nannan
```

In [15]:

```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

```
My kindergarten students have varied disabilities ranging from speech and language delays cognitive del
ays gross fine motor delays to autism They are eager beavers and always strive to work their hardest wo
rking past their limitations The materials we have are the ones I seek out for my students I teach in a
Title I school where most of the students receive free or reduced price lunch Despite their disabilitie
s and limitations my students love coming to school and come eager to learn and explore Have you ever f
elt like you had ants in your pants and you needed to groove and move as you were in a meeting This is
how my kids feel all the time The want to be able to move as they learn or so they say Wobble chairs ar
e the answer and I love then because they develop their core which enhances gross motor and in Turn fin
e motor skills They also want to learn through games my kids do not want to sit and do worksheets They
want to learn to count by jumping and playing Physical engagement is the key to our success The number
toss and color and shape mats can make that happen My students will forget they are doing work and just
have the fun a 6 year old deserves nannan
```

In [16]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
```

```
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself'
, \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 't
heir',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these',
'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'd
o', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'whil
e', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'bef
ore', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'a
gain', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each
', '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', "doesn
'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 [17]:

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

```
100%|██████████| 109248/109248 [00:46<00:
00, 2334.27it/s]
```

In [18]:

```
# after preprocesing
preprocessed_essays[20000]
```

Out[18]:

```
'kindergarten students varied disabilities ranging speech language delays cognitive delays gross fine m
otor delays autism eager beavers always strive work hardest working past limitations materials ones see
k students teach title school students receive free reduced price lunch despite disabilities limitation
s students love coming school come eager learn explore ever felt like ants pants needed groove move mee
ting kids feel time want able move learn say wobble chairs answer love develop core enhances gross moto
r turn fine motor skills also want learn games kids not want sit worksheets want learn count jumping pl
aying physical engagement key success number toss color shape mats make happen students forget work fun
6 year old deserves nannan'
```

In [19]:

```
# Updating dataframe for clean project title and remove old project title
project_data['clean_essay'] = preprocessed_essays
project_data.drop(['essay'], axis=1, inplace=True)
project_data.head(2)
```

Out[19]:

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

1.4 Preprocessing of `project_title`

In [20]:

```
# similarly you can preprocess the titles also
# Combining all the above stundents
from tqdm import tqdm
preprocessed_title = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['project_title'].values):
    sent = sentance.lower()
    sent = decontracted(sent)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_title.append(sent.strip())
```

```
100%|██████████| 109248/109248 [00:02<00:0
0, 53020.44it/s]
```

In [21]:

```
# after preprocesing
preprocessed_title[20000]
```

Out[21]:

```
'need move input'
```

In [22]:

```
# Updating dataframe for clean project title and remove old project title
project_data['clean_project_title'] = preprocessed_title
project_data.drop(['project_title'], axis=1, inplace=True)
project_data.head(2)
```

Out[22]:

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

Unnamed:	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_c

Preprocessing project_grade

In [23]:

```
# similarly you can preprocess the project_grade also
# Combining all the above stundents
from tqdm import tqdm
preprocessed_grade = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['project_grade_category'].values):
    sent = sentance.lower()
    sent = decontracted(sent)
    sent = sent.replace(' ', '_')
    sent = sent.replace('-', '_')
    # https://gist.github.com/sebleier/554280
#     sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed_grade.append(sent.strip())
```

```
100%|██████████| 109248/109248 [00:00<00:00
, 134234.91it/s]
```

In [24]:

```
preprocessed_grade[:10]
```

Out[24]:

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

In [25]:

```
# Updating dataframe for clean project title and remove old project title
project_data.drop(['project_grade_category'], axis=1, inplace=True)
project_data['project_grade_category'] = preprocessed_grade
project_data.head(2)
```

Out[25]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_essay_1
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	My students are English learners that are work...
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Our students arrive to our school eager to lea...

In [26]:

```
# remove unnecessary column: https://cmdlinetips.com/2018/04/how-to-drop-one-or-more-columns-in-pandas-dataframe/
project_data = project_data.drop(['Unnamed: 0', 'id', 'teacher_id', 'project_submitted_datetime', \
```

```
project_data = project_data.drop(['Unnamed: 0', 'id', 'teacher_id', 'project_submitted_datetime', \
                                  'project_essay_1','project_essay_2','project_essay_3','project_essay_
4', \
                                  'project_resource_summary'], axis=1)
```

In [27]:

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

Out[27]:

	teacher_prefix	school_state	teacher_number_of_previously_posted_projects	project_is_approved	price	quantity	clean_categories
0	Mrs.	IN	0	0	154.6	23	Literacy_Language
1	Mr.	FL	7	1	299.0	1	History_Civics Health_Sports

Check whether each column contain NaN or Not

In [28]:

```
project_data['teacher_prefix'].isnull().values.any()
```

Out[28]:

True

In [29]:

```
project_data['school_state'].isnull().values.any()
```

Out[29]:

False

In [30]:

```
project_data['teacher_number_of_previously_posted_projects'].isnull().values.any()
```

Out[30]:

False

In [31]:

```
project_data['project_is_approved'].isnull().values.any()
```

Out[31]:

False

In [32]:

```
project_data['price'].isnull().values.any()
```

Out[32]:

False

In [33]:

```
project_data['quantity'].isnull().values.any()
```

Out[33]:

False

In [34]:

```
project_data['clean_categories'].isnull().values.any()
```

Out[34]:

False

In [35]:

```
project_data['clean_subcategories'].isnull().values.any()
```

Out[35]:

False

In [36]:

```
project_data['clean_essay'].isnull().values.any()
```

Out[36]:

False

In [37]:

```
project_data['clean_project_title'].isnull().values.any()
```

Out[37]:

False

In [38]:

```
project_data['project_grade_category'].isnull().values.any()
```

Out[38]:

False

Since we got 'teacher prefix' attributes which contain NaN. Let check how many NaN are contain in this attributes

In [39]:

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

Out[39]:

3

1.5 Preparing data for models

In [40]:

```
project_data.columns
```

Out[40]:

```
Index(['teacher_prefix', 'school_state',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'price', 'quantity', 'clean_categories', 'clean_subcategories',
       'clean_essay', 'clean_project_title', 'project_grade_category'],
      dtype='object')
```

we are going to consider

```
       - school_state : categorical data
       - clean_categories : categorical data
       - clean_subcategories : categorical data
       - project_grade_category : categorical data
       - teacher_prefix : categorical data

       - project_title : text data
       - text : text data
       - project_resource_summary: text data (optinal)

       - quantity : numerical (optinal)
       - teacher_number_of_previously_posted_projects : numerical
       - price : numerical
```

1.5.1 Vectorizing Categorical data

- https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/

In [0]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True)
categories_one_hot = vectorizer.fit_transform(project_data['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",categories_one_hot.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds', 'Health_Sp
orts', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig  (109248, 9)
```

In [0]:

```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=True)
sub_categories_one_hot = vectorizer.fit_transform(project_data['clean_subcategories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",sub_categories_one_hot.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular', 'Civics_
Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger', 'SocialSciences', 'Perf
ormingArts', 'CharacterEducation', 'TeamSports', 'Other', 'College_CareerPrep', 'Music', 'History_Geogr
aphy', 'Health_LifeScience', 'EarlyDevelopment', 'ESL', 'Gym_Fitness', 'EnvironmentalScience', 'VisualA
rts', 'Health_Wellness', 'AppliedSciences', 'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Liter
acy']
Shape of matrix after one hot encodig  (109248, 30)
```

In [0]:

In [0]:

```
# you can do the similar thing with state, teacher_prefix and project_grade_category also
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [0]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10)
text_bow = vectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",text_bow.shape)
```

```
Shape of matrix after one hot encodig  (109248, 16623)
```

In [0]:

```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
```

1.5.2.2 TFIDF vectorizer

In [0]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
text_tfidf = vectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",text_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (109248, 16623)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [0]:

```
'''
# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print ("Done.",len(model)," words loaded!")
    return model
model = loadGloveModel('glove.42B.300d.txt')

# ============================
Output:

Loading Glove Model
1917495it [06:32, 4879.69it/s]
Done. 1917495  words loaded!

# ============================

words = []
for i in preproced_texts:
    words.extend(i.split(' '))

for i in preproced_titles:
```

```
    words.extend(i.split(' '))
print("all the words in the coupus", len(words))
words = set(words)
print("the unique words in the coupus", len(words))

inter_words = set(model.keys()).intersection(words)
print("The number of words that are present in both glove vectors and our coupus", \
      len(inter_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")

words_courpus = {}
words_glove = set(model.keys())
for i in words:
    if i in words_glove:
        words_courpus[i] = model[i]
print("word 2 vec length", len(words_courpus))


# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-an
d-load-variables-in-python/

import pickle
with open('glove_vectors', 'wb') as f:
    pickle.dump(words_courpus, f)


'''
```

Out[0]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef loadGloveModel(
gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFile,\'r\', encoding="utf8")\n    mod
el = {}\n    for line in tqdm(f):\n        splitLine = line.split()\n        word = splitLine[0]\n
embedding = np.array([float(val) for val in splitLine[1:]])\n        model[word] = embedding\n    print
("Done.",len(model)," words loaded!")\n    return model\nmodel = loadGloveModel(\'glove.42B.300d.txt\')
\n\n# ============================\nOutput:\n    \nLoading Glove Model\n1917495it [06:32, 4879.69it/s]\
nDone. 1917495  words loaded!\n\n# ============================\n\nwords = []\nfor i in preproced_texts
:\n    words.extend(i.split(\' \'))\n\nfor i in preproced_titles:\n    words.extend(i.split(\' \'))\npr
int("all the words in the coupus", len(words))\nwords = set(words)\nprint("the unique words in the coup
us", len(words))\n\ninter_words = set(model.keys()).intersection(words)\nprint("The number of words tha
t are present in both glove vectors and our coupus",       len(inter_words),"(",np.round(len(inter_word
s)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nwords_glove = set(model.keys())\nfor i in words:\n
if i in words_glove:\n        words_courpus[i] = model[i]\nprint("word 2 vec length", len(words_courpus
))\n\n\n# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to
-save-and-load-variables-in-python/\n\nimport pickle\nwith open(\'glove_vectors\', \'wb\') as f:\n    p
ickle.dump(words_courpus, f)\n\n\n'
```

In [0]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-an
d-load-variables-in-python/
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [0]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays): # 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)

print(len(avg_w2v_vectors))
print(len(avg_w2v_vectors[0]))
```

```
100%|██████████| 109248/109248 [01:06<00:00, 1631.10it/s]
```

```
109248
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [0]:

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

In [0]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays): # 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)

print(len(tfidf_w2v_vectors))
print(len(tfidf_w2v_vectors[0]))
```

```
100%|██████████| 109248/109248 [08:30<00:00, 214.10it/s]
```

```
109248
300
```

In [0]:

```
# Similarly you can vectorize for title also
```

1.5.3 Vectorizing Numerical features

In [0]:

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

In [0]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
```

```
from sklearn.preprocessing import StandardScaler

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.73
5.5 ].
# Reshape your data either using array.reshape(-1, 1)

price_scalar = StandardScaler()
price_scalar.fit(project_data['price'].values.reshape(-1,1)) # finding the mean and standard deviation
of this data
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

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

In [0]:

```
price_standardized
```

Out[0]:

```
array([[0.00098843, 0.00191166, 0.00330448, ..., 0.00153418, 0.00046704,
        0.00070265]])
```

1.5.4 Merging all the above features

- we need to merge all the numerical vectors i.e catogorical, text, numerical vectors

In [0]:

```
print(categories_one_hot.shape)
print(sub_categories_one_hot.shape)
print(text_bow.shape)
print(price_standardized.shape)
```

```
(109248, 9)
(109248, 30)
(109248, 16623)
(109248, 1)
```

In [0]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X = hstack((categories_one_hot, sub_categories_one_hot, text_bow, price_standardized))
X.shape
```

Out[0]:

```
(109248, 16663)
```

In [0]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# 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
```

Computing Sentiment Scores

In [0]:

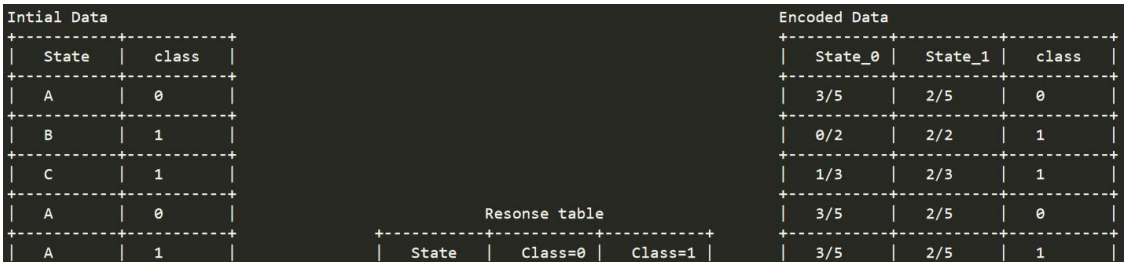
```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# import nltk
# nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest students with t
he biggest enthusiasm \
for learning my students learn in many different ways using all of our senses and multiple intelligence
s i use a wide range\
of techniques to help all my students succeed students in my class come from a variety of different bac
kgrounds which makes\
for wonderful sharing of experiences and cultures including native americans our school is a caring com
munity of successful \
learners which can be seen through collaborative student project based learning in and out of the class
room kindergarteners \
in my class love to work with hands on materials and have many different opportunities to practice a sk
ill before it is\
mastered having the social skills to work cooperatively with friends is a crucial aspect of the kinderg
arten curriculum\
montana is the perfect place to learn about agriculture and nutrition my students love to role play in
our pretend kitchen\
in the early childhood classroom i have had several kids ask me can we try cooking with real food i wil
l take their idea \
and create common core cooking lessons where we learn important math and writing concepts while cooking
delicious healthy \
food for snack time my students will have a grounded appreciation for the work that went into making th
e food and knowledge \
of where the ingredients came from as well as how it is healthy for their bodies this project would exp
and our learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade applesauce
make our own bread \
and mix up healthy plants from our classroom garden in the spring we will also create our own cookbooks
to be printed and \
shared with families students will gain math and literature skills as well as a life long enjoyment for
healthy cooking \
nannan'
ss = sid.polarity_scores(for_sentiment)

for k in ss:
    print('{0}: {1}, '.format(k, ss[k]), end='')

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

```
D:\installed\Anaconda3\lib\site-packages\nltk\twitter\__init__.py:20: UserWarning:

The twython library has not been installed. Some functionality from the twitter package will not be ava
ilable.
```

```
neg: 0.01, neu: 0.745, pos: 0.245, compound: 0.9975,
```

Assignment 9: RF and GBDT

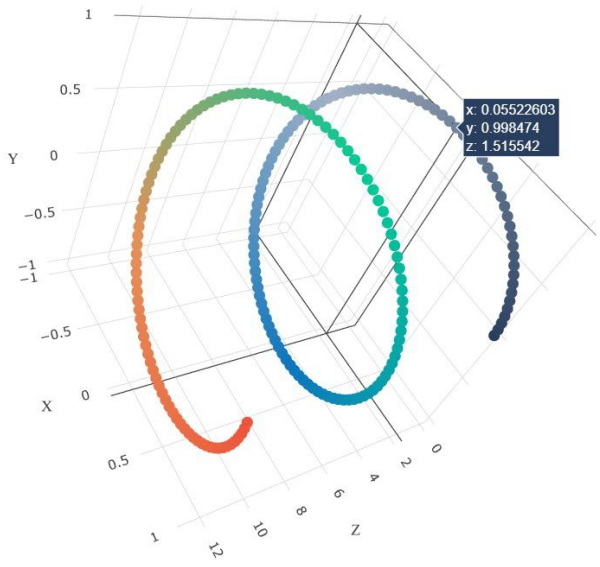
Response Coding: Example



B	1		A	3	2	0/2	2/2	1
A	0		B	0	2	3/5	2/5	0
A	1		C	1	2	3/5	2/5	1
C	1					1/3	2/3	1
C	0					1/3	2/3	0

> 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, 0.05]

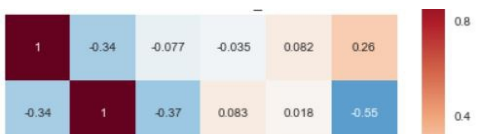
1. **Apply both Random Forrest and GBDT on these feature sets**
   - Set 1: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(BOW) + preprocessed_eassay (BOW)
   - Set 2: categorical(instead of one hot encoding, try response coding: 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)
   - Set 4: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)
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 k-fold cross validation/simple cross validation data
   - use gridsearch cv or randomsearch cv or 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

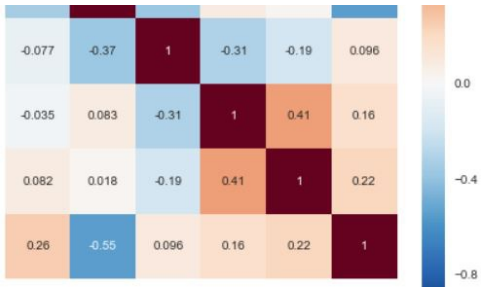


with X-axis as **n_estimators**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

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

	Predicted: NO	Predicted: YES
Actual: NO	TN = ??	FP = ??
Actual: YES	FN = ??	TP = ??

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 [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
# 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
```

In [41]:

```
# Combine the train.csv and resource.csv
# https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexes-for-all-groups-in-one-step
from sklearn.model_selection import train_test_split

# https://www.geeksforgeeks.org/python-pandas-dataframe-sample/
# Take 50k dataset
project_data = project_data.sample(n=50000)
```

```
project_data = project_data.sample(n=50000)
# Remove that row which contain NaN. We observed that only 3 rows that contain NaN
project_data = project_data[pd.notnull(project_data['teacher_prefix'])]
project_data.shape
```

Out[41]:

```
(49998, 11)
```

In [133]:

```
# Split train and test
tr_X, ts_X, tr_y, ts_y, = train_test_split(project_data, project_data['project_is_approved'].values, te
st_size=0.33, random_state=1, stratify=project_data['project_is_approved'].values)
tr_X = tr_X.reset_index(drop=True)
ts_X = ts_X.reset_index(drop=True)

# After train data, We are going to perform KFold Cross validation at the time of training model

# Reset index of df
tr_X = tr_X.reset_index(drop=True)
ts_X = ts_X.reset_index(drop=True)
tr_X.drop(['project_is_approved'], axis=1, inplace=True)
ts_X.drop(['project_is_approved'], axis=1, inplace=True)

print('Shape of train data:', tr_X.shape)
print('Shape of test data:', ts_X.shape)
```

```
Shape of train data: (33498, 10)
Shape of test data: (16500, 10)
```

In [134]:

```
print('Shape of Train Data',[tr_X.shape, tr_y.shape])
print('Shape of Test Data',[ts_X.shape, ts_y.shape])
```

```
Shape of Train Data [(33498, 10), (33498,)]
Shape of Test Data [(16500, 10), (16500,)]
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [44]:

```
# 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
```

In [45]:

```
# No need for Feature scaling in numerical features on DT base model
```

For Categorical Features

In [46]:

```
from prettytable import PrettyTable
```

```
def Response_Code_Fit(cat_list,X,y,toprint=False):
        # Create array for binary label w.r.t each categorical features
    pos_count = np.zeros(len(cat_list))
    neg_count = np.zeros(len(cat_list))

    # Create for Response Table
    for j in tqdm(range(len(X))):
        for i in range(len(cat_list)):
            if X[j] == cat_list[i] and tr_y[j] == 1:
                pos_count[i] += 1
            if X[j] == cat_list[i] and tr_y[j] == 0:
                neg_count[i] += 1

    # Final state table
    state = np.zeros((len(cat_list),2))
    for i in tqdm(range(len(cat_list))):
        state[i][0] = neg_count[i] / (neg_count[i]+pos_count[i])
        state[i][1] = pos_count[i] / (neg_count[i]+pos_count[i])

    if toprint:
        x = PrettyTable()
        x.field_names = ['Feature','Class 0','Class 1','Total','Train Total','State0','State1']
        for i in range(len(cat_list)):
            x.add_row([cat_list[i],neg_count[i],pos_count[i],neg_count[i]+pos_count[i],X[X==cat_list[i]
].count(), \
                       state[i][0], state[i][1]])

        print(x)

    # Store Category value as key item in dict() and their prob value of binary class as a values in di
ct()
    # Ex: {'TX': [0.5,0.5], 'WA': [0.65,0.65],....}
    # Dict() will help us for fast indexing at the time when transform operation revoke
    rt = dict()
    for i in range(len(cat_list)):
        rt[cat_list[i]] = [state[i][0], state[i][1]]

    return rt

def Response_Code_Transform(vectorizer,X,y):
    X_ = []

    # Loop each data points in X
    for i in tqdm(range(len(X))):
        # To check whether this feature in X contain in vectorizer or not
        if X[i] in vectorizer:
            # If it is present, call the dict() as a key value by each datapoint X[i] feature to fetch
directly rather than
            # looping
            X_.append(vectorizer[X[i]])
        else:
            # If not, set default value as [0.5,0.5]
            X_.append([0.5,0.5])

    return np.array(X_)
```

school state feature

In [47]:

```
# on Train Data
# NOTE: I just show the result table for this feature only to get the known of prob. values of binary c
lass
cat_list = tr_X['school_state'].unique()
vectorizer = Response_Code_Fit(cat_list,tr_X['school_state'], tr_y, toprint=True)
```

```
100%|██████████| 33498/33498 [00:21<00:
00, 1534.14it/s]
100%|██████████| 51/51
[00:00<?, ?it/s]
```

```
+---------+---------+---------+--------+-------------+---------------------+--------------------+
```

Feature	Class 0	Class 1	Total	Train Total	State0	State1
ID	42.0	175.0	217.0	217	0.1935483870967742	0.8064516129032258
TX	422.0	1788.0	2210.0	2210	0.19095022624434388	0.8090497737556561
FL	309.0	1552.0	1861.0	1861	0.1660397635679742	0.8339602364320258
OH	98.0	673.0	771.0	771	0.12710765239948119	0.8728923476005188
AR	66.0	263.0	329.0	329	0.2006079027355623	0.7993920972644377
PA	146.0	833.0	979.0	979	0.1491317671092952	0.8508682328907048
OK	110.0	583.0	693.0	693	0.15873015873015872	0.8412698412698413
CA	634.0	4086.0	4720.0	4720	0.13432203389830508	0.8656779661016949
AL	74.0	495.0	569.0	569	0.13005272407732865	0.8699472759226714
TN	79.0	473.0	552.0	552	0.1431159420289855	0.8568840579710145
ND	4.0	41.0	45.0	45	0.08888888888888889	0.9111111111111111
WA	88.0	579.0	667.0	667	0.13193403298350825	0.8680659670164917
NY	318.0	1947.0	2265.0	2265	0.1403973509933775	0.8596026490066225
MI	160.0	795.0	955.0	955	0.16753926701570682	0.8324607329842932
CT	51.0	462.0	513.0	513	0.09941520467836257	0.9005847953216374
MO	108.0	684.0	792.0	792	0.13636363636363635	0.8636363636363636
SC	172.0	1064.0	1236.0	1236	0.13915857605177995	0.86084142394822
NJ	111.0	554.0	665.0	665	0.16691729323308271	0.8330827067669173
IL	207.0	1207.0	1414.0	1414	0.1463932107496464	0.8536067892503536
CO	55.0	300.0	355.0	355	0.15492957746478872	0.8450704225352113
LA	116.0	639.0	755.0	755	0.15364238410596026	0.8463576158940397
AZ	106.0	540.0	646.0	646	0.16408668730650156	0.8359133126934984
KY	55.0	323.0	378.0	378	0.1455026455026455	0.8544973544973545
UT	83.0	450.0	533.0	533	0.15572232645403378	0.8442776735459663
OR	58.0	336.0	394.0	394	0.14720812182741116	0.8527918781725888
MD	74.0	387.0	461.0	461	0.16052060737527116	0.8394793926247288
NC	220.0	1320.0	1540.0	1540	0.14285714285714285	0.8571428571428571
WV	18.0	117.0	135.0	135	0.13333333333333333	0.8666666666666667
IN	148.0	687.0	835.0	835	0.17724550898203592	0.822754491017964
NV	67.0	366.0	433.0	433	0.15473441108545036	0.8452655889145496
MA	114.0	627.0	741.0	741	0.15384615384615385	0.8461538461538461
VA	87.0	530.0	617.0	617	0.14100486223662884	0.8589951377633711
IA	37.0	166.0	203.0	203	0.18226600985221675	0.8177339901477833
GA	184.0	998.0	1182.0	1182	0.155668358714044	0.8443316412859561
MN	61.0	330.0	391.0	391	0.15601023017902813	0.8439897698209718
NM	24.0	143.0	167.0	167	0.1437125748502994	0.8562874251497006
AK	14.0	86.0	100.0	100	0.14	0.86
DC	21.0	118.0	139.0	139	0.1510791366906475	0.8489208633093526
MS	64.0	323.0	387.0	387	0.165374677002584	0.834625322997416
ME	25.0	131.0	156.0	156	0.16025641025641027	0.8397435897435898
NE	12.0	66.0	78.0	78	0.15384615384615385	0.8461538461538461
SD	10.0	79.0	89.0	89	0.11235955056179775	0.8876404494382022
WI	95.0	445.0	540.0	540	0.17592592592592593	0.8240740740740741
KS	34.0	172.0	206.0	206	0.1650485436893204	0.8349514563106796
NH	8.0	92.0	100.0	100	0.08	0.92
HI	25.0	144.0	169.0	169	0.14792899408284024	0.8520710059171598
DE	8.0	96.0	104.0	104	0.07692307692307693	0.9230769230769231
VT	3.0	18.0	21.0	21	0.14285714285714285	0.8571428571428571
MT	13.0	63.0	76.0	76	0.17105263157894737	0.8289473684210527
RI	6.0	74.0	80.0	80	0.075	0.925
WY	9.0	25.0	34.0	34	0.2647058823529412	0.7352941176470589

In [48]:

```
school_state_respcode = Response_Code_Transform(vectorizer,tr_X['school_state'],tr_y)

school_state_respcode.shape
```

```
100%|██████████| 33498/33498 [00:00<00:00, 75052.00it/s]
```

Out[48]:

```
(33498, 2)
```

In [49]:

```
# On Test Data
ts_school_state_respcode = Response_Code_Transform(vectorizer,ts_X['school_state'],ts_y)
```

```
ts_school_state_respcode.shape
```

```
100%|| 16500/16500 [00:00<00:0
0, 74523.49it/s]
```

Out[49]:

```
(16500, 2)
```

Project Subject Category feature

In [50]:

```
# on train data
cat_list = tr_X['clean_categories'].unique()
vectorizer = Response_Code_Fit(cat_list,tr_X['clean_categories'],tr_y)
```

```
100%| 33498/33498 [00:21<00:
00, 1545.96it/s]
100%| 50/50
[00:00<?, ?it/s]
```

In [51]:

```
category_respcode = Response_Code_Transform(vectorizer,tr_X['clean_categories'],tr_y)

category_respcode.shape
```

```
100%| 33498/33498 [00:00<00:0
0, 73794.26it/s]
```

Out[51]:

```
(33498, 2)
```

In [52]:

```
# On test data
ts_category_respcode = Response_Code_Transform(vectorizer,ts_X['clean_categories'],ts_y)

ts_category_respcode.shape
```

```
100%| 16500/16500 [00:00<00:0
0, 71606.48it/s]
```

Out[52]:

```
(16500, 2)
```

Project Subject Subcategories feature

In [53]:

```
# on train data
cat_list = tr_X['clean_subcategories'].unique()
vectorizer = Response_Code_Fit(cat_list,tr_X['clean_subcategories'],tr_y)
```

```
100%| 33498/33498 [02:36<00
:00, 213.77it/s]
100%| 357/357 [00:00<00:00
, 357708.20it/s]
```

In [54]:

```
subcategory_respcode = Response_Code_Transform(vectorizer,tr_X['clean_subcategories'],tr_y)

subcategory_respcode.shape
```

```
100%|| 33498/33498 [00:00<00:0
0, 73489.37it/s]
```

Out[54]:

```
(33498, 2)
```

In [55]:

```
# on test data
ts_subcategory_respcode = Response_Code_Transform(vectorizer,ts_X['clean_subcategories'],ts_y)

ts_subcategory_respcode.shape
```

```
100%|| 16500/16500 [00:00<00:0
0, 73224.19it/s]
```

Out[55]:

```
(16500, 2)
```

Project grade category feature

In [56]:

```
# on train data
cat_list = tr_X['project_grade_category'].unique()
vectorizer = Response_Code_Fit(cat_list,tr_X['project_grade_category'],tr_y)
```

```
100%|| 33498/33498 [00:01<00:0
0, 17226.66it/s]
100%|| 4/4 [00:00<00:
00, 4087.02it/s]
```

In [57]:

```
project_grade_category_respcode = Response_Code_Transform(vectorizer,tr_X['project_grade_category'],tr_
y)

project_grade_category_respcode.shape
```

```
100%|| 33498/33498 [00:00<00:0
0, 74637.87it/s]
```

Out[57]:

```
(33498, 2)
```

In [58]:

```
# on test data
ts_project_grade_category_respcode = Response_Code_Transform(vectorizer,ts_X['project_grade_category'],
ts_y)

ts_project_grade_category_respcode.shape
```

```
100%|| 16500/16500 [00:00<00:0
```

```
0, 71600.33it/s]
```

Out[58]:

```
(16500, 2)
```

teacher prefix feature

In [59]:

```
# on train data
cat_list = tr_X['teacher_prefix'].unique()
vectorizer = Response_Code_Fit(cat_list,tr_X['teacher_prefix'],tr_y)
```

```
100%|██████████| 33498/33498 [00:02<00:0
0, 14388.57it/s]
100%|██████████| 5/5
[00:00<?, ?it/s]
```

In [60]:

```
teacher_prefix_respcode = Response_Code_Transform(vectorizer,tr_X['teacher_prefix'],tr_y)

teacher_prefix_respcode.shape
```

```
100%|██████████| 33498/33498 [00:00<00:0
0, 74350.85it/s]
```

Out[60]:

```
(33498, 2)
```

In [61]:

```
ts_teacher_prefix_respcode = Response_Code_Transform(vectorizer,ts_X['teacher_prefix'],ts_y)

ts_teacher_prefix_respcode.shape
```

```
100%|██████████| 16500/16500 [00:00<00:0
0, 73849.84it/s]
```

Out[61]:

```
(16500, 2)
```

2.3 Make Data Model Ready: encoding eassay, and project_title

In [62]:

```
# 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
```

BoW

BoW

In [63]:

```
### BoW in Essay and Title on Train

# # We are considering only the bigram words which appeared in at least 10 documents with max feature =
5000(rows or projects).
vectorizer_bow = CountVectorizer(min_df=10, max_features=5000)
tr_essay = vectorizer_bow.fit_transform(tr_X['clean_essay'].values)
print("Shape of essay matrix after one hot encodig on train",tr_essay.shape)

# # Similarly you can vectorize for title also
vectorizer_bowt = CountVectorizer(min_df=10, max_features=5000)
tr_title = vectorizer_bowt.fit_transform(tr_X['clean_project_title'].values)
print("Shape of title matrix after one hot encodig ",tr_title.shape)

### BoW in Essay and Title on Test
print('============================================================\n')
ts_essay = vectorizer_bow.transform(ts_X['clean_essay'].values)
print("Shape of essay matrix after one hot encodig on test",ts_essay.shape)

ts_title = vectorizer_bowt.transform(ts_X['clean_project_title'].values)
print("Shape of title matrix after one hot encodig on test",ts_title.shape)
```

```
Shape of essay matrix after one hot encodig on train (33498, 5000)
Shape of title matrix after one hot encodig  (33498, 1570)
============================================================

Shape of essay matrix after one hot encodig on test (16500, 5000)
Shape of title matrix after one hot encodig on test (16500, 1570)
```

TFIDF

In [90]:

```
### BoW in Essay and Title on Train

# # We are considering only the bigram words which appeared in at least 10 documents with max feature =
5000(rows or projects).
vectorizer_bow = TfidfVectorizer(min_df=10, max_features=5000)
tr_essay = vectorizer_bow.fit_transform(tr_X['clean_essay'].values)
print("Shape of essay matrix after one hot encodig on train",tr_essay.shape)

# # Similarly you can vectorize for title also
vectorizer_bowt = TfidfVectorizer(min_df=10, max_features=5000)
tr_title = vectorizer_bowt.fit_transform(tr_X['clean_project_title'].values)
print("Shape of title matrix after one hot encodig ",tr_title.shape)

### BoW in Essay and Title on Test
print('============================================================\n')
ts_essay = vectorizer_bow.transform(ts_X['clean_essay'].values)
print("Shape of essay matrix after one hot encodig on test",ts_essay.shape)

ts_title = vectorizer_bowt.transform(ts_X['clean_project_title'].values)
print("Shape of title matrix after one hot encodig on test",ts_title.shape)
```

```
Shape of essay matrix after one hot encodig on train (33498, 5000)
Shape of title matrix after one hot encodig  (33498, 1570)
============================================================

Shape of essay matrix after one hot encodig on test (16500, 5000)
Shape of title matrix after one hot encodig on test (16500, 1570)
```

AVGW2V

In [113]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-an
```

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

In [114]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_essay = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(tr_X['clean_essay'].values): # 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_essay.append(vector)

avg_w2v_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(tr_X['clean_project_title'].values): # 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_title.append(vector)

tr_essay = np.array(avg_w2v_essay)
tr_title = np.array(avg_w2v_title)
print('======================== Train Essay ==========================')
print(len(avg_w2v_essay))
print(len(avg_w2v_essay[0]))
print('======================== Train Title ==============================')
print(len(avg_w2v_title))
print(len(avg_w2v_title[0]))
# print(avg_w2v_essay[0])
```

```
100%|██████████| 33498/33498 [00:07<00:00, 4528.48it/s]
100%|██████████| 33498/33498 [00:00<00:00, 94711.87it/s]
```

```
======================== Train Essay ==========================
33498
300
======================== Train Title ==============================
33498
300
```

In [115]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_essay = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(ts_X['clean_essay'].values): # 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_essay.append(vector)
```

```
avg_w2v_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(ts_X['clean_project_title'].values): # 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_title.append(vector)

ts_essay = np.array(avg_w2v_essay)
ts_title = np.array(avg_w2v_title)
print('======================== Test Essay ============================')
print(len(avg_w2v_essay))
print(len(avg_w2v_essay[0]))
print('======================== Test Title ============================')
print(len(avg_w2v_title))
print(len(avg_w2v_title[0]))
# print(avg_w2v_essay[0])
```

```
100%|██████████| 16500/16500 [00:03<00:00, 4956.38it/s]
100%|██████████| 16500/16500 [00:00<00:00, 104541.76it/s]
```

```
======================== Test Essay ============================
16500
300
======================== Test Title ============================
16500
300
```

TFIDFW2V

In [135]:

```
# Tfidf weighted w2v on essay in train
tfidf_model = TfidfVectorizer()
tfidf_model.fit(tr_X['clean_essay'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())

# tfidf Word2Vec
# compute average word2vec for each essay
tfidf_w2v_essay = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(tr_X['clean_essay'].values): # 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_essay.append(vector)

tr_essay = np.array(tfidf_w2v_essay)

# compute average word2vec for each essay for test data
tfidf_w2v_essay = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(ts_X['clean_essay'].values): # 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_essay.append(vector)

ts_essay = np.array(tfidf_w2v_essay)
```

```
100%|██████████| 33498/33498 [00:50<00
:00, 667.07it/s]
100%|██████████| 16500/16500 [00:23<00
:00, 693.71it/s]
```

In [136]:

```
# tfidf Word2Vec on title
# compute average word2vec for each title for train data
tfidf_model = TfidfVectorizer()
tfidf_model.fit(tr_X['clean_project_title'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())

tfidf_w2v_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(tr_X['clean_project_title'].values): # 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_title.append(vector)

tr_title = np.array(tfidf_w2v_title)

# compute average word2vec for each title for test data
tfidf_w2v_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(ts_X['clean_project_title'].values): # 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_title.append(vector)

ts_title = np.array(tfidf_w2v_title)
```

```
100%|██████████| 33498/33498 [00:00<00:0
0, 47107.26it/s]
100%|██████████| 16500/16500 [00:00<00:0
0, 48516.87it/s]
```

Merge Them

In [137]:

```
tr_X['quantity'].values.reshape(-1,1).shape, tr_X['price'].values.reshape(-1,1).shape, \
tr_X['teacher_number_of_previously_posted_projects'].values.reshape(-1,1).shape
```

Out[137]:

```
((33498, 1), (33498, 1), (33498, 1))
```

In [138]:

```
school_state_respcode.shape, category_respcode.shape, subcategory_respcode.shape, project_grade_categor
y_respcode.shape, \
teacher_prefix_respcode.shape
```

Out[138]:

```
((33498, 2), (33498, 2), (33498, 2), (33498, 2), (33498, 2))
```

In [139]:

```
tr_essay.shape, tr_title.shape
```

Out[139]:

```
((33498, 300), (33498, 300))
```

In [140]:

```
# for train data
from scipy.sparse import hstack, coo_matrix
tr_X = hstack((tr_X['quantity'].values.reshape(-1,1), tr_X['price'].values.reshape(-1,1), \
               tr_X['teacher_number_of_previously_posted_projects'].values.reshape(-1,1), \
               school_state_respcode, category_respcode, subcategory_respcode, project_grade_category_r
espcode, \
               teacher_prefix_respcode, coo_matrix(tr_essay), coo_matrix(tr_title)))
tr_X.shape
```

Out[140]:

```
(33498, 613)
```

In [141]:

```
tr_X = tr_X.toarray()
```

In [142]:

```
tr_X
```

Out[142]:

```
array([[ 8.00000000e+00,  1.02550000e+02,  1.00000000e+00, ...,
         5.62740370e-01,  3.32185257e-01,  6.62938513e-03],
       [ 5.00000000e+01,  6.63160000e+02,  1.00000000e+00, ...,
        -1.07392031e-01,  1.13037392e-01, -5.32741321e-02],
       [ 4.10000000e+01,  7.71800000e+02,  2.00000000e+00, ...,
        -3.76540000e-01,  1.09930000e-01,  1.09630000e-01],
       ...,
       [ 9.00000000e+00,  1.49940000e+02,  2.00000000e+00, ...,
        -7.13769434e-02,  2.44995848e-01, -3.13615045e-02],
       [ 4.80000000e+01,  1.26090000e+02,  0.00000000e+00, ...,
```

```
         1.93080076e-01, -1.35910788e-02,  1.35584737e-01],
       [ 3.00000000e+01,  1.07200000e+01,  7.00000000e+00, ...,
         1.52446119e-01,  1.45104319e-01, -2.25854226e-02]])
```

In [143]:

```
# for train data
from scipy.sparse import hstack
ts_X = hstack((ts_X['quantity'].values.reshape(-1,1), ts_X['price'].values.reshape(-1,1), \
               ts_X['teacher_number_of_previously_posted_projects'].values.reshape(-1,1), \
              ts_school_state_respcode, ts_category_respcode, ts_subcategory_respcode, ts_project_grade
_category_respcode, \
              ts_teacher_prefix_respcode, coo_matrix(ts_essay), coo_matrix(ts_title)))
ts_X.shape
```

Out[143]:

```
(16500, 613)
```

In [144]:

```
ts_X = ts_X.toarray()
```

In [145]:

```
ts_X
```

Out[145]:

```
array([[ 1.00000000e+01,  1.01900000e+01,  3.50000000e+01, ...,
        -8.05803013e-02,  1.48266958e-01,  3.24619233e-01],
       [ 2.90000000e+01,  7.56700000e+01,  6.00000000e+00, ...,
         1.94009481e-01,  1.67108189e-01, -3.34668280e-02],
       [ 3.70000000e+01,  4.04290000e+02,  7.00000000e+00, ...,
         7.25610000e-01, -1.83320000e-01, -2.86450000e-01],
       ...,
       [ 3.00000000e+00,  6.47900000e+01,  0.00000000e+00, ...,
         1.05863359e-01,  5.08487832e-01, -6.00669971e-02],
       [ 1.20000000e+01,  1.86880000e+02,  2.00000000e+00, ...,
         6.87395826e-02,  1.52817303e-01, -5.68150349e-02],
       [ 2.10000000e+01,  6.38800000e+01,  3.00000000e+00, ...,
         8.75654292e-02,  6.95654009e-02, -1.04822263e-02]])
```

In [146]:

```
# check whether data still contain NaN or infinity or not
np.any(np.isnan(tr_X)), np.any(np.isnan(ts_X))
```

Out[146]:

```
(False, False)
```

In [147]:

```
np.all(np.isfinite(tr_X)), np.all(np.isfinite(ts_X))
```

Out[147]:

```
(True, True)
```

In [148]:

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

```
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

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

In [149]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
clf = RandomForestClassifier(random_state=1, class_weight='balanced')
```

In [150]:

```
parameter = {'n_estimators':[1, 2, 4, 8, 16, 32, 64, 100], \
             'max_depth':np.linspace(1, 32, 32, endpoint=True)}
```

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 [0]:

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

In [78]:

```
gridclf = GridSearchCV(clf,parameter,verbose=3,scoring='roc_auc',cv=3, n_jobs=3)
gridclf.fit(tr_X,tr_y)
```

```
Fitting 3 folds for each of 256 candidates, totalling 768 fits

[Parallel(n_jobs=3)]: Using backend LokyBackend with 3 concurrent workers.
[Parallel(n_jobs=3)]: Done  26 tasks      | elapsed:  1.2min
[Parallel(n_jobs=3)]: Done 122 tasks      | elapsed:  5.7min
[Parallel(n_jobs=3)]: Done 282 tasks      | elapsed: 15.4min
[Parallel(n_jobs=3)]: Done 506 tasks      | elapsed: 35.3min
[Parallel(n_jobs=3)]: Done 768 out of 768 | elapsed: 63.7min finished
```

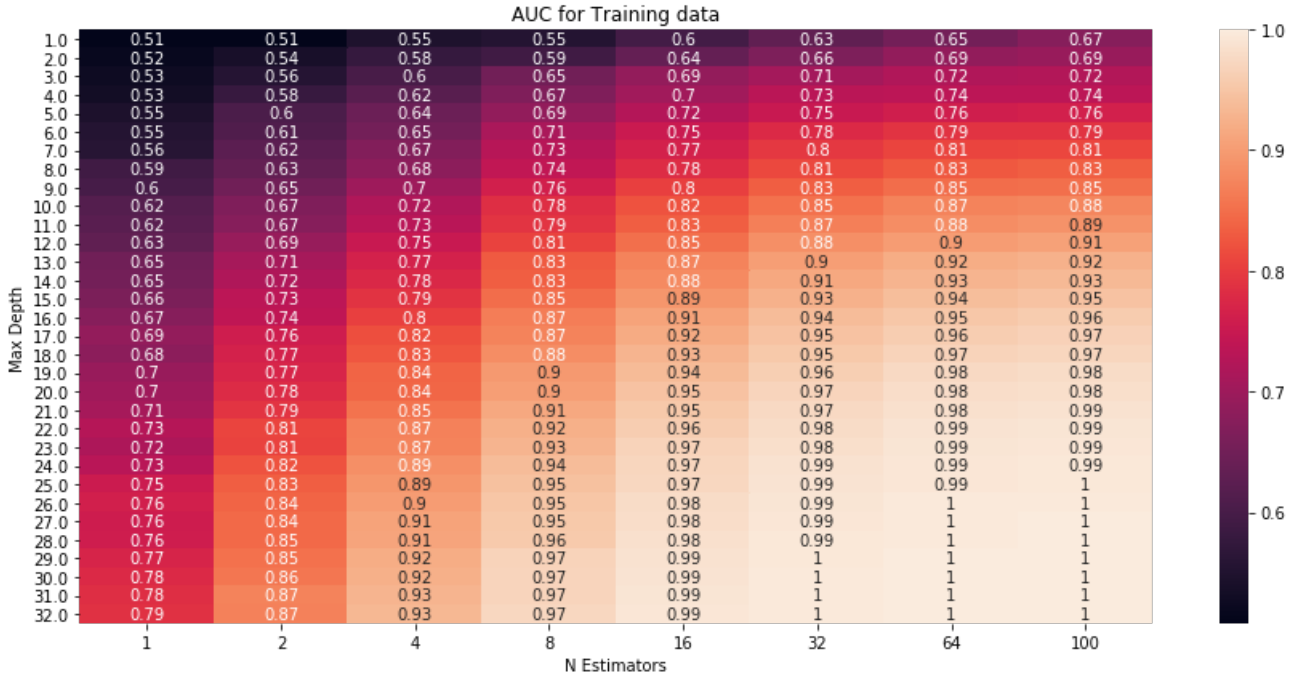
Out[78]:

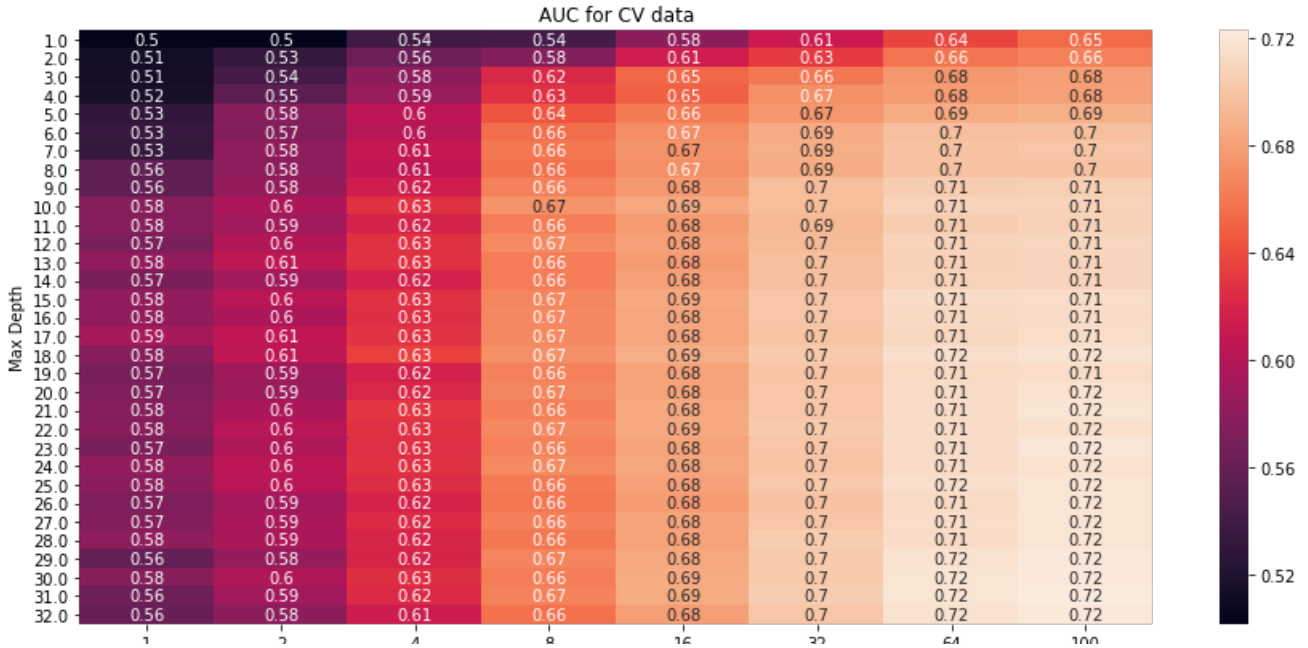
```
GridSearchCV(cv=3, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators='warn', n_jobs=None, oob_score=False,
            random_state=1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=3,
       param_grid={'n_estimators': [1, 2, 4, 8, 16, 32, 64, 100], 'max_depth': array([ 1.,  2.,  3.,  4
.,  5.,  6.,  7.,  8.,  9., 10., 11., 12., 13.,
       14., 15., 16., 17., 18., 19., 20., 21., 22., 23., 24., 25., 26.,
       27., 28., 29., 30., 31., 32.])},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
```

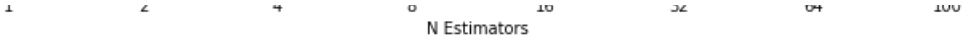
```
        scoring='roc_auc', verbose=3)
```

In [79]:

```
# Plot seaborn heatmap for gridsearchcv: https://www.kaggle.com/arindambanerjee/grid-search-simplified
max_depth_list = list(gridclf.cv_results_['param_max_depth'].data)
samplesplit_list = list(gridclf.cv_results_['param_n_estimators'].data)

plt.figure(1, figsize=(15,15))
plt.subplot(211)
data = pd.DataFrame(data={'Max Depth':max_depth_list, \
                          'N Estimators':samplesplit_list , \
                          'AUC':gridclf.cv_results_['mean_train_score']})
data = data.pivot(index='Max Depth', columns='N Estimators', values='AUC')
sns.heatmap(data, annot=True).set_title('AUC for Training data')

plt.subplot(212)
data = pd.DataFrame(data={'Max Depth':max_depth_list, \
                          'N Estimators':samplesplit_list , \
                          'AUC':gridclf.cv_results_['mean_test_score']})
data = data.pivot(index='Max Depth', columns='N Estimators', values='AUC')
sns.heatmap(data, annot=True).set_title('AUC for CV data')

plt.show()
```

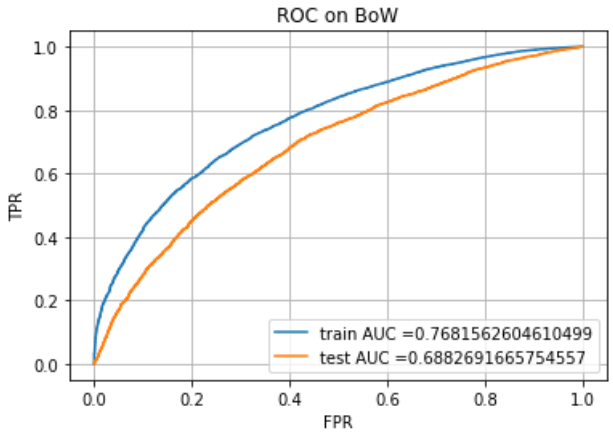






In [86]:

```
lr = RandomForestClassifier(max_depth=6, class_weight='balanced', n_estimators=100, random_state=1)
lr.fit(tr_X, tr_y)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive clas
s
# not the predicted outputs

y_train_pred = lr.predict_proba(tr_X)[:,1]
y_test_pred = lr.predict_proba(ts_X)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(tr_y, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(ts_y, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC on BoW")
plt.grid()
plt.show()
```



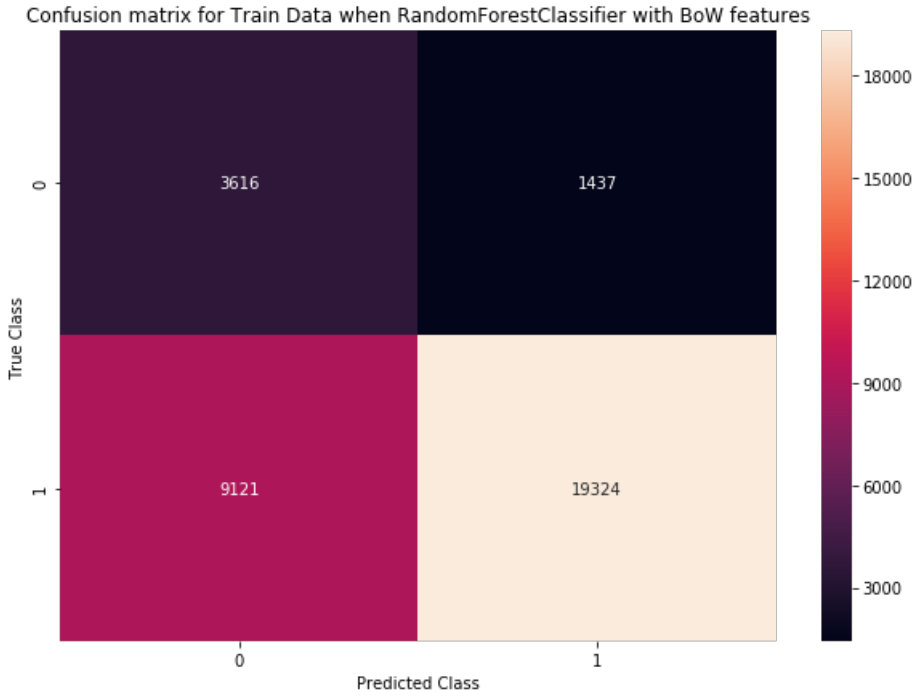
In [87]:

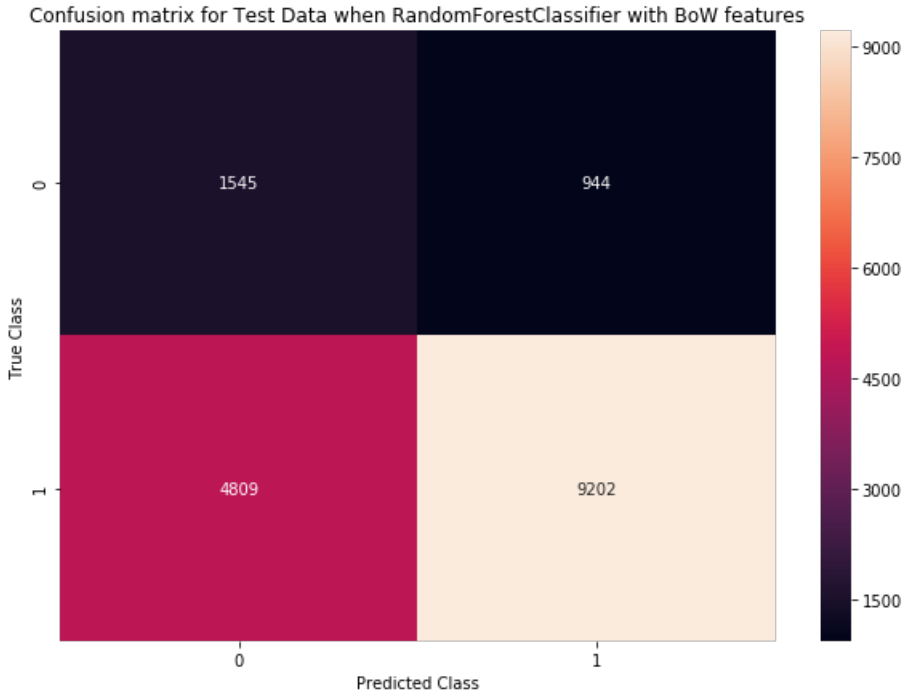
```
feature_names = 'BoW'
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
cm = metrics.confusion_matrix(tr_y, predict_with_best_t(y_train_pred, best_t))
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt="d")
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.title('Confusion matrix for Train Data when RandomForestClassifier with {0} features'.format(featur
e_names))

print("Test confusion matrix")
cm = metrics.confusion_matrix(ts_y, predict_with_best_t(y_test_pred, best_t))
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt="d")
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.title('Confusion matrix for Test Data when RandomForestClassifier with {0} features'.format(feature
_names))
```

```
the maximum value of tpr*(1-fpr) 0.48614991513580585 for threshold 0.503
Train confusion matrix
Test confusion matrix
```

Out[87]:

```
Text(0.5, 1.0, 'Confusion matrix for Test Data when RandomForestClassifier with BoW features')
```





2.4.2 Applying Random Forests on TFIDF, SET 2

In [0]:

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

In [105]:

```
gridclf = GridSearchCV(clf,parameter,verbose=3,scoring='roc_auc',cv=3, n_jobs=3)
gridclf.fit(tr_X,tr_y)
```

```
Fitting 3 folds for each of 256 candidates, totalling 768 fits
```

```
[Parallel(n_jobs=3)]: Using backend LokyBackend with 3 concurrent workers.
[Parallel(n_jobs=3)]: Done  26 tasks      | elapsed:  1.2min
```

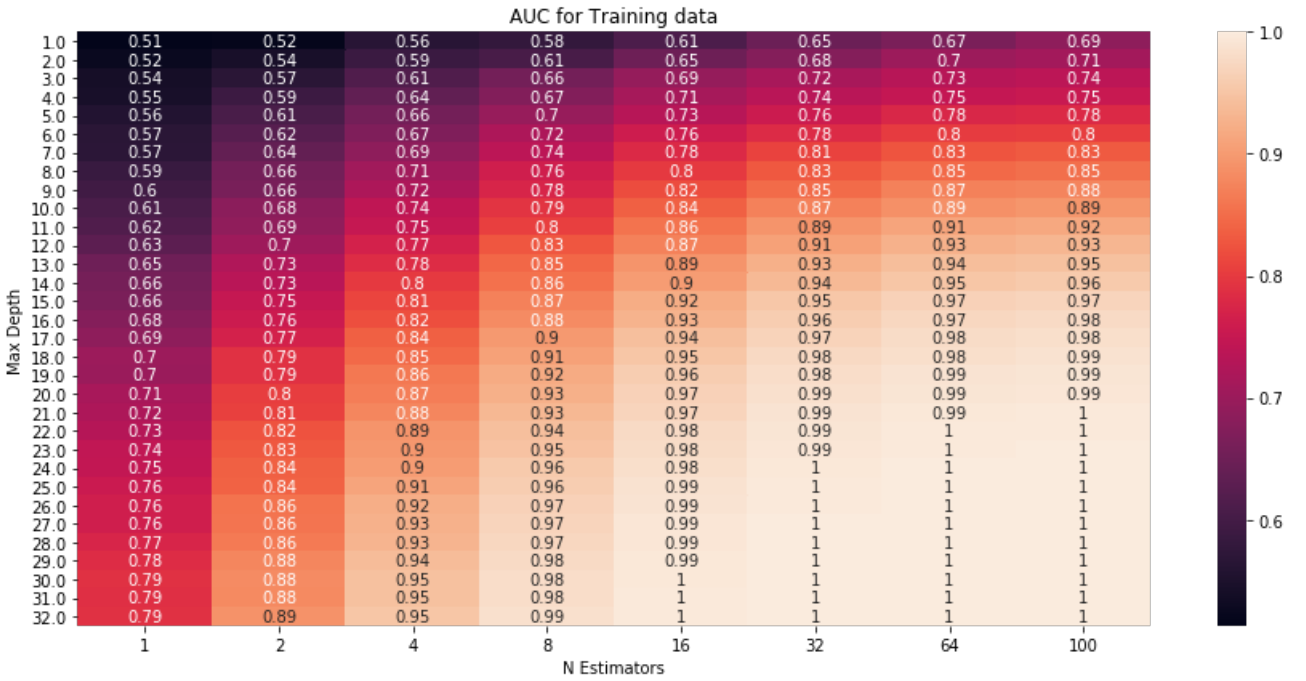
```
[Parallel(n_jobs=3)]: Done 122 tasks       | elapsed:  5.8min
[Parallel(n_jobs=3)]: Done 282 tasks       | elapsed: 15.7min
[Parallel(n_jobs=3)]: Done 506 tasks       | elapsed: 35.6min
[Parallel(n_jobs=3)]: Done 768 out of 768 | elapsed: 65.1min finished
```

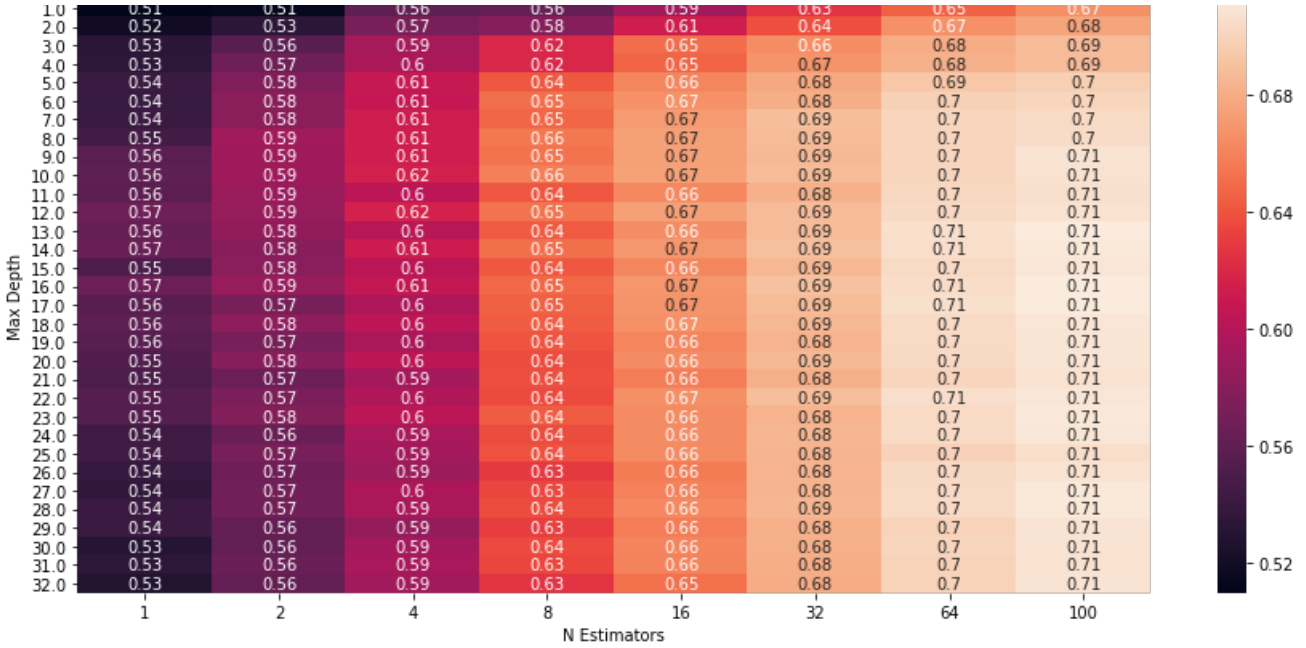
Out[105]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators='warn', n_jobs=None, oob_score=False,
            random_state=1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=3,
       param_grid={'n_estimators': [1, 2, 4, 8, 16, 32, 64, 100], 'max_depth': array([ 1.,  2.,  3.,  4
.,  5.,  6.,  7.,  8.,  9., 10., 11., 12., 13.,
       14., 15., 16., 17., 18., 19., 20., 21., 22., 23., 24., 25., 26.,
       27., 28., 29., 30., 31., 32.])},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=3)
```

In [106]:

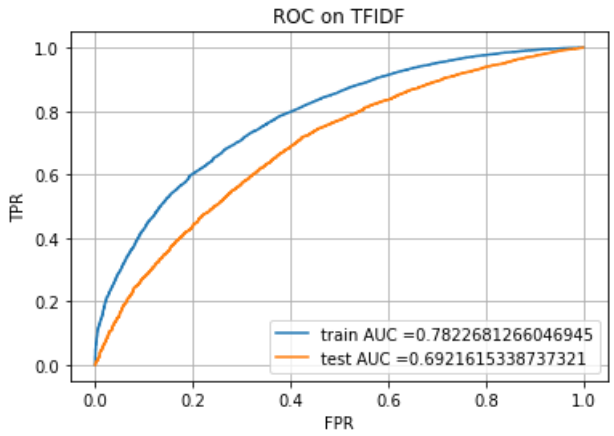
```
# Plot seaborn heatmap for gridsearchcv: https://www.kaggle.com/arindambanerjee/grid-search-simplified
max_depth_list = list(gridclf.cv_results_['param_max_depth'].data)
samplesplit_list = list(gridclf.cv_results_['param_n_estimators'].data)

plt.figure(1, figsize=(15,15))
plt.subplot(211)
data = pd.DataFrame(data={'Max Depth':max_depth_list, \
                          'N Estimators':samplesplit_list , \
                          'AUC':gridclf.cv_results_['mean_train_score']})
data = data.pivot(index='Max Depth', columns='N Estimators', values='AUC')
sns.heatmap(data, annot=True).set_title('AUC for Training data')

plt.subplot(212)
data = pd.DataFrame(data={'Max Depth':max_depth_list, \
                          'N Estimators':samplesplit_list , \
                          'AUC':gridclf.cv_results_['mean_test_score']})
data = data.pivot(index='Max Depth', columns='N Estimators', values='AUC')
sns.heatmap(data, annot=True).set_title('AUC for CV data')

plt.show()
```





In [109]:

```
lr = RandomForestClassifier(max_depth=6, class_weight='balanced', n_estimators=100, random_state=1)
lr.fit(tr_X, tr_y)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive clas
s
# not the predicted outputs

y_train_pred = lr.predict_proba(tr_X)[:,1]
y_test_pred = lr.predict_proba(ts_X)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(tr_y, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(ts_y, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC on TFIDF")
plt.grid()
plt.show()
```



In [110]:

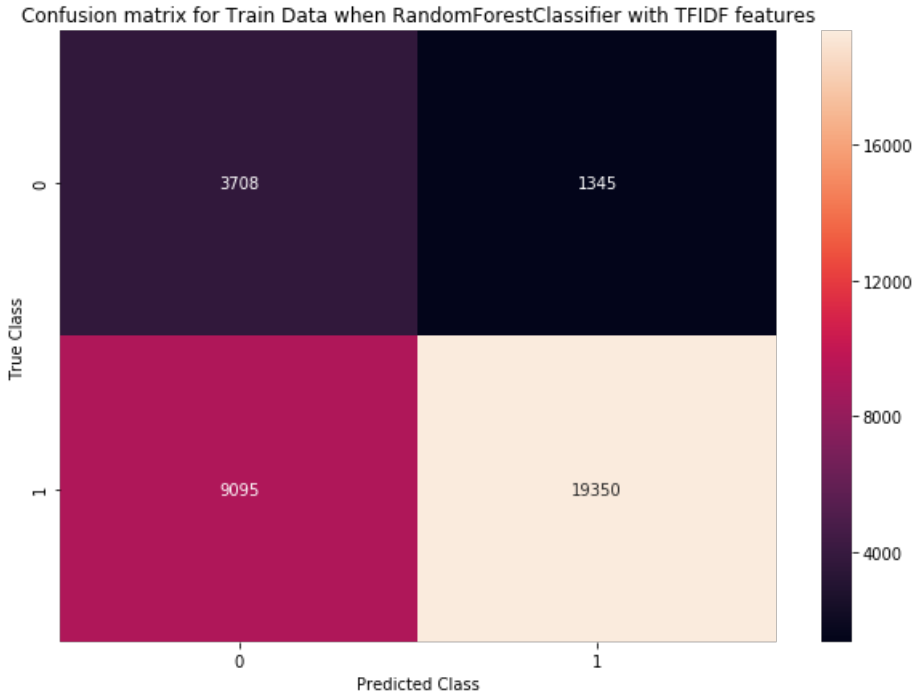
```
feature_names = 'TFIDF'
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
cm = metrics.confusion_matrix(tr_y, predict_with_best_t(y_train_pred, best_t))
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt="d")
```

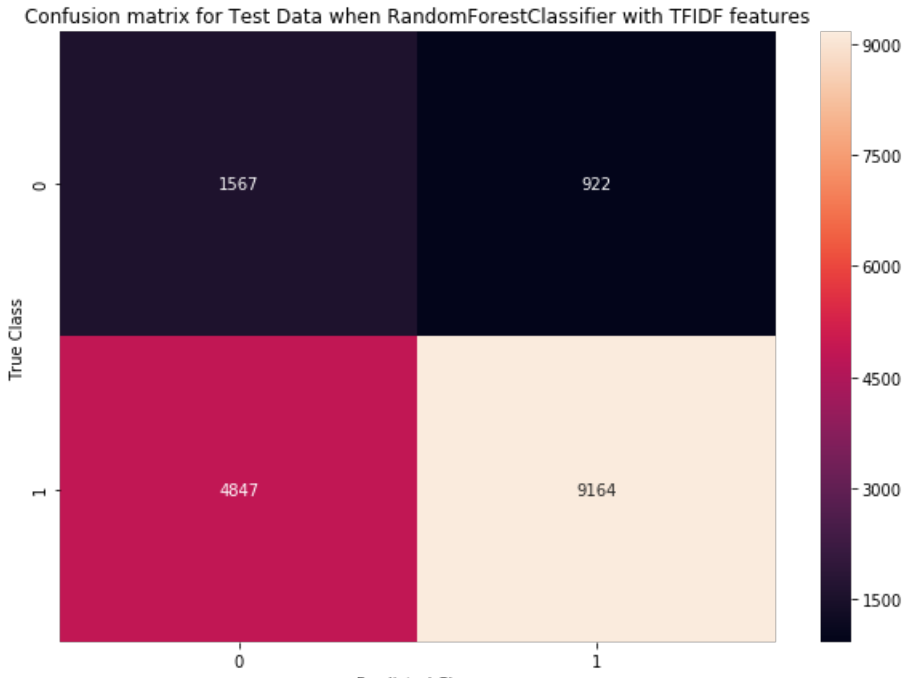
```
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.title('Confusion matrix for Train Data when RandomForestClassifier with {0} features'.format(featur
e_names))

print("Test confusion matrix")
cm = metrics.confusion_matrix(ts_y, predict_with_best_t(y_test_pred, best_t))
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt="d")
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.title('Confusion matrix for Test Data when RandomForestClassifier with {0} features'.format(feature
_names))
```

```
the maximum value of tpr*(1-fpr) 0.4991895192033177 for threshold 0.506
Train confusion matrix
Test confusion matrix
```

Out[110]:

```
Text(0.5, 1.0, 'Confusion matrix for Test Data when RandomForestClassifier with TFIDF features')
```





Predicted Class

2.4.3 Applying Random Forests on AVG W2V, SET 3

In [0]:

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

In [129]:

```
gridclf = GridSearchCV(clf,parameter,verbose=3,scoring='roc_auc',cv=3, n_jobs=5)
gridclf.fit(tr_X,tr_y)
```

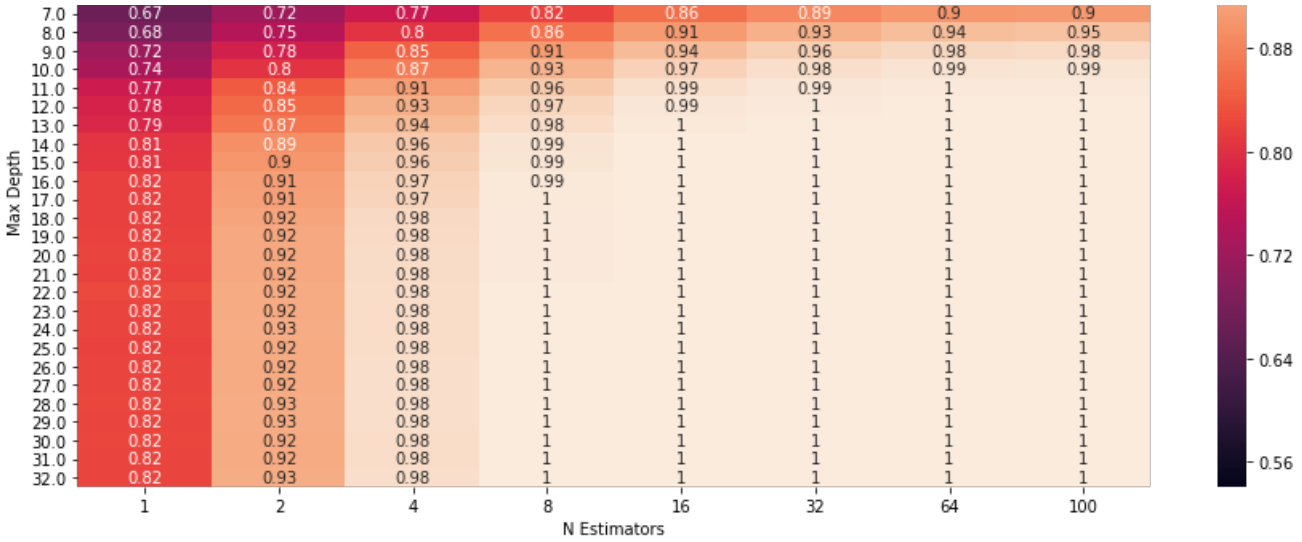
```
Fitting 3 folds for each of 256 candidates, totalling 768 fits

[Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent workers.
[Parallel(n_jobs=5)]: Done  22 tasks      | elapsed:    9.7s
[Parallel(n_jobs=5)]: Done 118 tasks      | elapsed:  1.4min
[Parallel(n_jobs=5)]: Done 278 tasks      | elapsed:  5.5min
[Parallel(n_jobs=5)]: Done 502 tasks      | elapsed: 13.8min
[Parallel(n_jobs=5)]: Done 768 out of 768 | elapsed: 24.5min finished
```

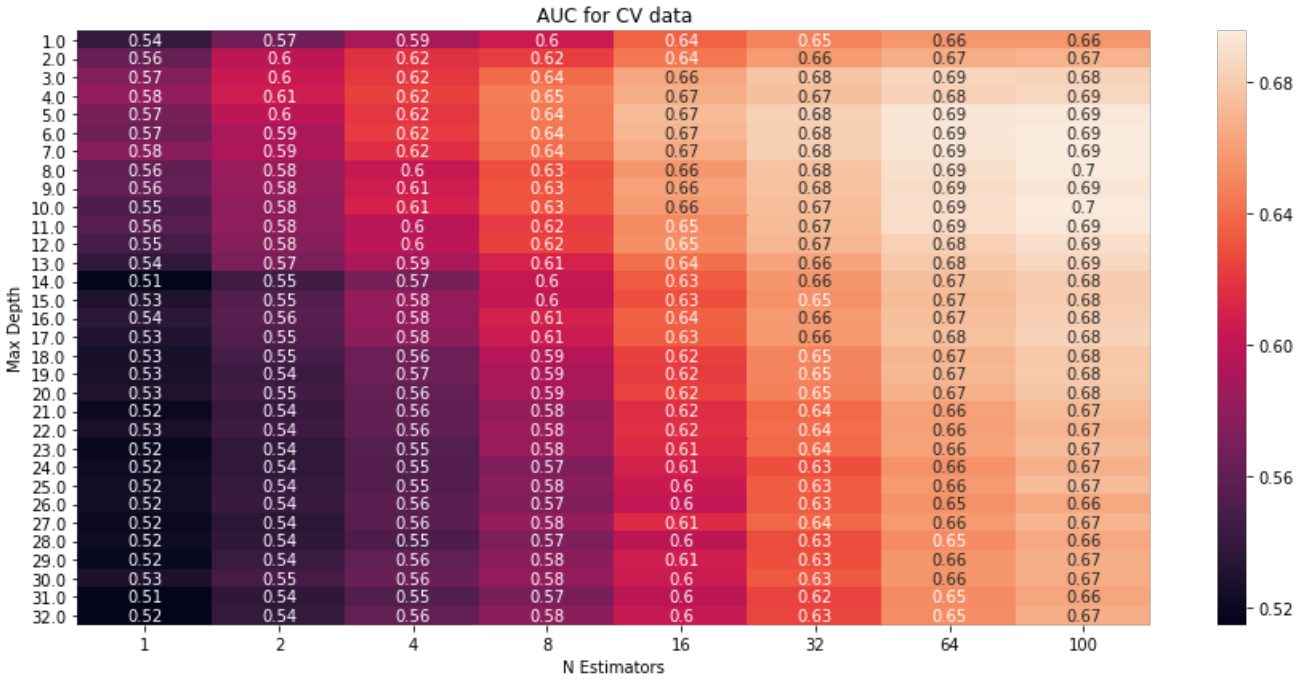
Out[129]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators='warn', n_jobs=None, oob_score=False,
            random_state=1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=5,
       param_grid={'n_estimators': [1, 2, 4, 8, 16, 32, 64, 100], 'max_depth': array([ 1.,  2.,  3.,  4
.,  5.,  6.,  7.,  8.,  9., 10., 11., 12., 13.,
       14., 15., 16., 17., 18., 19., 20., 21., 22., 23., 24., 25., 26.,
       27., 28., 29., 30., 31., 32.])},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=3)
```

In [130]:

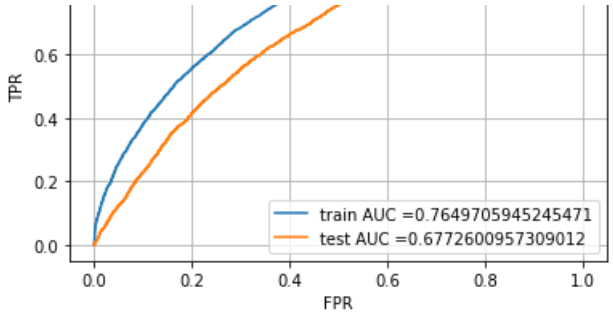
```
# Plot seaborn heatmap for gridsearchcv: https://www.kaggle.com/arindambanerjee/grid-search-simplified
max_depth_list = list(gridclf.cv_results_['param_max_depth'].data)
samplesplit_list = list(gridclf.cv_results_['param_n_estimators'].data)

plt.figure(1, figsize=(15,15))
plt.subplot(211)
data = pd.DataFrame(data={'Max Depth':max_depth_list, \
                          'N Estimators':samplesplit_list , \
                          'AUC':gridclf.cv_results_['mean_train_score']})
data = data.pivot(index='Max Depth', columns='N Estimators', values='AUC')
sns.heatmap(data, annot=True).set_title('AUC for Training data')

plt.subplot(212)
data = pd.DataFrame(data={'Max Depth':max_depth_list, \
                          'N Estimators':samplesplit_list , \
                          'AUC':gridclf.cv_results_['mean_test_score']})
data = data.pivot(index='Max Depth', columns='N Estimators', values='AUC')
sns.heatmap(data, annot=True).set_title('AUC for CV data')

plt.show()
```





In [131]:

```
lr = RandomForestClassifier(max_depth=5, class_weight='balanced', n_estimators=64, random_state=1)
lr.fit(tr_X, tr_y)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive clas
s
# not the predicted outputs

y_train_pred = lr.predict_proba(tr_X)[:,1]
y_test_pred = lr.predict_proba(ts_X)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(tr_y, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(ts_y, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC on AVGW2V")
plt.grid()
plt.show()
```



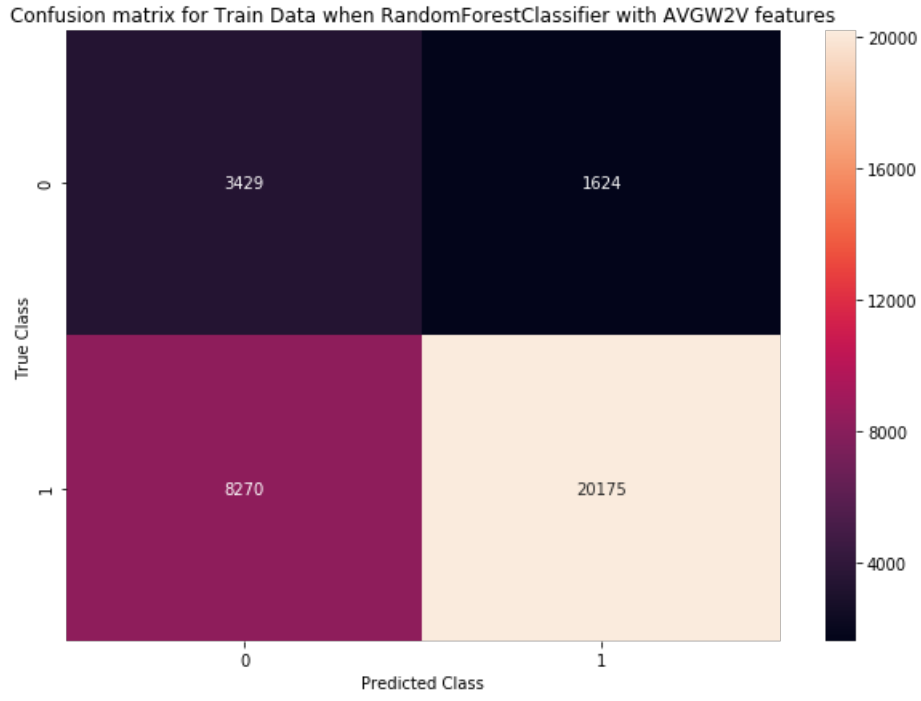
In [132]:

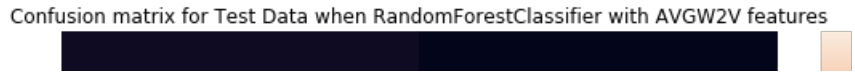
```
feature_names = 'AVGW2V'
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
cm = metrics.confusion_matrix(tr_y, predict_with_best_t(y_train_pred, best_t))
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt="d")
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.title('Confusion matrix for Train Data when RandomForestClassifier with {0} features'.format(featur
e_names))

print("Test confusion matrix")
cm = metrics.confusion_matrix(ts_y, predict_with_best_t(y_test_pred, best_t))
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt="d")
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.title('Confusion matrix for Test Data when RandomForestClassifier with {0} features'.format(feature
_names))
```

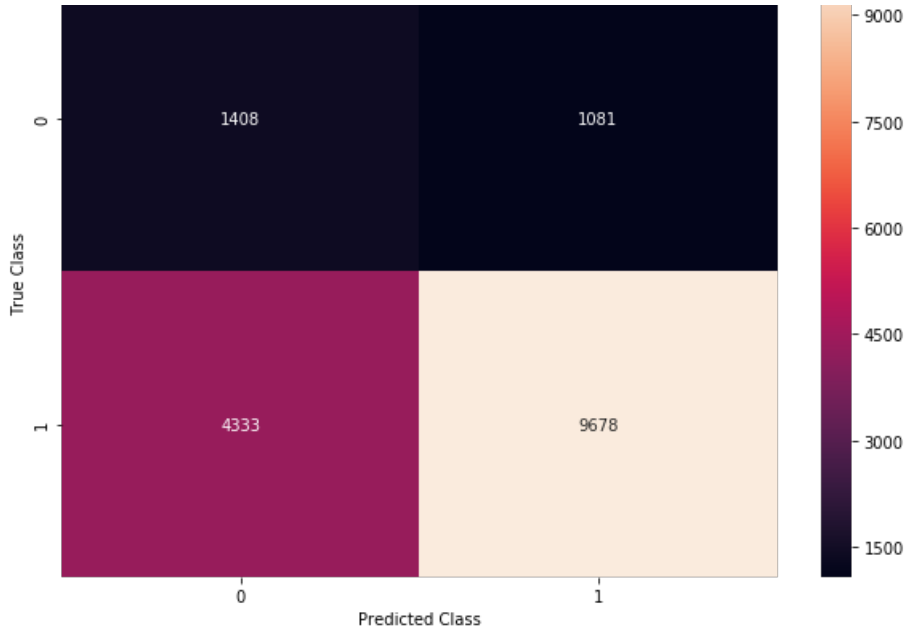
```
the maximum value of tpr*(1-fpr) 0.48131100543415406 for threshold 0.506
Train confusion matrix
Test confusion matrix
```

Out[132]:

```
Text(0.5, 1.0, 'Confusion matrix for Test Data when RandomForestClassifier with AVGW2V features')
```







2.4.4 Applying Random Forests on TFIDF W2V, SET 4

In [0]:

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

In [151]:

```
gridclf = GridSearchCV(clf,parameter,verbose=3,scoring='roc_auc',cv=3, n_jobs=5)
gridclf.fit(tr_X,tr_y)
```

```
Fitting 3 folds for each of 256 candidates, totalling 768 fits
```

```
[Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent workers.
[Parallel(n_jobs=5)]: Done  22 tasks      | elapsed:    9.0s
[Parallel(n_jobs=5)]: Done 118 tasks      | elapsed:  1.4min
[Parallel(n_jobs=5)]: Done 278 tasks      | elapsed:  5.4min
[Parallel(n_jobs=5)]: Done 502 tasks      | elapsed: 13.7min
[Parallel(n_jobs=5)]: Done 768 out of 768 | elapsed: 24.2min finished
```

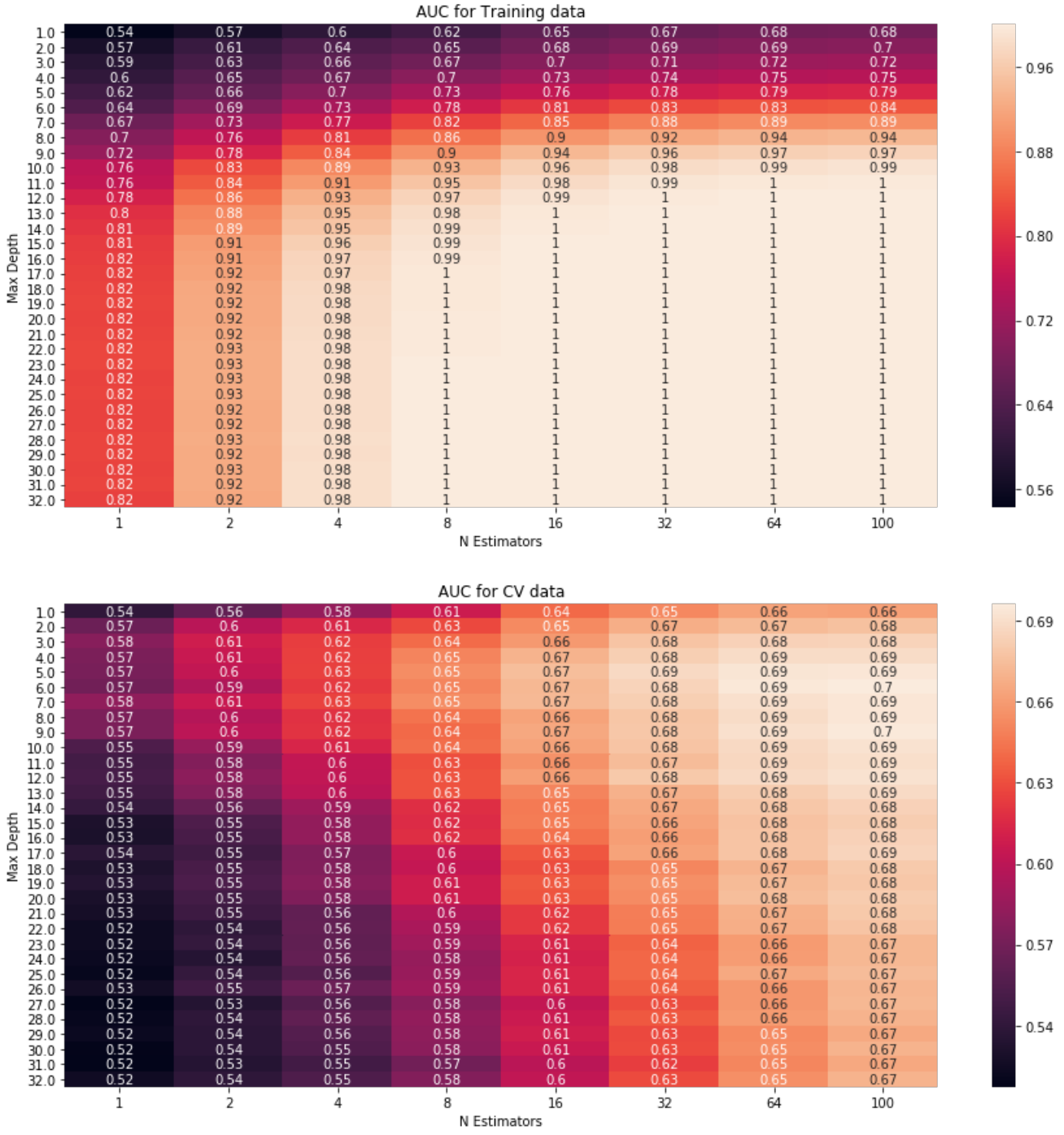
Out[151]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators='warn', n_jobs=None, oob_score=False,
            random_state=1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=5,
       param_grid={'n_estimators': [1, 2, 4, 8, 16, 32, 64, 100], 'max_depth': array([ 1.,  2.,  3.,  4
.,  5.,  6.,  7.,  8.,  9., 10., 11., 12., 13.,
       14., 15., 16., 17., 18., 19., 20., 21., 22., 23., 24., 25., 26.,
       27., 28., 29., 30., 31., 32.])},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=3)
```

In [152]:

```
# Plot seaborn heatmap for gridsearchcv: https://www.kaggle.com/arindambanerjee/grid-search-simplified
max_depth_list = list(gridclf.cv_results_['param_max_depth'].data)
samplesplit_list = list(gridclf.cv_results_['param_n_estimators'].data)

plt.figure(1, figsize=(15,15))
```

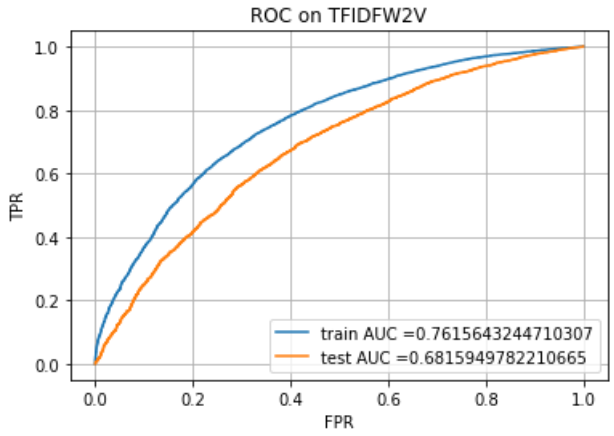
```
plt.subplot(211)
data = pd.DataFrame(data={'Max Depth':max_depth_list, \
                          'N Estimators':samplesplit_list , \
                          'AUC':gridclf.cv_results_['mean_train_score']})
data = data.pivot(index='Max Depth', columns='N Estimators', values='AUC')
sns.heatmap(data, annot=True).set_title('AUC for Training data')

plt.subplot(212)
data = pd.DataFrame(data={'Max Depth':max_depth_list, \
                          'N Estimators':samplesplit_list , \
                          'AUC':gridclf.cv_results_['mean_test_score']})
data = data.pivot(index='Max Depth', columns='N Estimators', values='AUC')
sns.heatmap(data, annot=True).set_title('AUC for CV data')

plt.show()
```



In [153]:

```
lr = RandomForestClassifier(max_depth=5, class_weight='balanced', n_estimators=100, random_state=1)
lr.fit(tr_X, tr_y)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive clas
s
```

```
# not the predicted outputs

y_train_pred = lr.predict_proba(tr_X)[:,1]
y_test_pred = lr.predict_proba(ts_X)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(tr_y, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(ts_y, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC on TFIDFW2V")
plt.grid()
plt.show()
```



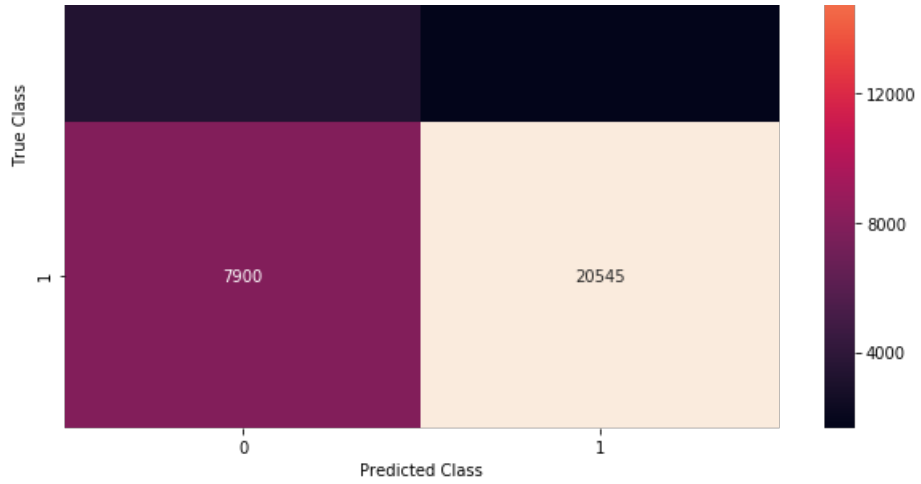
In [154]:

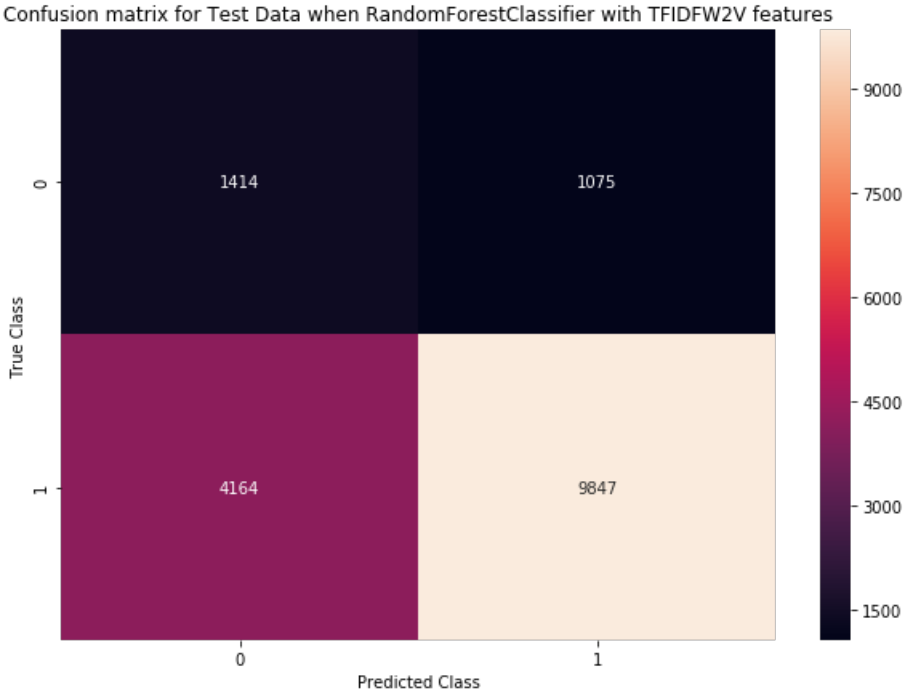
```
feature_names = 'TFIDFW2V'
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
cm = metrics.confusion_matrix(tr_y, predict_with_best_t(y_train_pred, best_t))
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt="d")
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.title('Confusion matrix for Train Data when RandomForestClassifier with {0} features'.format(featur
e_names))

print("Test confusion matrix")
cm = metrics.confusion_matrix(ts_y, predict_with_best_t(y_test_pred, best_t))
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt="d")
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.title('Confusion matrix for Test Data when RandomForestClassifier with {0} features'.format(feature
_names))
```

```
the maximum value of tpr*(1-fpr) 0.4859927900134824 for threshold 0.497
Train confusion matrix
Test confusion matrix
```

Out[154]:

```
Text(0.5, 1.0, 'Confusion matrix for Test Data when RandomForestClassifier with TFIDFW2V features')
```





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 [0]:

```
import xgboost as xgb
from sklearn.model_selection import GridSearchCV
clf = xgb.XGBClassifier(random_state=1)
parameter = {'n_estimators':[2, 3], \
             'max_depth':[3]}
```

2.5.1 Applying XGBOOST on BOW, SET 1

In [0]:

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

In [0]:

```
gridclf = GridSearchCV(clf,parameter,cv=3,verbose=3,scoring='roc_auc')
```

```
gridclf.fit(tr_X,tr_y)
```

```
Fitting 3 folds for each of 2 candidates, totalling 6 fits
```

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

```
[CV] max_depth=3, n_estimators=2 .....................................
[CV]  max_depth=3, n_estimators=2, score=0.6364190376103387, total=  18.7s
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   22.6s remaining:    0.0s
```

```
[CV] max_depth=3, n_estimators=2 .....................................
[CV]  max_depth=3, n_estimators=2, score=0.6633725479878396, total=  16.8s
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:   43.9s remaining:    0.0s
```

```
[CV] max_depth=3, n_estimators=2 .....................................
[CV]  max_depth=3, n_estimators=2, score=0.660543468460684, total=  18.0s
[CV] max_depth=3, n_estimators=3 .....................................
[CV]  max_depth=3, n_estimators=3, score=0.6384374063982794, total=  25.8s
[CV] max_depth=3, n_estimators=3 .....................................
[CV]  max_depth=3, n_estimators=3, score=0.6680518040749492, total=  20.7s
[CV] max_depth=3, n_estimators=3 .....................................
[CV]  max_depth=3, n_estimators=3, score=0.6625633885512254, total=  21.0s
```

```
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:  2.5min finished
```

Out[0]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
       estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=3, min_child_weight=1, missing=None,
       n_estimators=100, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=1, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=None,
       subsample=1, verbosity=1),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_estimators': [2, 3], 'max_depth': [3]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=3)
```

Best Params for BoW

In [0]:

```
best_d = gridclf.best_params_['max_depth']
best_n = gridclf.best_params_['n_estimators']
best_d, best_n
```

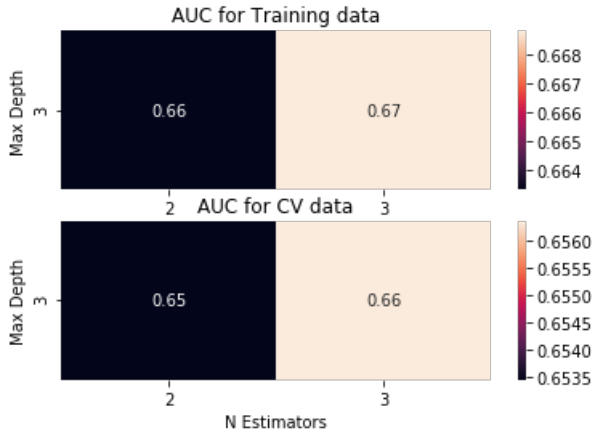
Out[0]:

```
(3, 3)
```

In [0]:

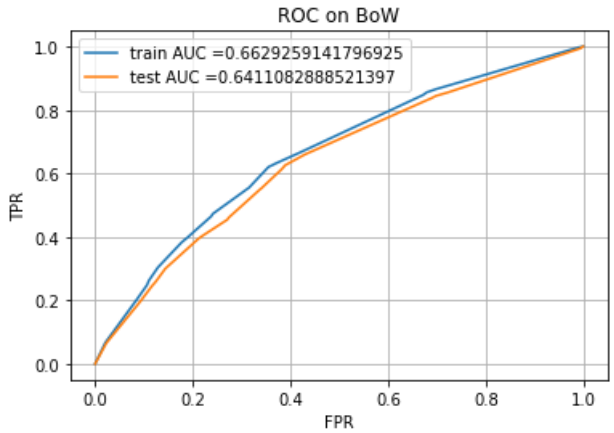
```
# Plot seaborn heatmap for gridsearchcv: https://www.kaggle.com/arindambanerjee/grid-search-simplified
max_depth_list = list(gridclf.cv_results_['param_max_depth'].data)
samplesplit_list = list(gridclf.cv_results_['param_n_estimators'].data)

plt.figure(1)
plt.subplot(211)
data = pd.DataFrame(data={'Max Depth':max_depth_list, \
                          'N Estimators':samplesplit_list , \
```

```
                           'AUC':gridclf.cv_results_['mean_train_score']})
data = data.pivot(index='Max Depth', columns='N Estimators', values='AUC')
sns.heatmap(data, annot=True).set_title('AUC for Training data')

plt.subplot(212)
data = pd.DataFrame(data={'Max Depth':max_depth_list, \
                           'N Estimators':samplesplit_list , \
                           'AUC':gridclf.cv_results_['mean_test_score']})
data = data.pivot(index='Max Depth', columns='N Estimators', values='AUC')
sns.heatmap(data, annot=True).set_title('AUC for CV data')

plt.show()
```



In [0]:

```
lr = xgb.XGBClassifier(max_depth=best_d, n_estimators=best_n, random_state=1)
lr.fit(tr_X, tr_y)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive clas
s
# not the predicted outputs

y_train_pred = lr.predict_proba(tr_X)[:,1]
y_test_pred = lr.predict_proba(ts_X)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(tr_y, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(ts_y, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC on BoW")
plt.grid()
plt.show()
```



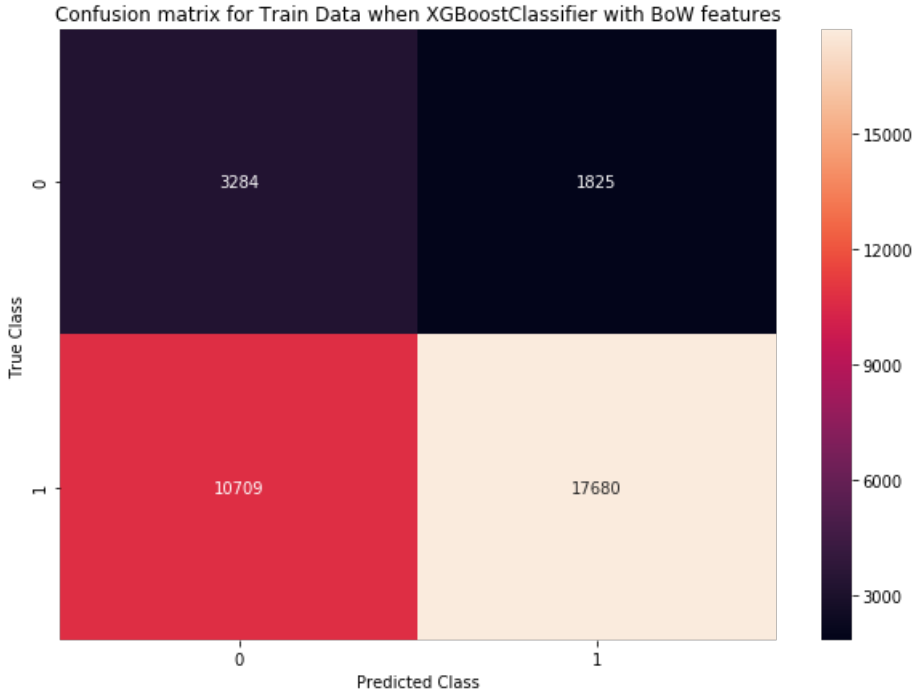
In [0]:

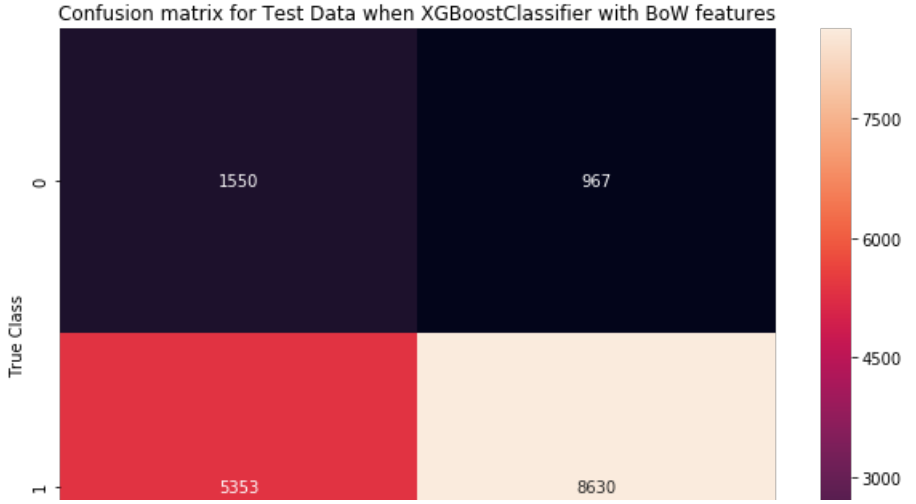
```
feature_names = 'BoW'
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
cm = metrics.confusion_matrix(tr_y, predict_with_best_t(y_train_pred, best_t))
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt="d")
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.title('Confusion matrix for Train Data when XGBoostClassifier with {0} features'.format(feature_nam
es))

print("Test confusion matrix")
cm = metrics.confusion_matrix(ts_y, predict_with_best_t(y_test_pred, best_t))
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt="d")
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.title('Confusion matrix for Test Data when XGBoostClassifier with {0} features'.format(feature_name
s))
```

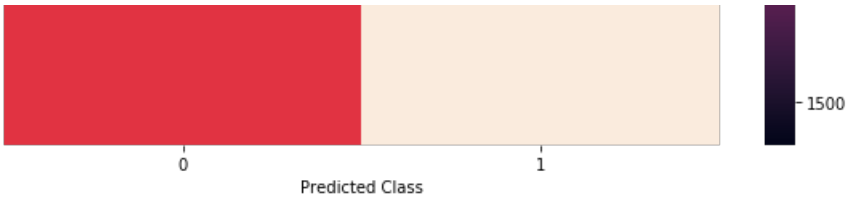
```
the maximum value of tpr*(1-fpr) 0.4003127398464642 for threshold 0.594
Train confusion matrix
Test confusion matrix
```

Out[0]:

```
Text(0.5, 1.0, 'Confusion matrix for Test Data when XGBoostClassifier with BoW features')
```







2.5.2 Applying XGBOOST on TFIDF, SET 2

In [0]:

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

In [86]:

```
gridclf = GridSearchCV(clf,parameter,cv=3,verbose=3,scoring='roc_auc', return_train_score=True)
gridclf.fit(tr_X,tr_y)
```

```
Fitting 3 folds for each of 2 candidates, totalling 6 fits
[CV] max_depth=3, n_estimators=2 .....................................

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

[CV]  max_depth=3, n_estimators=2, score=(train=0.654, test=0.634), total=  20.5s
[CV] max_depth=3, n_estimators=2 .....................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   22.0s remaining:    0.0s

[CV]  max_depth=3, n_estimators=2, score=(train=0.667, test=0.657), total=  18.6s
[CV] max_depth=3, n_estimators=2 .....................................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:   42.1s remaining:    0.0s

[CV]  max_depth=3, n_estimators=2, score=(train=0.656, test=0.646), total=  19.2s
[CV] max_depth=3, n_estimators=3 .....................................
[CV]  max_depth=3, n_estimators=3, score=(train=0.672, test=0.649), total=  21.3s
[CV] max_depth=3, n_estimators=3 .....................................
[CV]  max_depth=3, n_estimators=3, score=(train=0.675, test=0.657), total=  22.2s
[CV] max_depth=3, n_estimators=3 .....................................
[CV]  max_depth=3, n_estimators=3, score=(train=0.677, test=0.664), total=  25.0s

[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:  2.3min finished
```

Out[86]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                     colsample_bylevel=1, colsample_bynode=1,
                                     colsample_bytree=1, gamma=0,
                                     learning_rate=0.1, max_delta_step=0,
                                     max_depth=3, min_child_weight=1,
                                     missing=None, n_estimators=100, n_jobs=1,
                                     nthread=None, objective='binary:logistic',
                                     random_state=1, reg_alpha=0, reg_lambda=1,
                                     scale_pos_weight=1, seed=None, silent=None,
                                     subsample=1, verbosity=1),
             iid='warn', n_jobs=None,
             param_grid={'max_depth': [3], 'n_estimators': [2, 3]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=3)
```

Best Param on TFIDF

In [87]:

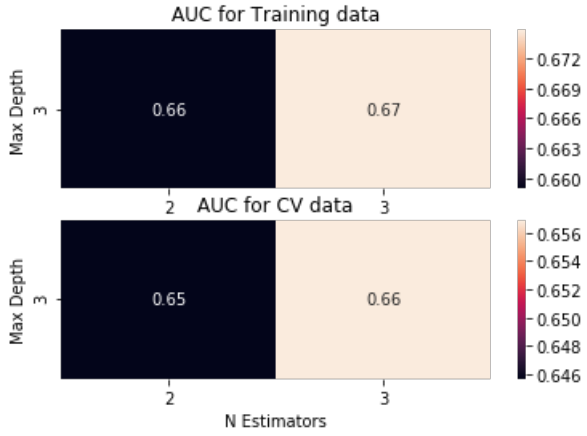
```
best_d = gridclf.best_params_['max_depth']
best_n = gridclf.best_params_['n_estimators']
best_d, best_n
```

Out[87]:

```
(3, 3)
```

In [89]:

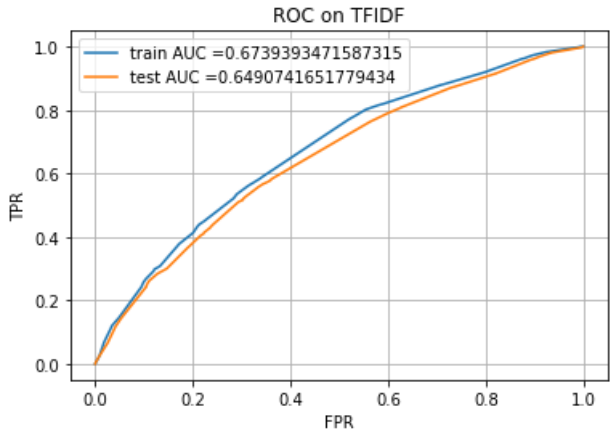
```
# Plot seaborn heatmap for gridsearchcv: https://www.kaggle.com/arindambanerjee/grid-search-simplified
max_depth_list = list(gridclf.cv_results_['param_max_depth'].data)
samplesplit_list = list(gridclf.cv_results_['param_n_estimators'].data)

plt.figure(1)
plt.subplot(211)
data = pd.DataFrame(data={'Max Depth':max_depth_list, \
                          'N Estimators':samplesplit_list , \
                          'AUC':gridclf.cv_results_['mean_train_score']})
data = data.pivot(index='Max Depth', columns='N Estimators', values='AUC')
sns.heatmap(data, annot=True).set_title('AUC for Training data')

plt.subplot(212)
data = pd.DataFrame(data={'Max Depth':max_depth_list, \
                          'N Estimators':samplesplit_list , \
                          'AUC':gridclf.cv_results_['mean_test_score']})
data = data.pivot(index='Max Depth', columns='N Estimators', values='AUC')
sns.heatmap(data, annot=True).set_title('AUC for CV data')

plt.show()
```



In [90]:

```
lr = xgb.XGBClassifier(max_depth=best_d, n_estimators=best_n, random_state=1)
lr.fit(tr_X, tr_y)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive clas
s
# not the predicted outputs

y_train_pred = lr.predict_proba(tr_X)[:,1]
y_test_pred = lr.predict_proba(ts_X)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(tr_y, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(ts_y, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC on TFIDF")
```

```
plt.grid()
plt.show()
```



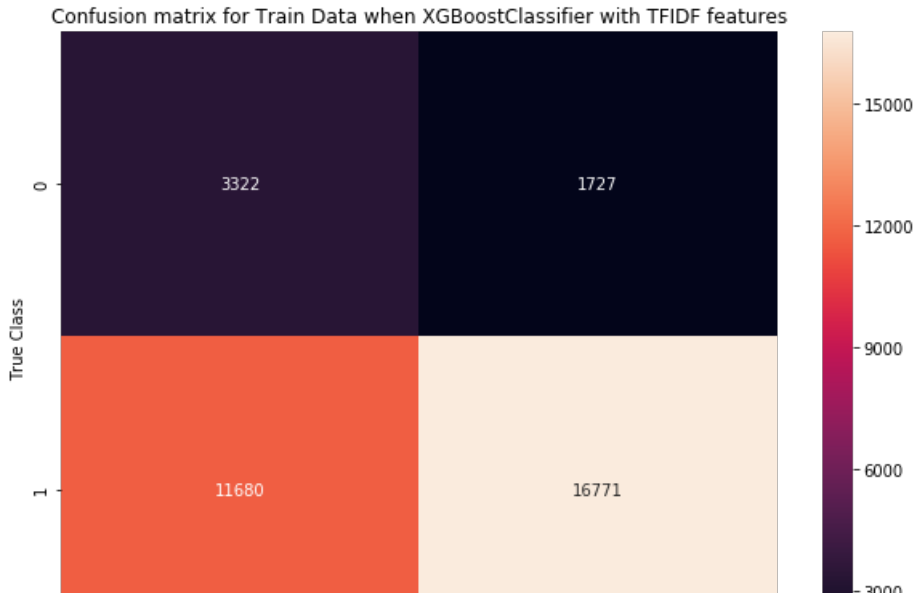
In [91]:

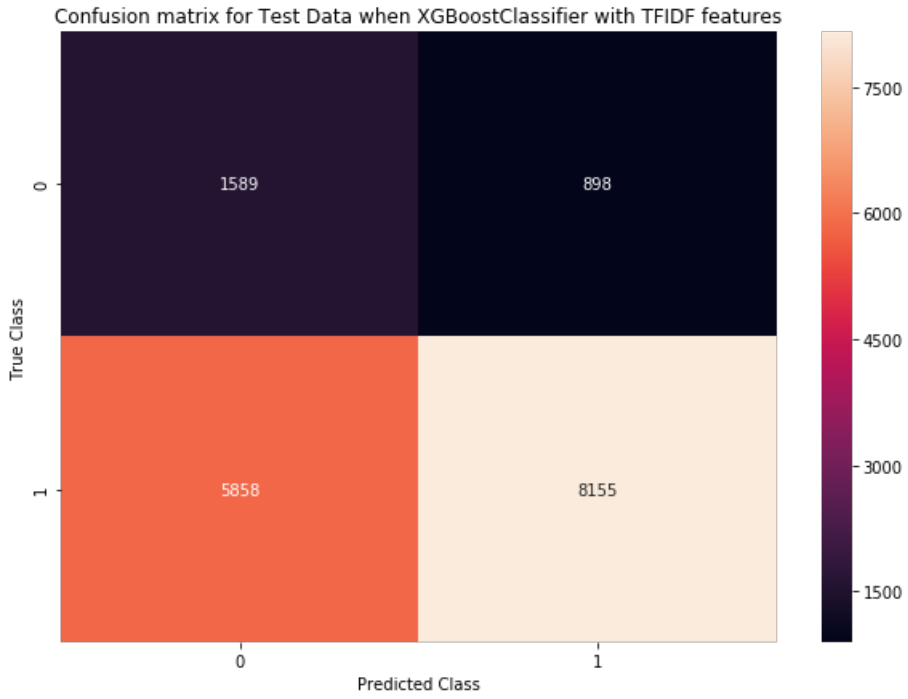
```
feature_names = 'TFIDF'
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
cm = metrics.confusion_matrix(tr_y, predict_with_best_t(y_train_pred, best_t))
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt="d")
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.title('Confusion matrix for Train Data when XGBoostClassifier with {0} features'.format(feature_nam
es))

print("Test confusion matrix")
cm = metrics.confusion_matrix(ts_y, predict_with_best_t(y_test_pred, best_t))
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt="d")
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.title('Confusion matrix for Test Data when XGBoostClassifier with {0} features'.format(feature_name
s))
```

```
the maximum value of tpr*(1-fpr) 0.38784275284594716 for threshold 0.59
Train confusion matrix
Test confusion matrix
```

Out[91]:

```
Text(0.5, 1.0, 'Confusion matrix for Test Data when XGBoostClassifier with TFIDF features')
```





2.5.4 Applying XGBOOST on AVG W2V, SET 3

In [0]:

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

In [109]:

```
gridclf = GridSearchCV(clf,parameter,cv=3,verbose=3,scoring='roc_auc', return_train_score=True)
gridclf.fit(tr_X,tr_y)
```

```
Fitting 3 folds for each of 2 candidates, totalling 6 fits
[CV] max_depth=3, n_estimators=2 .....................................
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[CV]  max_depth=3, n_estimators=2, score=(train=0.668, test=0.651), total=   3.6s
[CV] max_depth=3, n_estimators=2 .....................................
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    3.7s remaining:    0.0s
[CV]  max_depth=3, n_estimators=2, score=(train=0.669, test=0.651), total=   3.4s
[CV] max_depth=3, n_estimators=2 .....................................
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    7.3s remaining:    0.0s
[CV]  max_depth=3, n_estimators=2, score=(train=0.654, test=0.648), total=   3.3s
[CV] max_depth=3, n_estimators=3 .....................................
[CV]  max_depth=3, n_estimators=3, score=(train=0.675, test=0.652), total=   4.0s
[CV] max_depth=3, n_estimators=3 .....................................
[CV]  max_depth=3, n_estimators=3, score=(train=0.678, test=0.657), total=   3.9s
[CV] max_depth=3, n_estimators=3 .....................................
[CV]  max_depth=3, n_estimators=3, score=(train=0.676, test=0.664), total=   3.9s
```

```
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:   23.0s finished
```

Out[109]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                     colsample_bylevel=1, colsample_bynode=1,
                                     colsample_bytree=1, gamma=0,
                                     learning_rate=0.1, max_delta_step=0,
                                     max_depth=3, min_child_weight=1,
                                     missing=None, n_estimators=100, n_jobs=1,
                                     nthread=None, objective='binary:logistic',
                                     random_state=1, reg_alpha=0, reg_lambda=1,
                                     scale_pos_weight=1, seed=None, silent=None,
                                     subsample=1, verbosity=1),
             iid='warn', n_jobs=None,
             param_grid={'max_depth': [3], 'n_estimators': [2, 3]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=3)
```

Best Param on AVGW2V

In [110]:

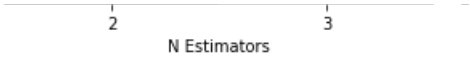
```
best_d = gridclf.best_params_['max_depth']
best_n = gridclf.best_params_['n_estimators']
best_d, best_n
```

Out[110]:

```
(3, 3)
```

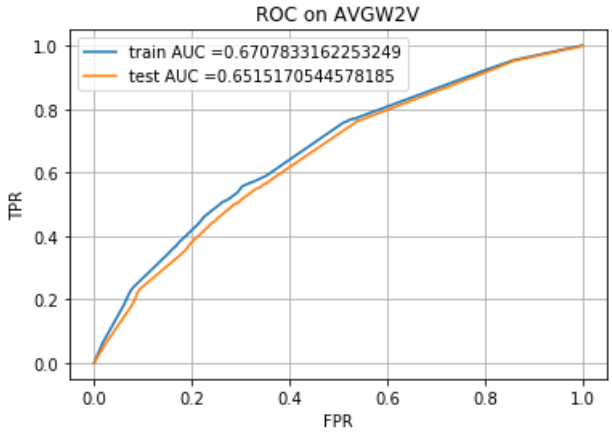
In [111]:

```
# Plot seaborn heatmap for gridsearchcv: https://www.kaggle.com/arindambanerjee/grid-search-simplified
max_depth_list = list(gridclf.cv_results_['param_max_depth'].data)
samplesplit_list = list(gridclf.cv_results_['param_n_estimators'].data)

plt.figure(1)
plt.subplot(211)
data = pd.DataFrame(data={'Max Depth':max_depth_list, \
                          'N Estimators':samplesplit_list , \
                          'AUC':gridclf.cv_results_['mean_train_score']})
data = data.pivot(index='Max Depth', columns='N Estimators', values='AUC')
sns.heatmap(data, annot=True).set_title('AUC for Training data')

plt.subplot(212)
data = pd.DataFrame(data={'Max Depth':max_depth_list, \
                          'N Estimators':samplesplit_list , \
                          'AUC':gridclf.cv_results_['mean_test_score']})
data = data.pivot(index='Max Depth', columns='N Estimators', values='AUC')
sns.heatmap(data, annot=True).set_title('AUC for CV data')

plt.show()
```



In [112]:

```
lr = xgb.XGBClassifier(max_depth=best_d, n_estimators=best_n, random_state=1)
lr.fit(tr_X, tr_y)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive clas
s
# not the predicted outputs

y_train_pred = lr.predict_proba(tr_X)[:,1]
y_test_pred = lr.predict_proba(ts_X)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(tr_y, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(ts_y, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC on AVGW2V")
plt.grid()
plt.show()
```



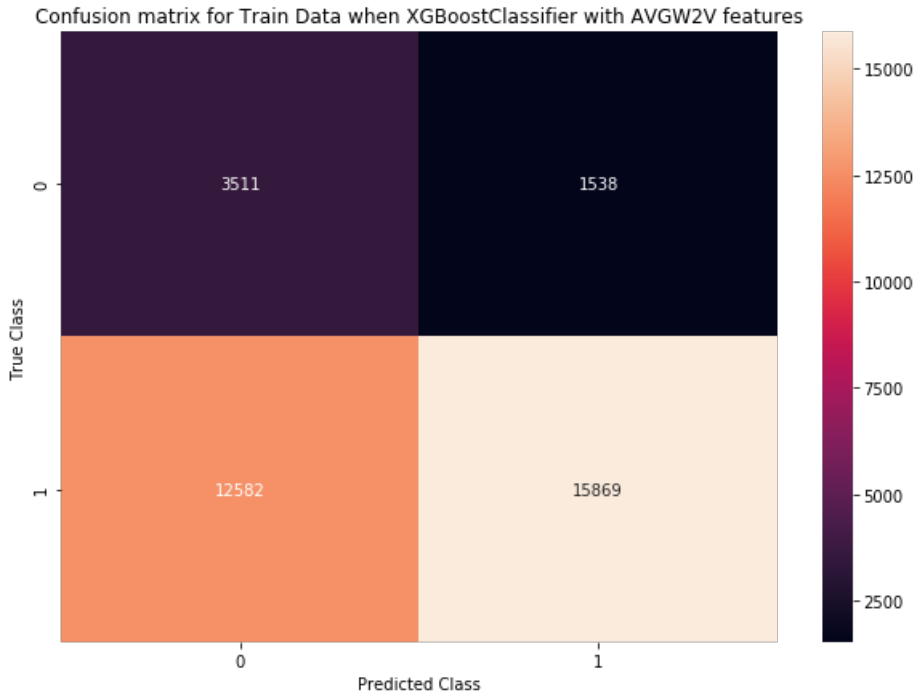
In [114]:

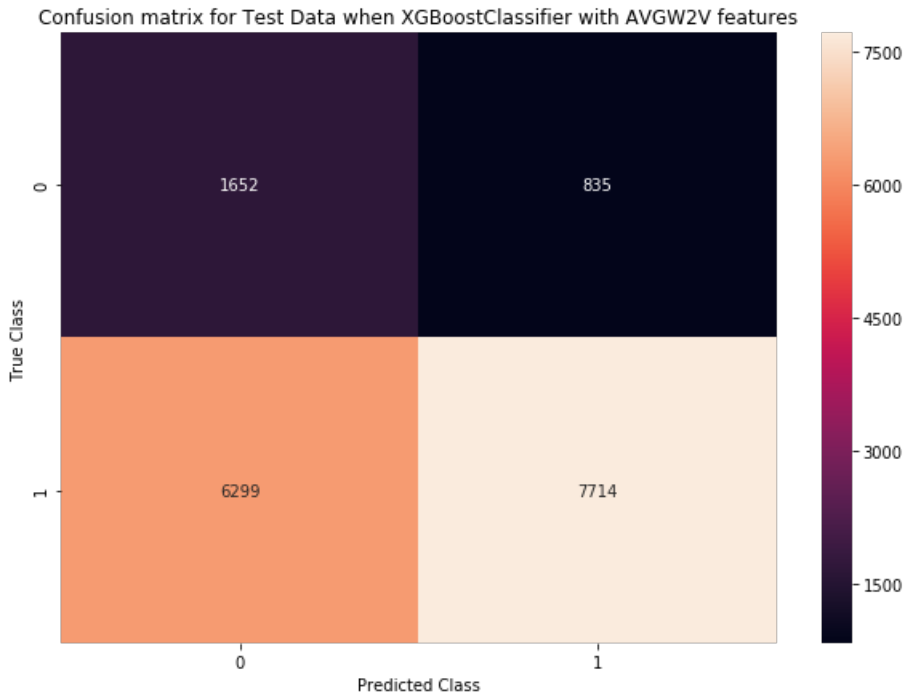
```
feature_names = 'AVGW2V'
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
cm = metrics.confusion_matrix(tr_y, predict_with_best_t(y_train_pred, best_t))
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt="d")
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.title('Confusion matrix for Train Data when XGBoostClassifier with {0} features'.format(feature_nam
es))

print("Test confusion matrix")
cm = metrics.confusion_matrix(ts_y, predict_with_best_t(y_test_pred, best_t))
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt="d")
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.title('Confusion matrix for Test Data when XGBoostClassifier with {0} features'.format(feature_name
s))
```

```
the maximum value of tpr*(1-fpr) 0.38786222390437686 for threshold 0.594
Train confusion matrix
Test confusion matrix
```

Out[114]:

```
Text(0.5, 1.0, 'Confusion matrix for Test Data when XGBoostClassifier with AVGW2V features')
```





2.5.3 Applying XGBOOST on TFIDF W2V, SET 4

In [131]:

```
gridclf = GridSearchCV(clf,parameter,cv=3,verbose=3,scoring='roc_auc', return_train_score=True)
gridclf.fit(tr_X,tr_y)
```

```
Fitting 3 folds for each of 2 candidates, totalling 6 fits
[CV] max_depth=3, n_estimators=2 ....................................
```

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

```
[CV]  max_depth=3, n_estimators=2, score=(train=0.669, test=0.642), total=   3.2s
[CV] max_depth=3, n_estimators=2 ....................................
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    3.4s remaining:    0.0s

[CV]  max_depth=3, n_estimators=2, score=(train=0.674, test=0.656), total=   3.2s
[CV] max_depth=3, n_estimators=2 .....................................

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

[CV]  max_depth=3, n_estimators=2, score=(train=0.665, test=0.654), total=   3.2s
[CV] max_depth=3, n_estimators=3 .....................................
[CV]  max_depth=3, n_estimators=3, score=(train=0.679, test=0.655), total=   4.1s
[CV] max_depth=3, n_estimators=3 .....................................
[CV]  max_depth=3, n_estimators=3, score=(train=0.681, test=0.664), total=   3.9s
[CV] max_depth=3, n_estimators=3 .....................................
[CV]  max_depth=3, n_estimators=3, score=(train=0.673, test=0.658), total=   3.9s

[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:   22.4s finished
```

Out[131]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                     colsample_bylevel=1, colsample_bynode=1,
                                     colsample_bytree=1, gamma=0,
                                     learning_rate=0.1, max_delta_step=0,
                                     max_depth=3, min_child_weight=1,
                                     missing=None, n_estimators=100, n_jobs=1,
                                     nthread=None, objective='binary:logistic',
                                     random_state=1, reg_alpha=0, reg_lambda=1,
                                     scale_pos_weight=1, seed=None, silent=None,
                                     subsample=1, verbosity=1),
             iid='warn', n_jobs=None,
             param_grid={'max_depth': [3], 'n_estimators': [2, 3]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=3)
```

Best Param on TFIDF W2V

In [132]:

```
best_d = gridclf.best_params_['max_depth']
best_n = gridclf.best_params_['n_estimators']
best_d, best_n
```

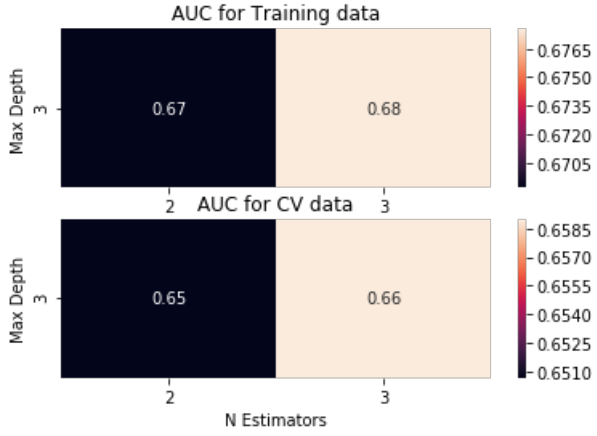
Out[132]:

```
(3, 3)
```

In [133]:

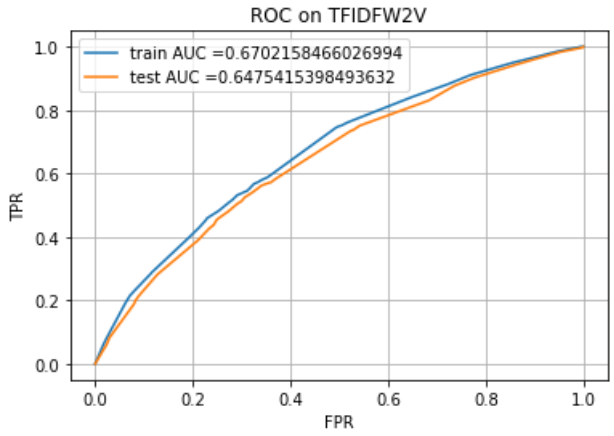
```
# Plot seaborn heatmap for gridsearchcv: https://www.kaggle.com/arindambanerjee/grid-search-simplified
max_depth_list = list(gridclf.cv_results_['param_max_depth'].data)
samplesplit_list = list(gridclf.cv_results_['param_n_estimators'].data)

plt.figure(1)
plt.subplot(211)
data = pd.DataFrame(data={'Max Depth':max_depth_list, \
                          'N Estimators':samplesplit_list , \
                          'AUC':gridclf.cv_results_['mean_train_score']})
data = data.pivot(index='Max Depth', columns='N Estimators', values='AUC')
sns.heatmap(data, annot=True).set_title('AUC for Training data')

plt.subplot(212)
data = pd.DataFrame(data={'Max Depth':max_depth_list, \
                          'N Estimators':samplesplit_list , \
                          'AUC':gridclf.cv_results_['mean_test_score']})
data = data.pivot(index='Max Depth', columns='N Estimators', values='AUC')
```

```
sns.heatmap(data, annot=True).set_title('AUC for CV data')

plt.show()
```



In [134]:

```
lr = xgb.XGBClassifier(max_depth=best_d, n_estimators=best_n, random_state=1)
lr.fit(tr_X, tr_y)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive clas
s
# not the predicted outputs

y_train_pred = lr.predict_proba(tr_X)[:,1]
y_test_pred = lr.predict_proba(ts_X)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(tr_y, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(ts_y, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC on TFIDFW2V")
plt.grid()
plt.show()
```



In [135]:

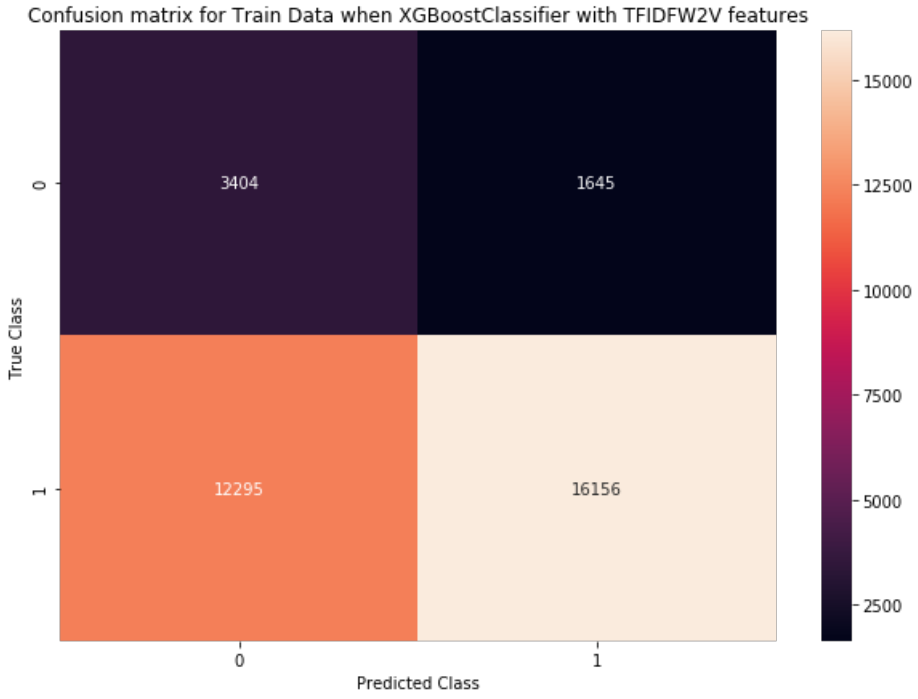
```
feature_names = 'TFIDFW2V'
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
cm = metrics.confusion_matrix(tr_y, predict_with_best_t(y_train_pred, best_t))
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt="d")
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.title('Confusion matrix for Train Data when XGBoostClassifier with {0} features'.format(feature_nam
```

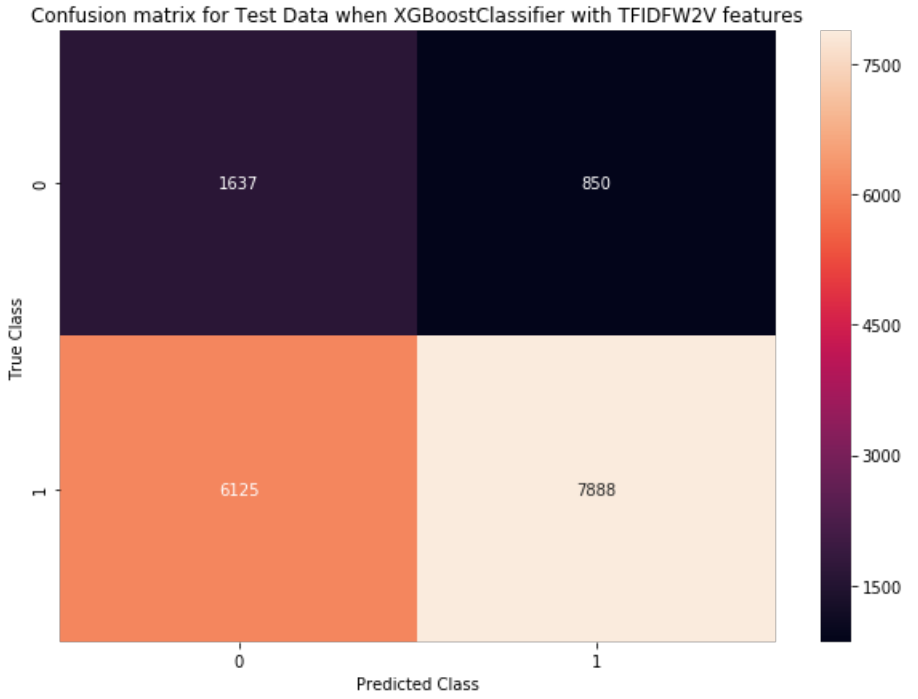
```
es))

print("Test confusion matrix")
cm = metrics.confusion_matrix(ts_y, predict_with_best_t(y_test_pred, best_t))
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt="d")
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.title('Confusion matrix for Test Data when XGBoostClassifier with {0} features'.format(feature_name
s))
```

```
the maximum value of tpr*(1-fpr) 0.3828428050217008 for threshold 0.595
Train confusion matrix
Test confusion matrix
```

Out[135]:

```
Text(0.5, 1.0, 'Confusion matrix for Test Data when XGBoostClassifier with TFIDFW2V features')
```





3. Conclusion

In [0]:

```
# Please compare all your models using Prettytable library
```

In [157]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ['Feature','Model','MaxDepth (Hyperparam1)','N_Estimators (Hyperparam2)','AUC']

x.add_row(['BoW','RandomForestClassifier',6,100,0.68826])
x.add_row(['BoW','XGBoostClassifier',3,3,0.64110])
x.add_row(['-----','-----'*4,'-----'*4,'-----'*5,'-----'*2])
x.add_row(['TFIDF','RandomForestClassifier',6,100,0.69216])
x.add_row(['TFIDF','XGBoostClassifier',3,3,0.64907])
x.add_row(['-----','-----'*4,'-----'*4,'-----'*5,'-----'*2])
x.add_row(['AVGW2V','RandomForestClassifier',5,64,0.67726])
x.add_row(['AVGW2V','XGBoostClassifier',3,3,0.65151])
x.add_row(['-----','-----'*4,'-----'*4,'-----'*5,'-----'*2])
x.add_row(['TFIDFW2V','RandomForestClassifier',5,100,0.68159])
x.add_row(['TFIDFW2V','XGBoostClassifier',3,3,0.64754])

print(x)
```

```
+----------+-----------------------+-----------------------+-----------------------------+------------
+
| Feature  |         Model         | MaxDepth (Hyperparam1) | N_Estimators (Hyperparam2) |    AUC
|
+----------+-----------------------+-----------------------+-----------------------------+------------
+
|   BoW    | RandomForestClassifier |           6           |            100              |  0.68826
|
|   BoW    |   XGBoostClassifier    |           3           |             3               |   0.6411
|
|  -----   |  --------------------  |  --------------------  | --------------------------  | ----------
|
|  TFIDF   | RandomForestClassifier |           6           |            100              |  0.69216
|
|  TFIDF   |   XGBoostClassifier    |           3           |             3               |  0.64907
|
|  -----   |  --------------------  |  --------------------  | --------------------------  | ----------
|
|  AVGW2V  | RandomForestClassifier |           5           |             64              |  0.67726
|
|  AVGW2V  |   XGBoostClassifier    |           3           |             3               |  0.65151
|
|  -----   |  --------------------  |  --------------------  | --------------------------  | ----------
|
| TFIDFW2V | RandomForestClassifier |           5           |            100              |  0.68159
|
| TFIDFW2V |   XGBoostClassifier    |           3           |             3               |  0.64754
|
+----------+-----------------------+-----------------------+-----------------------------+------------
+
```

In [ ]:

```

```