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	De
project_id	A unique identifier for the proposed project. **Example:**
project_title	Title of the project. **E** • Art Will Make You • First Gr
project_grade_category	Grade level of students for which the project is targeted. One of the enumerate • Grades • Gra • Gra • Grad
project_subject_categories	One or more (comma-separated) subject categories for the projec following enumerated list • Applied L • Care & • Health & • History & • Literacy & L • Math & • Music & T • Specia • **E** • Music & T • Literacy & Language, Math &
school_state	State where school is located (Two-letter U.S. p (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Posta **Exar**
project_subject_subcategories	One or more (comma-separated) subject subcategories for th **E** • L • Literature & Writing, Social S
project_resource_summary	An explanation of the resources needed for the project. **E** • My students need hands on literacy materials to sensory
project_essay_1	First applicat
project_essay_2	Second applicat
project_essay_3	Third applicat
project_essay_4	Fourth applicat
project_submitted_datetime	Datetime when project application was submitted. **Example:** 201 12:43
teacher_id	A unique identifier for the teacher of the proposed project. bdf8baa8fedef6bfeec7ae4ff

Feature	De
	Teacher's title. One of the following enumerate
`teacher_prefix`	• • • • • • T
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the sam **Exa**

* 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

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 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]:

```
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]:

```
project_data.head()
```

Out[5]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	proj
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY	
4	172407	p104768	be1f7507a41f8479dc06f047086a39ec	Mrs.	TX	

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/47
301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-stri
ng
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-pyth
on
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", "Warmt
h", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "M
ath & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace
 it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"M
ath & Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spa
ces
        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/47
301924/4084039

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

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", "Warmt
h", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "M
ath & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace
 it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"M
ath & Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spa
ces
        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	proj
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	

In [10]:

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

In [11]:

```
#to drop a row having nan https://stackoverflow.com/questions/13413590
project_data=project_data.dropna(subset=['teacher_prefix'])
```

In [12]:

```
# 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 seco
nd or third languages. We are a melting pot of refugees, immigrants, and n
ative-born Americans bringing the gift of language to our school. \r\n\r\n
We have over 24 languages represented in our English Learner program with
students at every level of mastery. We also have over 40 countries repres
ented with the families within our school. Each student brings a wealth o
f knowledge and experiences to us that open our eyes to new cultures, beli
efs, and respect.\"The limits of your language are the limits of your worl
d.\"-Ludwig Wittgenstein Our English learner's have a strong support syst
em at home that begs for more resources. Many times our parents are learn
ing to read and speak English along side of their children. Sometimes thi
s creates barriers for parents to be able to help their child learn phonet
ics, letter recognition, and other reading skills.\r\n\r\nBy providing the
se dvd's and players, students are able to continue their mastery of the E
nglish language even if no one at home is able to assist. All families wi
th students within the Level 1 proficiency status, will be a offered to be
a part of this program. These educational videos will be specially chosen
by the English Learner Teacher and will be sent home regularly to watch.
The videos are to help the child develop early reading skills.\r\n\r\nPare
nts that do not have access to a dvd player will have the opportunity to c
heck out a dvd player to use for the year. The plan is to use these video
s and educational dvd's for the years to come for other EL students.\r\nna
nnan

==

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% a
re minority students. \r\nThe school has a vibrant community that loves to
get together and celebrate. Around Halloween there is a whole school parad
e to show off the beautiful costumes that students wear. On Cinco de Mayo
we put on a big festival with crafts made by the students, dances, and gam
es. At the end of the year the school hosts a carnival to celebrate the ha
rd work put in during the school year, with a dunk tank being the most pop
ular activity.My students will use these five brightly colored Hokki stool
s 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 i
ndividual 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 reading times. The rest of the day they will be used by th
e 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 mis
sing, my students always say more Hokki Stools. They can't get their fill
of the 5 stools we already have. When the students are sitting in group wi
th me on the Hokki Stools, they are always moving, but at the same time do
ing their work. Anytime the students get to pick where they can sit, the H
okki 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 hours a day. The Hokki stools will be a compromise that allow my student
s to do desk work and move at the same time. These stools will help studen
ts 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, th
ese chairs will take away the barrier that exists in schools for a child w
ho can't sit still.nannan

==

How do you remember your days of school? Was it in a sterile environment w
ith plain walls, rows of desks, and a teacher in front of the room? A typi
cal day in our room is nothing like that. I work hard to create a warm inv
iting themed room for my students look forward to coming to each day.\r\n
\r\nMy class is made up of 28 wonderfully unique boys and girls of mixed r

aces 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 school is an \"open classroom\" concept, which is very unique as there are no walls separating the classrooms. These 9 and 10 year-old students are very eager learners; they are like sponges, absorbing all the information and experiences and keep on wanting more.With these resources such as the comfy red throw pillows and the whimsical nautical hanging decor and the blue fish nets, I will be able to help create the mood in our classroom setting to be one of a themed nautical environment. Creating a classroom environment is very important in the success in each and every child's education. The nautical photo props will be used with each child as they step foot into our classroom for the first time on Meet the Teacher evening. I'll take pictures of each child with them, have them developed, and then hung in our classroom ready for their first day of 4th grade. This kind gesture will set the tone before even the first 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 our classroom a fun, inviting, learning environment from day one.\r\n\r\nIt costs lost of money out of my 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 wonderful students are 3, 4, and 5 years old. We are located in a small town outside of Charlotte, NC. All of my 22 students are children of school district employees.\r\nMy students are bright, energetic, and they love to learn! They love hands-on activities that get them moving. Like most preschoolers, they enjoy music and creating different things. \r\nAll of my students come from wonderful families that are very supportive of our classroom. Our parents enjoy watching their children's growth as much as we do!These materials will help me teach my students all about the life cycle of a butterfly. We will watch as the Painted Lady caterpillars grow bigger and build their chrysalis. After a few weeks they will emerge from the chrysalis as beautiful butterflies! We already have a net for the chrysalises, but we still need the caterpillars and feeding station.\r\nThis will be an unforgettable experience for my students. My student absolutely love hands-on materials. They learn so much from getting to touch and manipulate different things. The supporting materials I have selected will help my students understand the life cycle through exploration.nannan

==========

The students in my classroom are learners, readers, writers, explorers, scientists, and mathematicians! The potential in these first graders is endless! Each day they come in grinning from ear-to-ear and ready to learn more. \r\nI choose curriculum that is real and relevant to the students, but it will also prepare them for their futures. These kids are encouraged to investigate concepts that are exciting for them and I hope we can keep this momentum going! These kids deserve the best, please help me give that to them! Thank you! :)These kits include a wide variety of science, technology, engineering, and mechanics for my students to dive into at the beginning of the year. I want them to hit the ground running this upcoming year and these kits always encourage high interest.\r\nWho wouldn't want to build their own roller coaster, design a car, or even think critically to make a bean bag bounce as far as it can go?? These kits will also shows students potential careers that they may have never heard of before!\r\nAny donations would be greatly appreciated and my students will know exactly who to thank for them!nannan

==========

In [13]:

```
# 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 [14]:

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

```
My wonderful students are 3, 4, and 5 years old.  We are located in a smal
l town outside of Charlotte, NC.  All of my 22 students are children of sc
hool district employees.\r\nMy students are bright, energetic, and they lo
ve to learn!  They love hands-on activities that get them moving.  Like mo
st preschoolers, they enjoy music and creating different things. \r\nAll o
f my students come from wonderful families that are very supportive of our
classroom.  Our parents enjoy watching their children is growth as much as
we do!These materials will help me teach my students all about the life cy
cle of a butterfly.  We will watch as the Painted Lady caterpillars grow b
igger and build their chrysalis.  After a few weeks they will emerge from
the chrysalis as beautiful butterflies!  We already have a net for the chr
ysalises, but we still need the caterpillars and feeding station.\r\nThis
will be an unforgettable experience for my students.  My student absolutel
y love hands-on materials.  They learn so much from getting to touch and m
anipulate different things.  The supporting materials I have selected will
help my students understand the life cycle through exploration.nannan
==================================================
```

In [15]:

```
# \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 wonderful students are 3, 4, and 5 years old.  We are located in a smal
l town outside of Charlotte, NC.  All of my 22 students are children of sc
hool district employees.  My students are bright, energetic, and they love
to learn!  They love hands-on activities that get them moving.  Like most
preschoolers, they enjoy music and creating different things.   All of my
students come from wonderful families that are very supportive of our clas
sroom.  Our parents enjoy watching their children is growth as much as we
do!These materials will help me teach my students all about the life cycle
of a butterfly.  We will watch as the Painted Lady caterpillars grow bigge
r and build their chrysalis.  After a few weeks they will emerge from the
chrysalis as beautiful butterflies!  We already have a net for the chrysal
ises, but we still need the caterpillars and feeding station.  This will b
e an unforgettable experience for my students.  My student absolutely love
hands-on materials.  They learn so much from getting to touch and manipula
te different things.  The supporting materials I have selected will help m
y students understand the life cycle through exploration.nannan
```

In [16]:

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

```
My wonderful students are 3 4 and 5 years old We are located in a small to
wn outside of Charlotte NC All of my 22 students are children of school di
strict employees My students are bright energetic and they love to learn T
hey love hands on activities that get them moving Like most preschoolers t
hey enjoy music and creating different things All of my students come from
wonderful families that are very supportive of our classroom Our parents e
njoy watching their children is growth as much as we do These materials wi
ll help me teach my students all about the life cycle of a butterfly We wi
ll watch as the Painted Lady caterpillars grow bigger and build their chry
salis After a few weeks they will emerge from the chrysalis as beautiful b
utterflies We already have a net for the chrysalises but we still need the
caterpillars and feeding station This will be an unforgettable experience
for my students My student absolutely love hands on materials They learn s
o much from getting to touch and manipulate different things The supportin
g materials I have selected will help my students understand the life cycl
e through exploration nannan
```

In [17]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', '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 [18]:

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

```
100%|██████████| 109245/109245 [01:12<00:00, 1506.00it/s]
```

In [19]:

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

Out[19]:

```
'my wonderful students 3 4 5 years old we located small town outside charl
otte nc all 22 students children school district employees my students bri
ght energetic love learn they love hands activities get moving like presch
oolers enjoy music creating different things all students come wonderful f
amilies supportive classroom our parents enjoy watching children growth mu
ch these materials help teach students life cycle butterfly we watch paint
ed lady caterpillars grow bigger build chrysalis after weeks emerge chrysa
lis beautiful butterflies we already net chrysalises still need caterpilla
rs feeding station this unforgettable experience students my student absol
utely love hands materials they learn much getting touch manipulate differ
ent things the supporting materials i selected help students understand li
fe cycle exploration nannan'
```

1.5 Preparing data for models

In [20]:

```
project_data.columns
```

Out[20]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', 'project_ti
tle',
       'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approve
d',
       'clean_categories', 'clean_subcategories', 'essay'],
      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/ (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/)

In [21]:

```
# 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)
```

```
['Health_Sports', 'AppliedLearning', 'SpecialNeeds', 'Literacy_Language',
'Warmth', 'Care_Hunger', 'History_Civics', 'Math_Science', 'Music_Arts']
Shape of matrix after one hot encodig  (109245, 9)
```

In [22]:

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

```
['FinancialLiteracy', 'VisualArts', 'Economics', 'Mathematics', 'History_G
eography', 'CharacterEducation', 'EnvironmentalScience', 'CommunityServic
e', 'NutritionEducation', 'College_CareerPrep', 'ParentInvolvement', 'Earl
yDevelopment', 'AppliedSciences', 'Gym_Fitness', 'ForeignLanguages', 'Team
Sports', 'Health_Wellness', 'Other', 'Civics_Government', 'SpecialNeeds',
'Extracurricular', 'Literature_Writing', 'Health_LifeScience', 'SocialScie
nces', 'Literacy', 'Warmth', 'Care_Hunger', 'ESL', 'Music', 'PerformingArt
s']
Shape of matrix after one hot encodig  (109245, 30)
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [23]:

```
# We are considering only the words which appeared in at least 10 documents(rows or pro
jects).
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  (109245, 16623)
```

1.5.2.2 TFIDF vectorizer

In [24]:

```
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  (109245, 16623)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [25]:

```
'''
# 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-p
ickle-to-save-and-load-variables-in-python/

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


'''
```

Out[25]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/3823034
9/4084039\ndef loadGloveModel(gloveFile):\n    print ("Loading Glove Mode
l")\n    f = open(gloveFile,\'r\', encoding="utf8")\n    model = {}\n    f
or line in tqdm(f):\n        splitLine = line.split()\n        word = spli
tLine[0]\n        embedding = np.array([float(val) for val in splitLine
[1:]])\n        model[word] = embedding\n    print ("Done.",len(model)," w
ords loaded!")\n    return model\nmodel = loadGloveModel(\'glove.42B.300d.
txt\')\n\n# ============================\nOutput:\n    \nLoading Glove Mod
el\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n# ====
========================\n\nwords = []\nfor i in preproced_texts:\n    wor
ds.extend(i.split(\' \'))\n\nfor i in preproced_titles:\n    words.extend
(i.split(\' \'))\nprint("all the words in the coupus", len(words))\nwords
= set(words)\nprint("the unique words in the coupus", len(words))\n\ninter
_words = set(model.keys()).intersection(words)\nprint("The number of words
that are present in both glove vectors and our coupus",       len(inter_wo
rds),"(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_courpu
s = {}\nwords_glove = set(model.keys())\nfor i in words:\n    if i in word
s_glove:\n        words_courpus[i] = model[i]\nprint("word 2 vec length",
len(words_courpus))\n\n\n\n# stronging variables into pickle files python: h
ttp://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
pickle.dump(words_courpus, f)\n\n\n'
```

In [26]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-p
ickle-to-save-and-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 [27]:

```
# 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%|██████████| 109245/109245 [00:42<00:00, 2596.29it/s]

109245
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [28]:

```
# 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 [29]:

```
# 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((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting 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%|██████████| 109245/109245 [04:03<00:00, 448.65it/s]

109245
300
```

1.5.3 Vectorizing Numerical features

In [30]:

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

In [31]:

```
# 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("Mean :" +str(price_scalar.mean_[0])+", Standard deviation :" +str(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))
```

```
Mean :298.1152448166964, Standard deviation :367.49642545627506
```

In [32]:

```
price_standardized
```

Out[32]:

```
array([[-0.39052147],
       [ 0.00240752],
       [ 0.5952024 ],
       ...,
       [-0.1582471 ],
       [-0.61242839],
       [-0.51215531]])
```

1.5.4 Merging all the above features

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

In [33]:

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

```
(109245, 9)
(109245, 30)
(109245, 16623)
(109245, 1)
```

In [34]:

```
# 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[34]:

```
(109245, 16663)
```

Computing Sentiment Scores

In [35]:

```
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 the biggest enthusiasm \
for learning my students learn in many different ways using all of our senses and multi
ple intelligences i use a wide range\
of techniques to help all my students succeed students in my class come from a variety
 of different backgrounds which makes\
for wonderful sharing of experiences and cultures including native americans our school
is a caring community of successful \
learners which can be seen through collaborative student project based learning in and
 out of the classroom kindergarteners \
in my class love to work with hands on materials and have many different opportunities
 to practice a skill before it is\
mastered having the social skills to work cooperatively with friends is a crucial aspec
t of the kindergarten 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 will take their idea \
and create common core cooking lessons where we learn important math and writing concep
ts while cooking delicious healthy \
food for snack time my students will have a grounded appreciation for the work that wen
t into making the food and knowledge \
of where the ingredients came from as well as how it is healthy for their bodies this p
roject would expand our learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to make hom
emade applesauce make our own bread \
and mix up healthy plants from our classroom garden in the spring we will also create o
ur own cookbooks to be printed and \
shared with families students will gain math and literature skills as well as a life lo
ng 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
```

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

Assignment 9: RF and GBDT

Response Coding: Example

> The response tabel is built only on train dataset. For a category which is not there in train data and present in test data, we will encode them with default values Ex: in our test data if have State: D then we encode it as [0.5, 0.05]

1. **Apply both Random Forrest and GBDT on these feature sets**
    - Set 1: categorical(instead of one hot encoding, try response coding (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/): use probability values), numerical features + project_title(BOW) + preprocessed_eassay (BOW)
    - Set 2: categorical(instead of one hot encoding, try response coding (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/): use probability values), numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
    - Set 3: categorical(instead of one hot encoding, try response coding (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/): use probability values), numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
    - Set 4: categorical(instead of one hot encoding, try response coding (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/): 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)**
    - Find the best hyper parameter which will give the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) 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 (https://seaborn.pydata.org/generated/seaborn.heatmap.html) 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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points

4. **Conclusion**

   - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link (http://zetcode.com/python/prettytable/)

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. (https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf)

2. Random Forest and GBDT

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

In [36]:

```
project_data1 = project_data
project_data1.head(3)
```

Out[36]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	proj
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	

In [37]:

```
y = project_data1['project_is_approved'].values
project_data1.drop(['project_is_approved'], axis=1, inplace=True)
```

In [38]:

```
X = project_data1
# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [39]:

```
#creating new datasets from X_train and X_test for response coding
new_train_data = X_train
new_train_data['c_label'] = y_train
new_test_data = X_test
new_test_data['c_label'] = y_test
```

2.2.1 Response coding for categorical features:teacher_prefix

In [40]:

```
list1 = (new_train_data.teacher_prefix.unique()).tolist()

prob0 = [None]*len(list1)
prob1 = [None]*len(list1)

for l in list1:
    total_data = new_train_data[new_train_data.teacher_prefix==l]
    total_project = total_data.shape[0]
    total_rejected_project = total_data[total_data.c_label==0].shape[0]
    total_accepted_project = total_data[total_data.c_label==1].shape[0]
    i=list1.index(l)
    prob0[i] = total_rejected_project/total_project
    prob1[i] = total_accepted_project/total_project
for l in list1:
    i=list1.index(l)
    new_train_data.loc[(new_train_data.teacher_prefix == l), 'teacher_0'] = prob0[i]
    new_train_data.loc[(new_train_data.teacher_prefix == l), 'teacher_1'] = prob1[i]
```

In [41]:

```
X_train_teacher_0  = (X_train['teacher_0'].values).reshape(-1,1)
X_train_teacher_0 = X_train_teacher_0.astype(float)
X_train_teacher_1  = (X_train['teacher_1'].values).reshape(-1,1)
X_train_teacher_1 = X_train_teacher_1.astype(float)
```

In [42]:

```
#appending the calculated probabilities for categorical features from train datasets to
test datasets

list1_test = (new_test_data.teacher_prefix.unique()).tolist()

for l in list1:
    i=list1.index(l)
    new_test_data.loc[(new_test_data.teacher_prefix == l), 'teacher_0'] = prob0[i]
    new_test_data.loc[(new_test_data.teacher_prefix == l), 'teacher_1'] = prob1[i]

#for new category in teacher_prefix.we will put teacher_0=0.5, teacher_1=0.5

def Diff(li1, li2):
    return (list(set(li1) - set(li2)))

new_list = Diff(list1_test, list1)

for m in new_list:
    new_test_data.loc[(new_test_data.teacher_prefix == m), 'grade_0'] = 0.5
    new_test_data.loc[(new_test_data.teacher_prefix == m), 'grade_1'] = 0.5
```

In [43]:

```
X_test_teacher_0  = (X_test['teacher_0'].values).reshape(-1,1)
X_test_teacher_0 = X_test_teacher_0.astype(float)
X_test_teacher_1  = (X_test['teacher_1'].values).reshape(-1,1)
X_test_teacher_1 = X_test_teacher_1.astype(float)
```

2.2.2 Response coding for categorical features:project_grade_category

In [44]:

```
list1 = (new_train_data.project_grade_category.unique()).tolist()

prob0 = [None]*len(list1)
prob1 = [None]*len(list1)

for l in list1:
    total_data = new_train_data[new_train_data.project_grade_category==l]
    total_project = total_data.shape[0]
    total_rejected_project = total_data[total_data.c_label==0].shape[0]
    total_accepted_project = total_data[total_data.c_label==1].shape[0]
    i=list1.index(l)
    prob0[i] = total_rejected_project/total_project
    prob1[i] = total_accepted_project/total_project
for l in list1:
    i=list1.index(l)
    new_train_data.loc[(new_train_data.project_grade_category == l), 'grade_0'] = prob0
[i]
    new_train_data.loc[(new_train_data.project_grade_category == l), 'grade_1'] = prob1
[i]
```

In [45]:

```
X_train_grade_0 = (X_train['grade_0'].values).reshape(-1,1)
X_train_grade_0 = X_train_grade_0.astype(float)
X_train_grade_1 = (X_train['grade_1'].values).reshape(-1,1)
X_train_grade_1 = X_train_grade_0.astype(float)
```

In [46]:

```
#appending the calculated probabilities for categorical features from train datasets to
test datasets

list1_test = (new_test_data.project_grade_category.unique()).tolist()

for l in list1:
    i=list1.index(l)
    new_test_data.loc[(new_test_data.project_grade_category == l), 'grade_0'] = prob0[i
]
    new_test_data.loc[(new_test_data.project_grade_category == l), 'grade_1'] = prob1[i
]

#for new category in project_grade_category.we will put grade_0=0.5, grade_1=0.5

def Diff(li1, li2):
    return (list(set(li1) - set(li2)))

new_list = Diff(list1_test, list1)

for m in new_list:
    new_test_data.loc[(new_test_data.project_grade_category == m), 'grade_0'] = 0.5
    new_test_data.loc[(new_test_data.project_grade_category == m), 'grade_1'] = 0.5
```

In [47]:

```
X_test_grade_0 = (X_test['grade_0'].values).reshape(-1,1)
X_test_grade_0 = X_test_grade_0.astype(float)
X_test_grade_1 = (X_test['grade_1'].values).reshape(-1,1)
X_test_grade_1 = X_test_grade_1.astype(float)
```

2.2.3 Response coding for categorical features:school_state

In [48]:

```
list1 = (new_train_data.school_state.unique()).tolist()

prob0 = [None]*len(list1)
prob1 = [None]*len(list1)

for l in list1:
    total_data = new_train_data[new_train_data.school_state==l]
    total_project = total_data.shape[0]
    total_rejected_project = total_data[total_data.c_label==0].shape[0]
    total_accepted_project = total_data[total_data.c_label==1].shape[0]
    i=list1.index(l)
    prob0[i] = total_rejected_project/total_project
    prob1[i] = total_accepted_project/total_project
for l in list1:
    i=list1.index(l)
    new_train_data.loc[(new_train_data.school_state == l), 'state_0'] = prob0[i]
    new_train_data.loc[(new_train_data.school_state == l), 'state_1'] = prob1[i]
```

In [49]:

```
X_train_state_0 = (X_train['state_0'].values).reshape(-1,1)
X_train_state_0 = X_train_state_0.astype(float)
X_train_state_1 = (X_train['state_1'].values).reshape(-1,1)
X_train_state_1 = X_train_state_0.astype(float)
```

In [50]:

```
#appending the calculated probabilities for categorical features from train datasets to
test datasets

list1_test = (new_test_data.school_state.unique()).tolist()

for l in list1:
    i=list1.index(l)
    new_test_data.loc[(new_test_data.school_state == l), 'state_0'] = prob0[i]
    new_test_data.loc[(new_test_data.school_state == l), 'state_1'] = prob1[i]

#for new category in school_state.we will put state_0=0.5,state_1=0.5

def Diff(li1, li2):
    return (list(set(li1) - set(li2)))

new_list = Diff(list1_test, list1)

for m in new_list:
    new_test_data.loc[(new_test_data.school_state == m), 'state_0'] = 0.5
    new_test_data.loc[(new_test_data.school_state == m), 'state_1'] = 0.5
```

In [51]:

```
X_test_state_0 = (X_test['state_0'].values).reshape(-1,1)
X_test_state_0 = X_test_state_0.astype(float)
X_test_state_1 = (X_test['state_1'].values).reshape(-1,1)
X_test_state_1 = X_test_state_1.astype(float)
```

2.2.4 Response coding for categorical features:clean_categories

In [52]:

```
list1 = (new_train_data.clean_categories.unique()).tolist()

prob0 = [None]*len(list1)
prob1 = [None]*len(list1)

for l in list1:
    total_data = new_train_data[new_train_data.clean_categories==l]
    total_project = total_data.shape[0]
    total_rejected_project = total_data[total_data.c_label==0].shape[0]
    total_accepted_project = total_data[total_data.c_label==1].shape[0]
    i=list1.index(l)
    prob0[i] = total_rejected_project/total_project
    prob1[i] = total_accepted_project/total_project
for l in list1:
    i=list1.index(l)
    new_train_data.loc[(new_train_data.clean_categories == l), 'categories_0'] = prob0[
i]
    new_train_data.loc[(new_train_data.clean_categories == l), 'categories_1'] = prob1[
i]
```

In [53]:

```
X_train_categories_0 = (X_train['categories_0'].values).reshape(-1,1)
X_train_categories_0 = X_train_categories_0.astype(float)
X_train_categories_1 = (X_train['categories_1'].values).reshape(-1,1)
X_train_categories_1 = X_train_categories_1.astype(float)
```

In [54]:

```
#appending the calculated probabilities for categorical features from train datasets to
test datasets

list1_test = (new_test_data.clean_categories.unique()).tolist()

for l in list1:
    i=list1.index(l)
    new_test_data.loc[(new_test_data.clean_categories == l), 'categories_0'] = prob0[i]
    new_test_data.loc[(new_test_data.clean_categories == l), 'categories_1'] = prob1[i]

#for new category in categorical features. we will put categories_0=0.5,categories_1=0.
5
def Diff(li1, li2):
    return (list(set(li1) - set(li2)))

new_list = Diff(list1_test, list1)

for m in new_list:
    new_test_data.loc[(new_test_data.clean_categories == m), 'categories_0'] = 0.5
    new_test_data.loc[(new_test_data.clean_categories == m), 'categories_1'] = 0.5
```

In [55]:

```
X_test_categories_0 = (X_test['categories_0'].values).reshape(-1,1)
X_test_categories_0 = X_test_categories_0.astype(float)
X_test_categories_1 = (X_test['categories_1'].values).reshape(-1,1)
X_test_categories_1 = X_test_categories_1.astype(float)
```

2.2.5 Response coding for categorical features:clean_subcategories

In [56]:

```
#categorical features:clean_subcategories
list1 = (new_train_data.clean_subcategories.unique()).tolist()

prob0 = [None]*len(list1)
prob1 = [None]*len(list1)

for l in list1:
    total_data = new_train_data[new_train_data.clean_subcategories==l]
    total_project = total_data.shape[0]
    total_rejected_project = total_data[total_data.c_label==0].shape[0]
    total_accepted_project = total_data[total_data.c_label==1].shape[0]
    i=list1.index(l)
    prob0[i] = total_rejected_project/total_project
    prob1[i] = total_accepted_project/total_project
for l in list1:
    i=list1.index(l)
    new_train_data.loc[(new_train_data.clean_subcategories == l), 'sub_categories_0'] =
prob0[i]
    new_train_data.loc[(new_train_data.clean_subcategories == l), 'sub_categories_1'] =
prob1[i]
```

In [57]:

```
X_train_sub_categories_0 = (X_train['sub_categories_0'].values).reshape(-1,1)
X_train_sub_categories_0 = X_train_sub_categories_0.astype(float)
X_train_sub_categories_1 = (X_train['sub_categories_1'].values).reshape(-1,1)
X_train_sub_categories_1 = X_train_sub_categories_1.astype(float)
```

In [58]:

```
#appending the calculated probabilities for categorical features from train datasets to
test datasets

list1_test = (new_test_data.clean_subcategories.unique()).tolist()

for l in list1:
    i=list1.index(l)
    new_test_data.loc[(new_test_data.clean_subcategories == l), 'sub_categories_0'] = p
rob0[i]
    new_test_data.loc[(new_test_data.clean_subcategories == l), 'sub_categories_1'] = p
rob1[i]

#for new category in subcategorical features. we will put sub_categories_0=0.5, sub_cat
egories_1=0.5
def Diff(li1, li2):
    return (list(set(li1) - set(li2)))

new_list = Diff(list1_test, list1)

for m in new_list:
    new_test_data.loc[(new_test_data.clean_subcategories == m), 'sub_categories_0'] =
0.5
    new_test_data.loc[(new_test_data.clean_subcategories == m), 'sub_categories_1'] =
0.5
```

In [59]:

```
X_test_sub_categories_0 = (X_test['sub_categories_0'].values).reshape(-1,1)
X_test_sub_categories_0 = X_test_sub_categories_0.astype(float)
X_test_sub_categories_1 = (X_test['sub_categories_1'].values).reshape(-1,1)
X_test_sub_categories_1 = X_test_sub_categories_1.astype(float)
```

In [60]:

```
new_train_data.drop(['c_label'], axis=1, inplace=True)
X_train = new_train_data
new_test_data.drop(['c_label'], axis=1, inplace=True)
X_test = new_test_data
```

2.2.6 Normalizing the numerical features: Price

In [61]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
normalizer.fit(X_train['price'].values.reshape(-1,1))

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(-1,1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(73194, 1) (73194,)
(36051, 1) (36051,)
====================================================================================================
```

2.2.7 Normalizing the numerical features: teacher_number_of_previously_posted_projects

In [62]:

```
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

X_train_tnppp_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_test_tnppp_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_tnppp_norm.shape, y_train.shape)
print(X_test_tnppp_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(73194, 1) (73194,)
(36051, 1) (36051,)
====================================================================================================
```

2.3 Make Data Model Ready: encoding eassay, and project_title

2.3.1.1 Text preprocessing

In [63]:

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

100%|██████████| 73194/73194 [00:48<00:00, 1510.48it/s]

In [64]:

```
#text preprocessing on X_test datasets
X_test_preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(X_test['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    X_test_preprocessed_essays.append(sent.lower().strip())
```

100%|██████████| 36051/36051 [00:23<00:00, 1506.64it/s]

2.3.1.2 Vectorizing Text data:Bag of words

In [65]:

```
# We are considering only the words which appeared in at least 10 documents(rows or pro
jects).
vectorizer = CountVectorizer(min_df=10, max_features=5000)
vectorizer.fit(X_train_preprocessed_essays) # fit has to happen only on train data

X_train_text_bow = vectorizer.transform(X_train_preprocessed_essays)
X_test_text_bow = vectorizer.transform(X_test_preprocessed_essays)

print("After vectorizations")
print(X_train_text_bow.shape, y_train.shape)
print(X_test_text_bow.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(73194, 5000) (73194,)
(36051, 5000) (36051,)
====================================================================================
==========================
```

2.3.1.3 Vectorizing Text data: tfidf

In [66]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10, max_features=5000)
vectorizer.fit(X_train_preprocessed_essays) # fit has to happen only on train data
X_train_text_tfidf = vectorizer.transform(X_train_preprocessed_essays)
X_test_text_tfidf = vectorizer.transform(X_test_preprocessed_essays)

print("After vectorizations")
print(X_train_text_tfidf.shape, y_train.shape)
print(X_test_text_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(73194, 5000) (73194,)
(36051, 5000) (36051,)
====================================================================================================
```

2.3.1.4 Vectorizing Text data: avg w2v

In [67]:

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

In [68]:

```
# average Word2Vec for X_train
X_train_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train_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
    X_train_avg_w2v_vectors.append(vector)

print(len(X_train_avg_w2v_vectors))
print(len(X_train_avg_w2v_vectors[0]))
```

```
100%|██████████| 73194/73194 [00:26<00:00, 2781.89it/s]

73194
300
```

In [69]:

```
# average Word2Vec for X_test
X_test_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this l
ist
for sentence in tqdm(X_test_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
    X_test_avg_w2v_vectors.append(vector)

print(len(X_test_avg_w2v_vectors))
print(len(X_test_avg_w2v_vectors[0]))
```

```
100%|██████████| 36051/36051 [00:12<00:00, 2903.32it/s]

36051
300
```

2.3.1.5 Vectorizing Text data: tfidf weighted w2v

In [70]:

```
#tfidf w2v for X_train

# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit_transform(X_train_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())

X_train_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in thi
s list
for sentence in tqdm(X_train_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((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting 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
    X_train_tfidf_w2v_vectors.append(vector)

print(len(X_train_tfidf_w2v_vectors))
print(len(X_train_tfidf_w2v_vectors[0]))
```

```
100%|██████████| 73194/73194 [02:29<00:00, 489.90it/s]


73194
300
```

In [71]:

```
#tfidf w2v for X_test

# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model.transform(X_test_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())

X_test_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this
list
for sentence in tqdm(X_test_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((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting 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
    X_test_tfidf_w2v_vectors.append(vector)

print(len(X_test_tfidf_w2v_vectors))
print(len(X_test_tfidf_w2v_vectors[0]))
```

```
100%|██████████| 36051/36051 [01:11<00:00, 505.32it/s]

36051
300
```

2.3.2.1 Title preprocessing

In [72]:

```
#Title preprocessing on X_train datasets
from tqdm import tqdm
X_train_preprocessed_titles = []
# tqdm is for printing the status bar
for sentance in tqdm(X_train['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    X_train_preprocessed_titles.append(sent.lower().strip())
```

```
100%|██████████| 73194/73194 [00:02<00:00, 35561.82it/s]
```

In [73]:

```
#title preprocessing on X_test datasets
from tqdm import tqdm
X_test_preprocessed_titles = []
# tqdm is for printing the status bar
for sentance in tqdm(X_test['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    X_test_preprocessed_titles.append(sent.lower().strip())
```

```
100%|██████████| 36051/36051 [00:01<00:00, 35831.36it/s]
```

2.3.2.2 Vectorizing project_titles data:Bag of words

In [74]:

```
# We are considering only the words which appeared in at least 10 documents(rows or pro
jects).
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(X_train_preprocessed_titles) # fit has to happen only on train data

X_train_titles_bow = vectorizer.transform(X_train_preprocessed_titles)
X_test_titles_bow = vectorizer.transform(X_test_preprocessed_titles)

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

```
After vectorizations
(73194, 2534) (73194,)
(36051, 2534) (36051,)
====================================================================================
==========================
```

2.3.2.3 Vectorizing Text data: tfidf

In [75]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train_preprocessed_titles) # fit has to happen only on train data
X_train_titles_tfidf = vectorizer.transform(X_train_preprocessed_titles)
X_test_titles_tfidf = vectorizer.transform(X_test_preprocessed_titles)

print("After vectorizations")
print(X_train_titles_tfidf.shape, y_train.shape)
print(X_test_titles_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(73194, 2534) (73194,)
(36051, 2534) (36051,)
====================================================================================================
==========================
```

2.3.2.4 Vectorizing project_title data: avg w2v

In [76]:

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

In [77]:

```
# average Word2Vec for X_train
X_train_avg_w2v_vectors_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train_preprocessed_titles): # 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
    X_train_avg_w2v_vectors_titles.append(vector)

print(len(X_train_avg_w2v_vectors_titles))
print(len(X_train_avg_w2v_vectors_titles[0]))
```

```
100%|██████████| 73194/73194 [00:01<00:00, 60952.99it/s]

73194
300
```

In [78]:

```
# average Word2Vec for X_test
X_test_avg_w2v_vectors_titles = []; # the avg-w2v for each sentence/review is stored in
this list
for sentence in tqdm(X_test_preprocessed_titles): # 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
    X_test_avg_w2v_vectors_titles.append(vector)

print(len(X_test_avg_w2v_vectors_titles))
print(len(X_test_avg_w2v_vectors_titles[0]))
```

```
100%|██████████| 36051/36051 [00:00<00:00, 59209.10it/s]

36051
300
```

2.3.2.5 Vectorizing project_title data: tfidf weighted w2v

In [79]:

```
#tfidf w2v for X_train

# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit_transform(X_train_preprocessed_titles)
# 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())

X_train_tfidf_w2v_vectors_titles = []; # the avg-w2v for each sentence/review is stored
in this list
for sentence in tqdm(X_train_preprocessed_titles): # 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((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting 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
    X_train_tfidf_w2v_vectors_titles.append(vector)

print(len(X_train_tfidf_w2v_vectors_titles))
print(len(X_train_tfidf_w2v_vectors_titles[0]))
```

```
100%|██████████| 73194/73194 [00:02<00:00, 25918.21it/s]


73194
300
```

In [80]:

```
#tfidf w2v for X_test

# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model.transform(X_test_preprocessed_titles)
# 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())

X_test_tfidf_w2v_vectors_titles = []; # the avg-w2v for each sentence/review is stored
 in this list
for sentence in tqdm(X_test_preprocessed_titles): # 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((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting 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
    X_test_tfidf_w2v_vectors_titles.append(vector)

print(len(X_test_tfidf_w2v_vectors_titles))
print(len(X_test_tfidf_w2v_vectors_titles[0]))
```

```
100%|██████████| 36051/36051 [00:01<00:00, 27870.39it/s]

36051
300
```

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

In [81]:

```
X_train.head(1)
```

Out[81]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
96812	146651	p201363	2d5bbbe237a9466d143c35c0323e9b7b	Mr.	WA

1 rows × 29 columns

Concatinating all the features:model1-BOW

In [82]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr_m1 = hstack((X_train_price_norm, X_train_tnppp_norm, X_train_categories_0, X_train
_sub_categories_0, X_train_state_0,
                  X_train_teacher_0, X_train_grade_0, X_train_categories_1, X_train_sub
_categories_1, X_train_state_1,
                  X_train_teacher_1, X_train_grade_1, X_train_text_bow, X_train_titles_
bow))

X_te_m1 = hstack((X_test_price_norm, X_test_tnppp_norm, X_test_categories_0, X_test_sub
_categories_0, X_test_state_0,
                  X_test_teacher_0, X_test_grade_0, X_test_categories_1, X_test_sub_cat
egories_1, X_test_state_1,
                  X_test_teacher_1, X_test_grade_1, X_test_text_bow, X_test_titles_bow
))

print("Final Data matrix")
print(X_tr_m1.shape, y_train.shape)
print(X_te_m1.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(73194, 7546) (73194,)
(36051, 7546) (36051,)
====================================================================================
==========================
```

Concatinating all the features:model2-TFIDF

In [83]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039

X_tr_m2 = hstack((X_train_price_norm, X_train_tnppp_norm, X_train_categories_0, X_train
_sub_categories_0, X_train_state_0,
                  X_train_teacher_0, X_train_grade_0, X_train_categories_1, X_train_sub
_categories_1, X_train_state_1,
                  X_train_teacher_1, X_train_grade_1, X_train_text_tfidf, X_train_title
s_tfidf))

X_te_m2 = hstack((X_test_price_norm, X_test_tnppp_norm, X_test_categories_0, X_test_sub
_categories_0, X_test_state_0,
                  X_test_teacher_0, X_test_grade_0, X_test_categories_1, X_test_sub_cat
egories_1, X_test_state_1,
                  X_test_teacher_1, X_test_grade_1, X_test_text_tfidf, X_test_titles_tf
idf))

print("Final Data matrix")
print(X_tr_m2.shape, y_train.shape)
print(X_te_m2.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(73194, 7546) (73194,)
(36051, 7546) (36051,)
====================================================================================
==========================
```

Concatinating all the features:model3-AVG W2V

In [84]:

```
X_tr_m3 = np.hstack((X_train_price_norm, X_train_tnppp_norm, X_train_categories_0, X_tr
ain_sub_categories_0, X_train_teacher_0,
                     X_train_state_0, X_train_grade_0, X_train_categories_1, X_train_su
b_categories_1, X_train_teacher_1,
                     X_train_state_1, X_train_grade_1, X_train_avg_w2v_vectors, X_train
_avg_w2v_vectors_titles))

X_te_m3 = np.hstack((X_test_price_norm, X_test_tnppp_norm, X_test_categories_0, X_test_
sub_categories_0, X_test_teacher_0,
                     X_test_state_0, X_test_grade_0, X_test_categories_1, X_test_sub_ca
tegories_1, X_test_teacher_1,
                     X_test_state_1, X_test_grade_1, X_test_avg_w2v_vectors, X_test_avg
_w2v_vectors_titles))

print("Final Data matrix")
print(X_tr_m3.shape, y_train.shape)
print(X_te_m3.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(73194, 612) (73194,)
(36051, 612) (36051,)
====================================================================================
==========================
```

Concatinating all the features:model4-TFIDF WEIGHTED W2V

In [85]:

```
X_tr_m4 = np.hstack((X_train_categories_0, X_train_sub_categories_0, X_train_teacher_0,
X_train_state_0, X_train_grade_0,
                     X_train_categories_1, X_train_sub_categories_1, X_train_teacher_1,
X_train_state_1, X_train_grade_1,
                     X_train_price_norm, X_train_tnppp_norm, X_train_tfidf_w2v_vectors,
X_train_tfidf_w2v_vectors_titles))

X_te_m4 = np.hstack((X_test_categories_0, X_test_sub_categories_0, X_test_teacher_0, X_
test_state_0, X_test_grade_0,
                     X_test_categories_1, X_test_sub_categories_1, X_test_teacher_1, X_
test_state_1, X_test_grade_1,
                     X_test_price_norm, X_test_tnppp_norm, X_test_tfidf_w2v_vectors, X_
test_tfidf_w2v_vectors_titles))

print("Final Data matrix")
print(X_tr_m4.shape, y_train.shape)
print(X_te_m4.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(73194, 612) (73194,)
(36051, 612) (36051,)
====================================================================================
==========================
```

2.4.1 Applying Random Forests on BOW, SET 1

In [86]:

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

X_tr_m1 = csr_matrix(X_tr_m1)
X_te_m1 = csr_matrix(X_te_m1)
```

In [114]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchC
V.html
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

neigh = RandomForestClassifier(n_jobs=-1, class_weight='balanced')
parameters = {'n_estimators': [5, 10, 50, 100], 'max_depth': [2, 5, 10, 20]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc', return_train_score=True)
clf.fit(X_tr_m1, y_train)

train_auc= clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']
```

In [115]:

```
train_auc
```

Out[115]:

```
array([0.5810404 , 0.61553172, 0.66619344, 0.67117654, 0.61866341,
       0.65405803, 0.69519882, 0.71609093, 0.68780431, 0.7312237 ,
       0.78007172, 0.80147544, 0.8196731 , 0.8719692 , 0.93601164,
       0.94633538])
```

In [116]:

```
#making dataframe for train_auc values with max_depth & min_sample_split:
d = {'max_depth': [2, 5, 10, 20],
    'n_estimators = 5': [0.5810404, 0.61553172, 0.66619344, 0.67117654],
    'n_estimators = 10': [0.61866341, 0.65405803, 0.69519882, 0.71609093],
    'n_estimators = 50': [0.68780431, 0.7312237, 0.78007172, 0.80147544],
    'n_estimators = 100': [0.8196731 , 0.8719692, 0.93601164, 0.94633538]}
df = pd.DataFrame(d).set_index('max_depth')
df
```

Out[116]:

max_depth	n_estimators = 10	n_estimators = 100	n_estimators = 5	n_estimators = 50
2	0.618663	0.819673	0.581040	0.687804
5	0.654058	0.871969	0.615532	0.731224
10	0.695199	0.936012	0.666193	0.780072
20	0.716091	0.946335	0.671177	0.801475

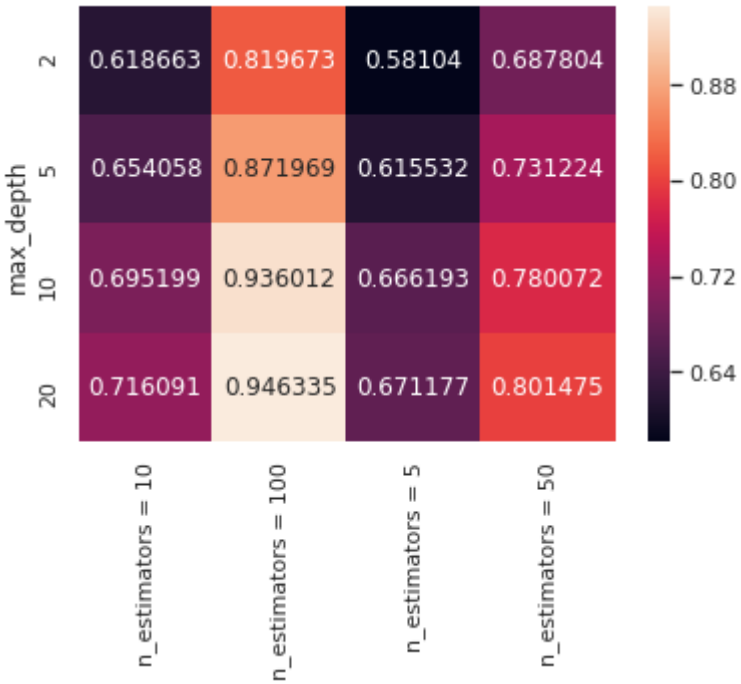
In [117]:

```
#heatmap for train_auc in each cases:
import seaborn as sns; sns.set()

sns.set(font_scale = 1.0)
sns.heatmap(df, annot=True, fmt='g')
```

Out[117]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd165ef4470>
```



In [118]:

```
cv_auc
```

Out[118]:

```
array([0.57347039, 0.6064519 , 0.64999205, 0.65247507, 0.59797158,
       0.62492206, 0.65570409, 0.66884503, 0.61430192, 0.64081933,
       0.67243004, 0.68333212, 0.61805836, 0.64390891, 0.68546027,
       0.68953662])
```

In [119]:

```
#cv_auc values in each cases:
d = {'max_depth': [2, 5, 10, 20],
     'n_estimators = 5': [0.57347039, 0.6064519, 0.64999205, 0.65247507],
     'n_estimators = 10': [0.59797158, 0.62492206, 0.65570409, 0.66884503],
     'n_estimators = 50': [0.61430192, 0.64081933, 0.67243004, 0.68333212],
     'n_estimators = 100': [0.61805836, 0.64390891, 0.68546027, 0.68953662]}
df = pd.DataFrame(d).set_index('max_depth')
df
```

Out[119]:

max_depth	n_estimators = 10	n_estimators = 100	n_estimators = 5	n_estimators = 50
2	0.597972	0.618058	0.573470	0.614302
5	0.624922	0.643909	0.606452	0.640819
10	0.655704	0.685460	0.649992	0.672430
20	0.668845	0.689537	0.652475	0.683332

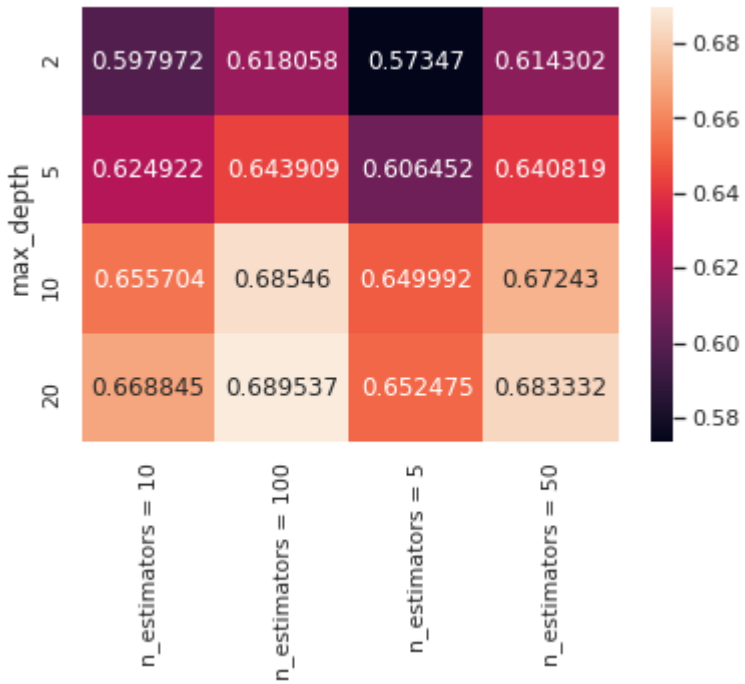
In [120]:

```
#heatmap for cv_auc in each cases:
import seaborn as sns; sns.set()

sns.set(font_scale = 1.0)
sns.heatmap(df, annot=True, fmt='g')
```

Out[120]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd16b278fd0>
```



In [121]:

```
#1.from the heatmap plot we choose max_depth and  n_estimators such that we will have m
aximum AUC on cv data.
#2.Gap between cv_auc and train_auc should be less.

best_max_depth = 20
best_n_estimators = 10
```

In [123]:

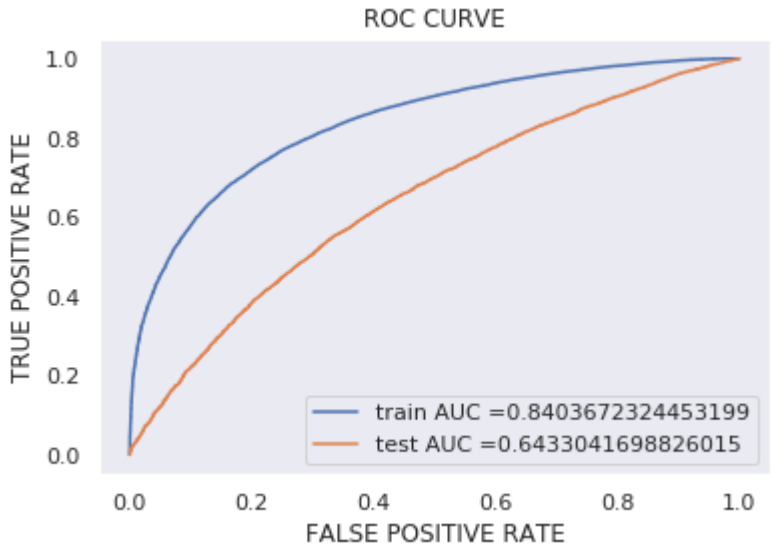
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#skle
arn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

neigh = RandomForestClassifier(n_estimators = 10, max_depth = 20, n_jobs = -1, class_we
ight = 'balanced')
neigh.fit(X_tr_m1, y_train)

y_train_pred = neigh.predict_proba(X_tr_m1)[:,1]
y_test_pred = neigh.predict_proba(X_te_m1)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE")
plt.ylabel("TRUE POSITIVE RATE")
plt.title("ROC CURVE")
plt.grid()
plt.show()
```



In [124]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, 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.rou
nd(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

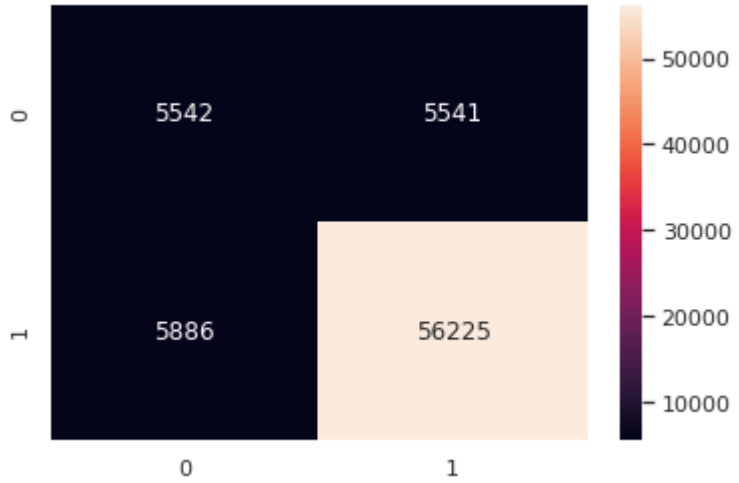
In [125]:

```
import seaborn as sns; sns.set()
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
conf_mat_data = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fp
r, train_fpr))
sns.set(font_scale = 1.0)
sns.heatmap(conf_mat_data, annot=True, fmt='g')
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2499999979647145 for threshold 0.455
```

Out[125]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd165b7fe10>
```



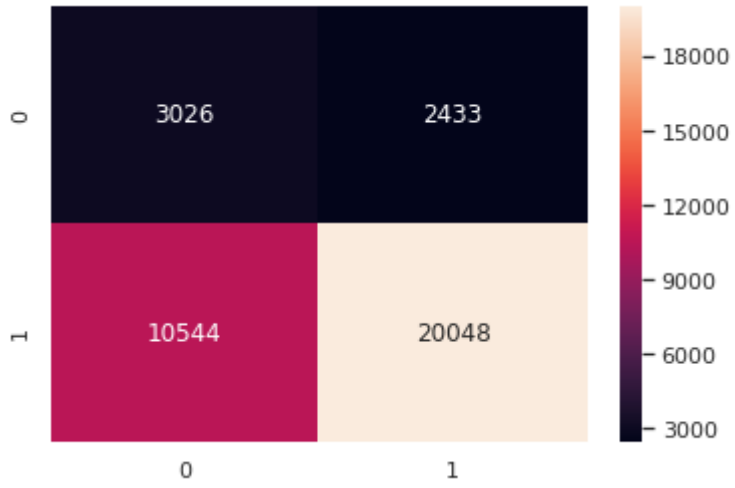
In [126]:

```
print("Test confusion matrix")
sns.set(font_scale = 1.0)
conf_mat_data = confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr,
test_fpr))
sns.heatmap(conf_mat_data, annot=True, fmt='g')
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.514
```

Out[126]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd165eede80>
```



2.4.2 Applying Random Forests on TFIDF, SET 2

In [112]:

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

X_tr_m2 = csr_matrix(X_tr_m2)
X_te_m2 = csr_matrix(X_te_m2)
```

In [127]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchC
V.html
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

neigh = RandomForestClassifier(n_jobs=-1, class_weight='balanced')
parameters = {'n_estimators': [5, 10, 50, 100], 'max_depth': [2, 5, 10, 20]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc', return_train_score=True)
clf.fit(X_tr_m2, y_train)

train_auc= clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']
```

In [128]:

```
train_auc
```

Out[128]:

```
array([0.56552584, 0.61166084, 0.67461508, 0.68532553, 0.62972194,
       0.66946222, 0.72012947, 0.7296182 , 0.70909883, 0.75498831,
       0.81435292, 0.82427403, 0.84406269, 0.90157177, 0.96636164,
       0.97482214])
```

In [129]:

```
#making dataframe for train_auc values with max_depth & min_sample_split:
d = {'max_depth': [2, 5, 10, 20],
    'n_estimators = 5': [0.56552584, 0.61166084, 0.67461508, 0.68532553],
    'n_estimators = 10': [0.62972194, 0.66946222, 0.72012947, 0.7296182],
    'n_estimators = 50': [0.70909883, 0.75498831, 0.81435292, 0.82427403],
    'n_estimators = 100': [0.84406269, 0.90157177, 0.96636164, 0.97482214]}
df = pd.DataFrame(d).set_index('max_depth')
df
```

Out[129]:

max_depth	n_estimators = 10	n_estimators = 100	n_estimators = 5	n_estimators = 50
2	0.629722	0.844063	0.565526	0.709099
5	0.669462	0.901572	0.611661	0.754988
10	0.720129	0.966362	0.674615	0.814353
20	0.729618	0.974822	0.685326	0.824274

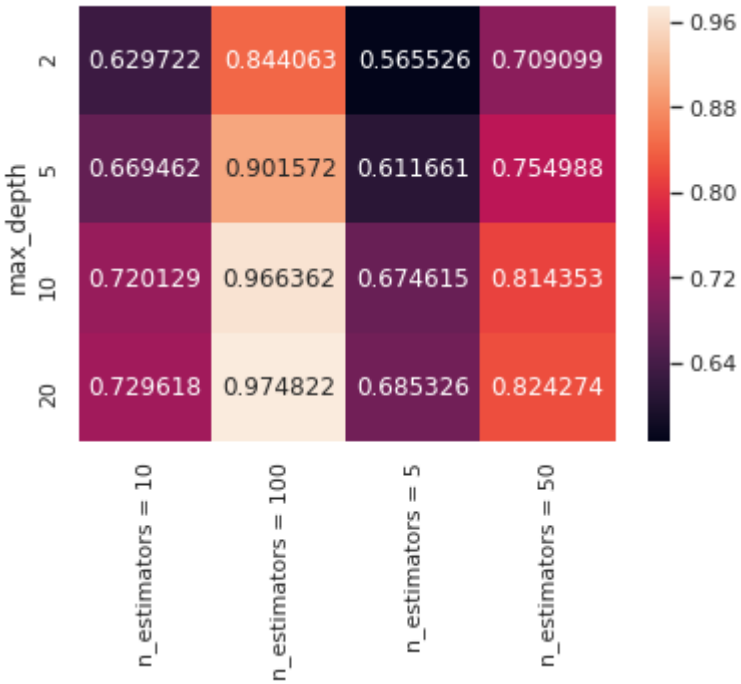
In [130]:

```
#heatmap for train_auc in each cases:
import seaborn as sns; sns.set()

sns.set(font_scale = 1.0)
sns.heatmap(df, annot=True, fmt='g')
```

Out[130]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd165c87ef0>
```



In [131]:

```
cv_auc
```

Out[131]:

```
array([0.55487828, 0.5927961 , 0.65088579, 0.66301216, 0.59762241,
       0.62937126, 0.66505099, 0.67432231, 0.61326776, 0.63041085,
       0.6753445 , 0.68191328, 0.60614464, 0.62318654, 0.67190857,
       0.68397336])
```

In [132]:

```
#cv_auc values in each cases:
d = {'max_depth': [2, 5, 10, 20],
     'n_estimators = 5': [0.55487828, 0.5927961, 0.65088579, 0.66301216],
     'n_estimators = 10': [0.59762241, 0.62937126, 0.66505099, 0.67432231],
     'n_estimators = 50': [0.61326776, 0.63041085, 0.6753445, 0.68191328],
     'n_estimators = 100': [0.60614464, 0.62318654, 0.67190857, 0.68397336]}
df = pd.DataFrame(d).set_index('max_depth')
df
```

Out[132]:

max_depth	n_estimators = 10	n_estimators = 100	n_estimators = 5	n_estimators = 50
2	0.597622	0.606145	0.554878	0.613268
5	0.629371	0.623187	0.592796	0.630411
10	0.665051	0.671909	0.650886	0.675345
20	0.674322	0.683973	0.663012	0.681913

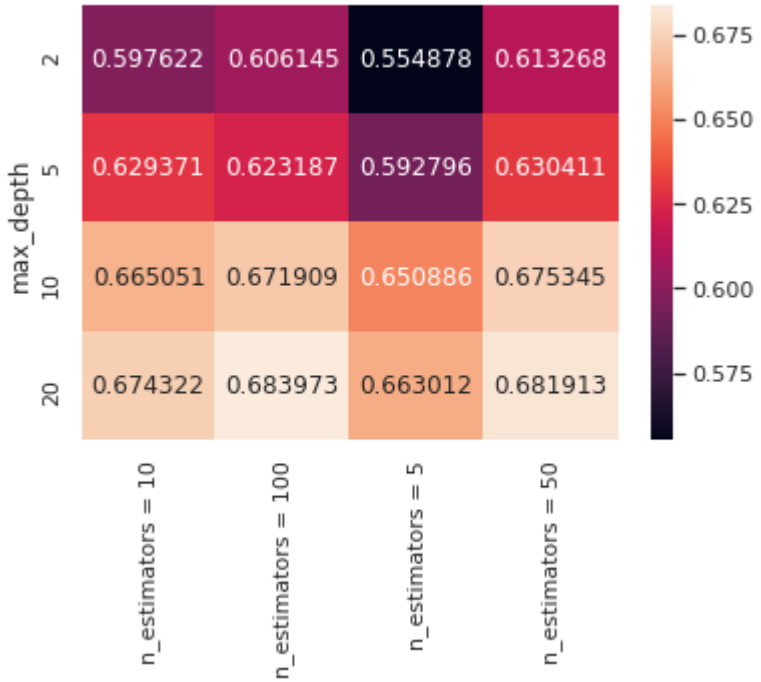
In [133]:

```
#heatmap for cv_auc in each cases:
import seaborn as sns; sns.set()

sns.set(font_scale = 1.0)
sns.heatmap(df, annot=True, fmt='g')
```

Out[133]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd16b723ef0>
```



In [134]:

```
#1.from the heatmap plot we choose max_depth and  n_estimators such that we will have m
aximum AUC on cv data.
#2.Gap between cv_auc and train_auc should be less.

best_max_depth = 20
best_n_estimators = 10
```

In [135]:

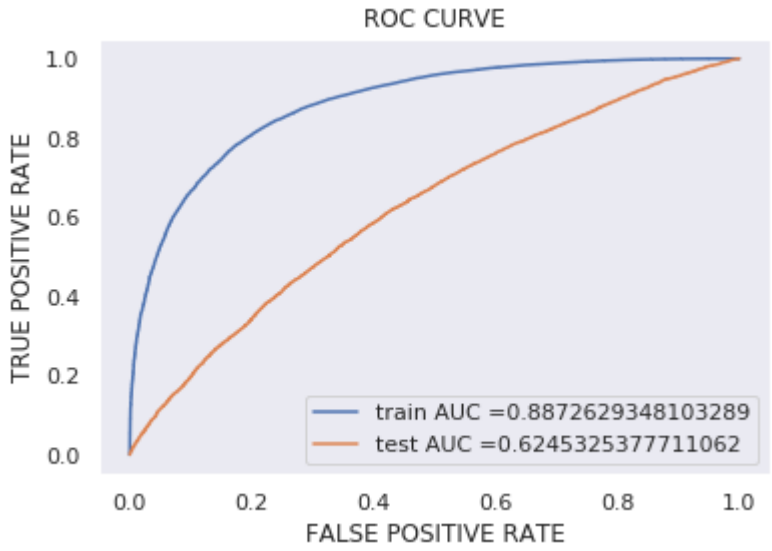
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#skle
arn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

neigh = RandomForestClassifier(n_estimators = 10, max_depth = 20, n_jobs = -1, class_we
ight = 'balanced')
neigh.fit(X_tr_m2, y_train)

y_train_pred = neigh.predict_proba(X_tr_m2)[:,1]
y_test_pred = neigh.predict_proba(X_te_m2)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE")
plt.ylabel("TRUE POSITIVE RATE")
plt.title("ROC CURVE")
plt.grid()
plt.show()
```



In [136]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, 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.rou
nd(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

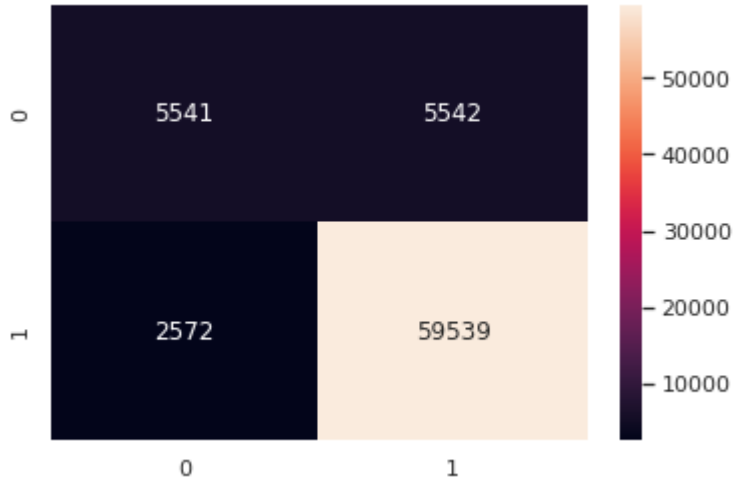
In [137]:

```
import seaborn as sns; sns.set()
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
conf_mat_data = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fp
r, train_fpr))
sns.set(font_scale = 1.0)
sns.heatmap(conf_mat_data, annot=True, fmt='g')
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999796471448 for threshold 0.454
```

Out[137]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd165af7128>
```



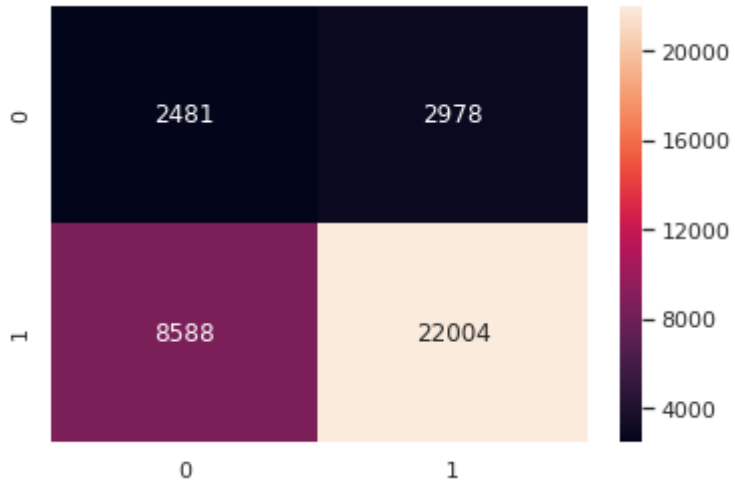
In [138]:

```
print("Test confusion matrix")
sns.set(font_scale = 1.0)
conf_mat_data = confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr,
test_fpr))
sns.heatmap(conf_mat_data, annot=True, fmt='g')
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.522
```

Out[138]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd16b52c8d0>
```



2.4.3 Applying Random Forests on AVG W2V, SET 3

In [139]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchC
V.html
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

neigh = RandomForestClassifier(n_jobs=-1, class_weight='balanced')
parameters = {'n_estimators': [5, 10, 50, 100], 'max_depth': [2, 5, 10, 20]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc', return_train_score=True)
clf.fit(X_tr_m3, y_train)

train_auc= clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']
```

In [140]:

```
train_auc
```

Out[140]:

```
array([0.62078913, 0.63996746, 0.66437374, 0.66596438, 0.67755175,
       0.69847807, 0.72181783, 0.72500706, 0.85300449, 0.90567649,
       0.95635103, 0.96437899, 0.98635591, 0.99869345, 0.99998591,
       0.99999469])
```

In [141]:

```
#making dataframe for train_auc values with max_depth & min_sample_split:
d = {'max_depth': [2, 5, 10, 20],
     'n_estimators = 5': [0.62078913, 0.63996746, 0.66437374, 0.66596438],
     'n_estimators = 10': [0.67755175, 0.69847807, 0.72181783, 0.72500706],
     'n_estimators = 50': [0.85300449, 0.90567649, 0.95635103, 0.96437899],
     'n_estimators = 100': [0.98635591, 0.99869345, 0.99998591, 0.99999469]}
df = pd.DataFrame(d).set_index('max_depth')
df
```

Out[141]:

max_depth	n_estimators = 10	n_estimators = 100	n_estimators = 5	n_estimators = 50
2	0.677552	0.986356	0.620789	0.853004
5	0.698478	0.998693	0.639967	0.905676
10	0.721818	0.999986	0.664374	0.956351
20	0.725007	0.999995	0.665964	0.964379

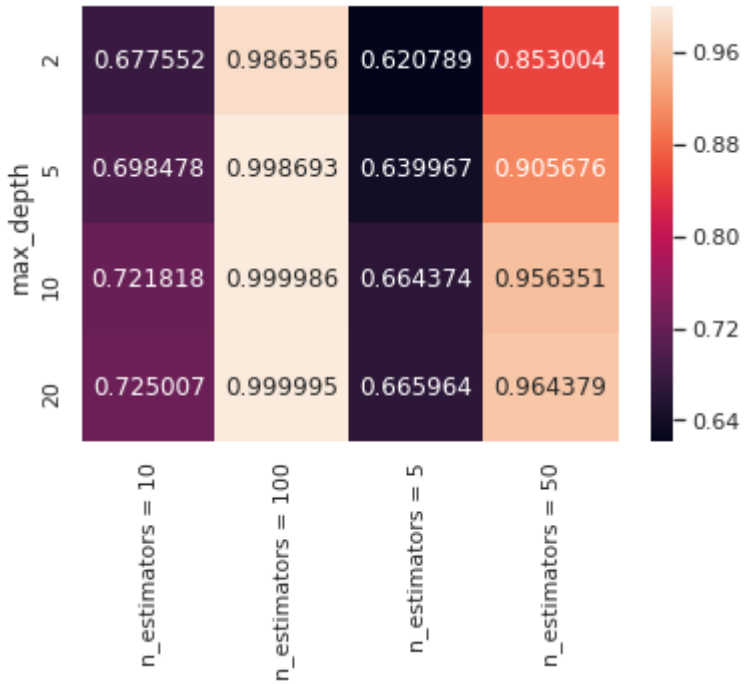
In [142]:

```
#heatmap for train_auc in each cases:
import seaborn as sns; sns.set()

sns.set(font_scale = 1.0)
sns.heatmap(df, annot=True, fmt='g')
```

Out[142]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd165bb3320>
```



In [143]:

```
cv_auc
```

Out[143]:

```
array([0.61053516, 0.62703417, 0.65051599, 0.65344313, 0.63221068,
       0.64990006, 0.66474851, 0.66828139, 0.60440726, 0.62553745,
       0.66486826, 0.66965381, 0.57341922, 0.59026108, 0.64960345,
       0.65962125])
```

In [144]:

```
#cv_auc values in each cases:
d = {'max_depth': [2, 5, 10, 20],
    'n_estimators = 5': [0.61053516, 0.62703417, 0.65051599, 0.65344313],
    'n_estimators = 10': [0.63221068, 0.64990006, 0.66474851, 0.66828139],
    'n_estimators = 50': [0.60440726, 0.62553745, 0.66486826, 0.66965381],
    'n_estimators = 100': [0.57341922, 0.59026108, 0.64960345, 0.65962125]}
df = pd.DataFrame(d).set_index('max_depth')
df
```

Out[144]:

max_depth	n_estimators = 10	n_estimators = 100	n_estimators = 5	n_estimators = 50
2	0.632211	0.573419	0.610535	0.604407
5	0.649900	0.590261	0.627034	0.625537
10	0.664749	0.649603	0.650516	0.664868
20	0.668281	0.659621	0.653443	0.669654

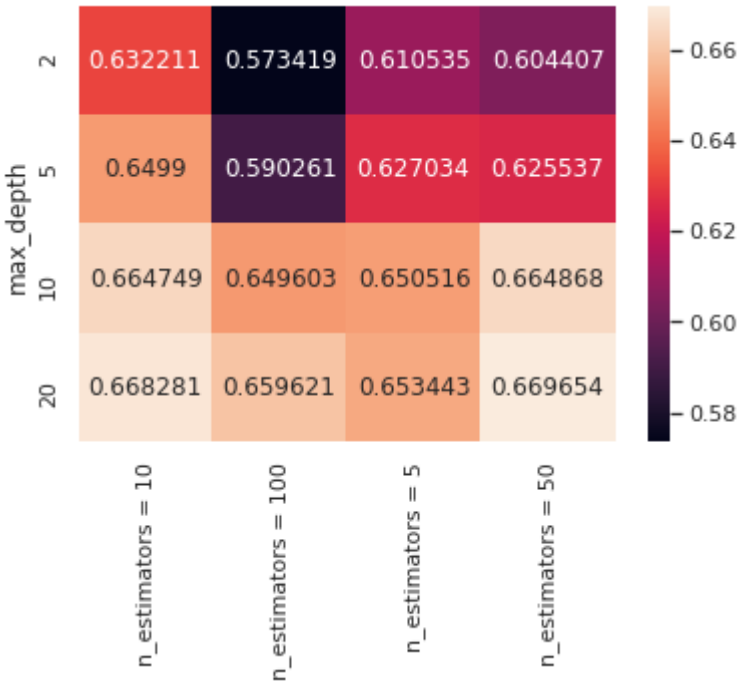
In [145]:

```
#heatmap for cv_auc in each cases:
import seaborn as sns; sns.set()

sns.set(font_scale = 1.0)
sns.heatmap(df, annot=True, fmt='g')
```

Out[145]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd16b2055f8>
```



In [146]:

```
#1.from the heatmap plot we choose max_depth and  n_estimators such that we will have m
aximum AUC on cv data.
#2.Gap between cv_auc and train_auc should be less.

best_max_depth = 10
best_n_estimators = 10
```

In [148]:

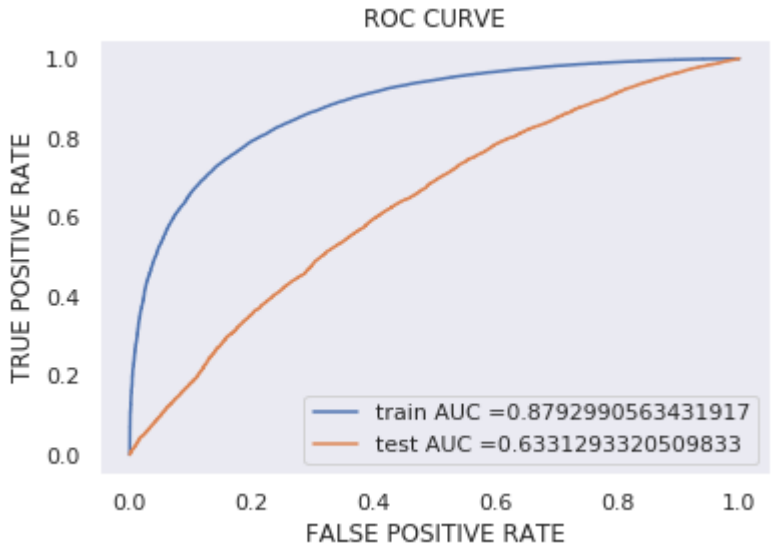
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

neigh = RandomForestClassifier(n_estimators = 10, max_depth = 10, n_jobs = -1, class_weight = 'balanced')
neigh.fit(X_tr_m3, y_train)

y_train_pred = neigh.predict_proba(X_tr_m3)[:,1]
y_test_pred = neigh.predict_proba(X_te_m3)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE")
plt.ylabel("TRUE POSITIVE RATE")
plt.title("ROC CURVE")
plt.grid()
plt.show()
```



In [149]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, 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.rou
nd(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

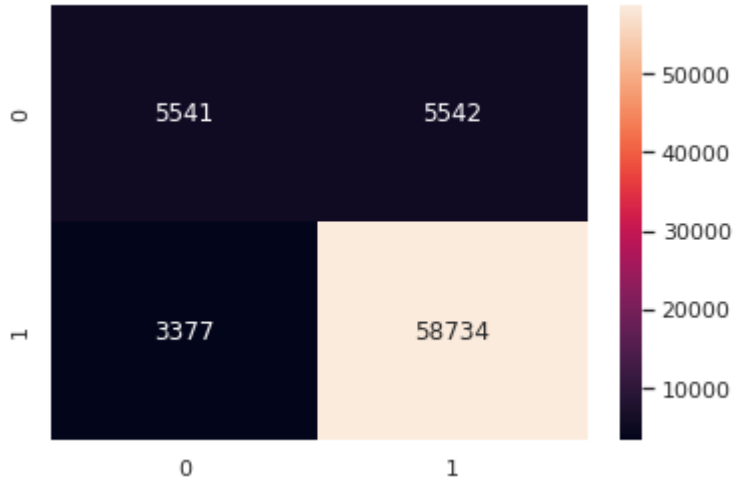
In [150]:

```
import seaborn as sns; sns.set()
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
conf_mat_data = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fp
r, train_fpr))
sns.set(font_scale = 1.0)
sns.heatmap(conf_mat_data, annot=True, fmt='g')
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999796471448 for threshold 0.439
```

Out[150]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd16b240a58>
```



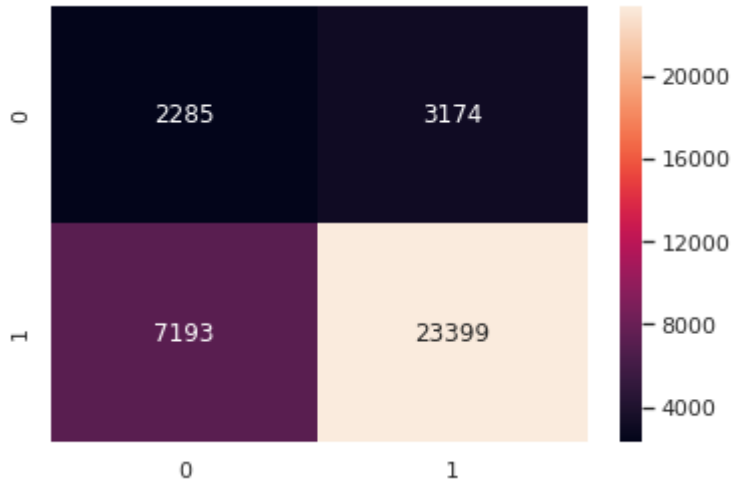
In [151]:

```
print("Test confusion matrix")
sns.set(font_scale = 1.0)
conf_mat_data = confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr,
test_fpr))
sns.heatmap(conf_mat_data, annot=True, fmt='g')
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.52
```

Out[151]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd16b836ba8>
```



2.4.4 Applying Random Forests on TFIDF W2V, SET 4

In [153]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchC
V.html
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

neigh = RandomForestClassifier(n_jobs=-1, class_weight='balanced')
parameters = {'n_estimators': [5, 10, 50, 100], 'max_depth': [2, 5, 10, 20]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc', return_train_score=True)
clf.fit(X_tr_m4, y_train)

train_auc= clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']
```

In [154]:

```
train_auc
```

Out[154]:

```
array([0.62495297, 0.64534517, 0.6666345 , 0.66835791, 0.68035245,
       0.70204036, 0.72115144, 0.72380369, 0.84732456, 0.8977544 ,
       0.94503284, 0.95137505, 0.98531218, 0.99796287, 0.99998809,
       0.99998928])
```

In [155]:

```
#making dataframe for train_auc values with max_depth & min_sample_split:
d = {'max_depth': [2, 5, 10, 20],
     'n_estimators = 5': [0.62495297, 0.64534517, 0.6666345, 0.66835791],
     'n_estimators = 10': [0.68035245, 00.70204036, 0.72115144, 0.72380369],
     'n_estimators = 50': [0.84732456, 0.8977544, 0.94503284, 0.95137505],
     'n_estimators = 100': [0.98531218, 0.99796287, 0.99998809, 0.99998928]}
df = pd.DataFrame(d).set_index('max_depth')
df
```

Out[155]:

max_depth	n_estimators = 10	n_estimators = 100	n_estimators = 5	n_estimators = 50
2	0.680352	0.985312	0.624953	0.847325
5	0.702040	0.997963	0.645345	0.897754
10	0.721151	0.999988	0.666635	0.945033
20	0.723804	0.999989	0.668358	0.951375

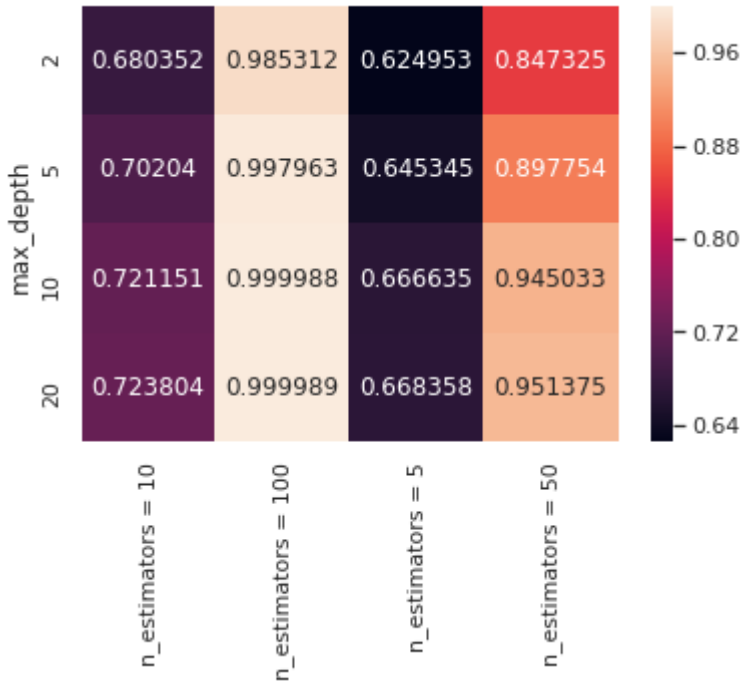
In [156]:

```
#heatmap for train_auc in each cases:
import seaborn as sns; sns.set()

sns.set(font_scale = 1.0)
sns.heatmap(df, annot=True, fmt='g')
```

Out[156]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd16bb90dd8>
```



In [157]:

```
cv_auc
```

Out[157]:

```
array([0.61351685, 0.63478261, 0.65430005, 0.6559618 , 0.63986681,
       0.65485001, 0.67088913, 0.67225698, 0.61858843, 0.63955709,
       0.67170296, 0.67538016, 0.57826159, 0.60335016, 0.65471912,
       0.66669321])
```

In [158]:

```
#cv_auc values in each cases:
d = {'max_depth': [2, 5, 10, 20],
    'n_estimators = 5': [0.61351685, 0.63478261, 0.65430005, 0.6559618],
    'n_estimators = 10': [0.63986681, 0.65485001, 0.67088913, 0.67225698],
    'n_estimators = 50': [0.61858843, 0.63955709, 0.67170296, 0.67538016],
    'n_estimators = 100': [0.57826159, 0.60335016, 0.65471912, 0.66669321]}
df = pd.DataFrame(d).set_index('max_depth')
df
```

Out[158]:

max_depth	n_estimators = 10	n_estimators = 100	n_estimators = 5	n_estimators = 50
2	0.639867	0.578262	0.613517	0.618588
5	0.654850	0.603350	0.634783	0.639557
10	0.670889	0.654719	0.654300	0.671703
20	0.672257	0.666693	0.655962	0.675380

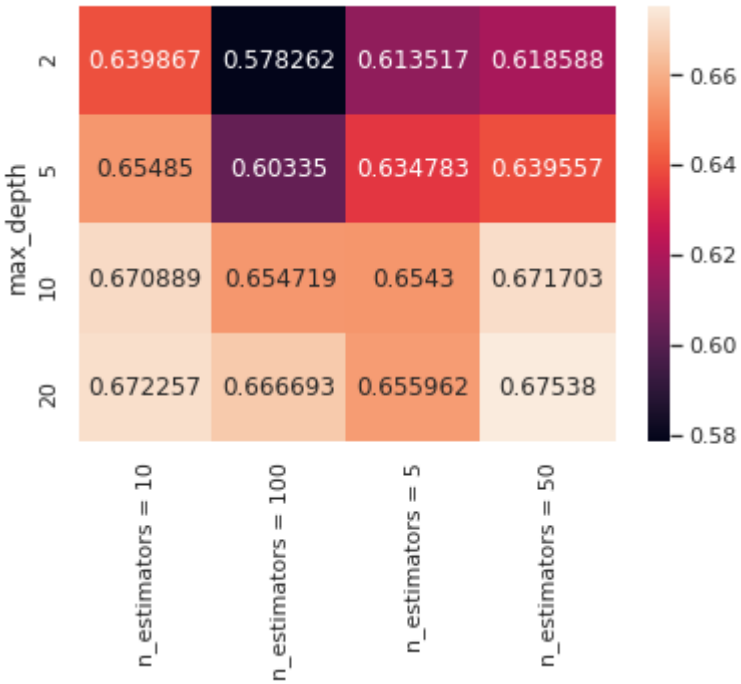
In [159]:

```
#heatmap for cv_auc in each cases:
import seaborn as sns; sns.set()

sns.set(font_scale = 1.0)
sns.heatmap(df, annot=True, fmt='g')
```

Out[159]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd165d819e8>
```



In [160]:

```
#1.from the heatmap plot we choose max_depth and  n_estimators such that we will have m
aximum AUC on cv data.
#2.Gap between cv_auc and train_auc should be less.

best_max_depth = 10
best_n_estimators = 10
```

In [162]:

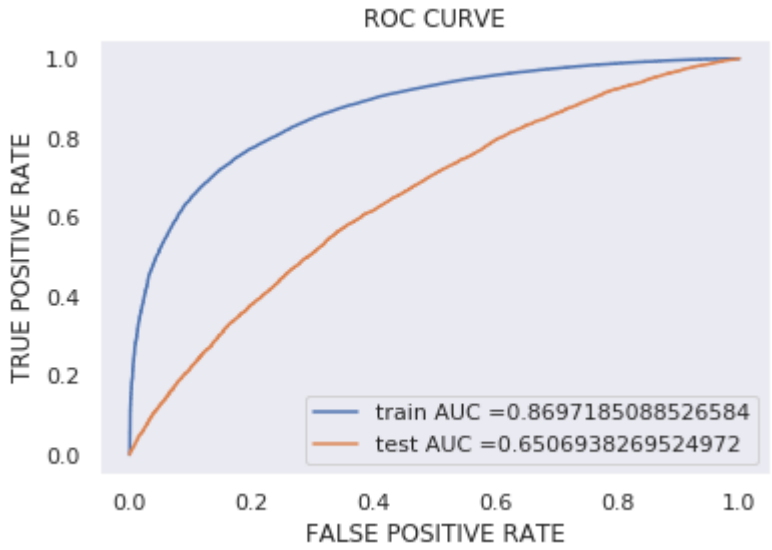
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

neigh = RandomForestClassifier(n_estimators = 10, max_depth = 10, n_jobs = -1, class_weight = 'balanced')
neigh.fit(X_tr_m4, y_train)

y_train_pred = neigh.predict_proba(X_tr_m4)[:,1]
y_test_pred = neigh.predict_proba(X_te_m4)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE")
plt.ylabel("TRUE POSITIVE RATE")
plt.title("ROC CURVE")
plt.grid()
plt.show()
```



In [163]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, 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.rou
nd(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

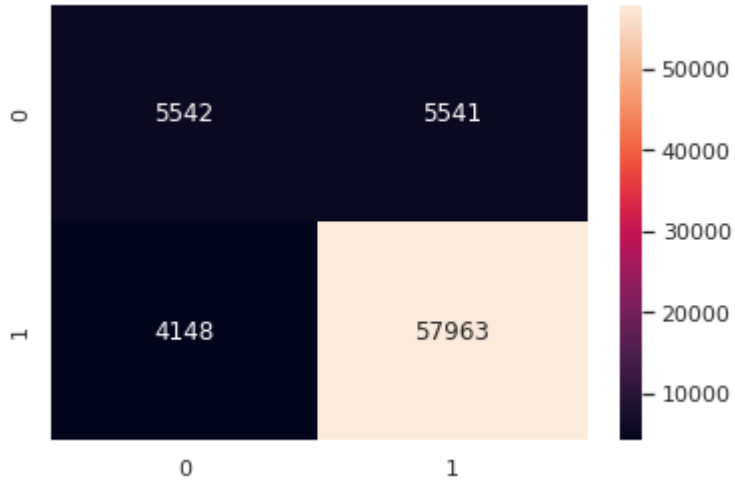
In [164]:

```
import seaborn as sns; sns.set()
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
conf_mat_data = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fp
r, train_fpr))
sns.set(font_scale = 1.0)
sns.heatmap(conf_mat_data, annot=True, fmt='g')
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2499999979647145 for threshold 0.428
```

Out[164]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd16b64a390>
```



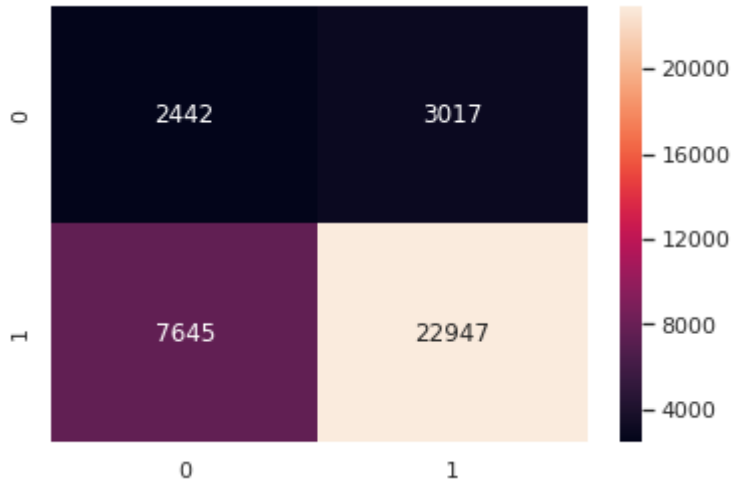
In [165]:

```
print("Test confusion matrix")
sns.set(font_scale = 1.0)
conf_mat_data = confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr,
test_fpr))
sns.heatmap(conf_mat_data, annot=True, fmt='g')
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.509
```

Out[165]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd165e116a0>
```



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

2.5.1 Applying XGBOOST on BOW, SET 1

In [86]:

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

X_tr_m1 = csr_matrix(X_tr_m1)
X_te_m1 = csr_matrix(X_te_m1)
```

In [88]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
import xgboost as xgb

neigh = xgb.XGBClassifier(n_jobs=-1)
parameters = {'max_depth': [2, 4, 6, 8], 'n_estimators': [5, 10, 50, 100]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc', return_train_score=True)
clf.fit(X_tr_m1, y_train)

train_auc= clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']
```

In [89]:

```
train_auc
```

Out[89]:

```
array([0.59307484, 0.61180976, 0.66877343, 0.69872358, 0.63623711,
       0.6526147 , 0.72503972, 0.76906201, 0.67279492, 0.6963472 ,
       0.79358579, 0.84731312, 0.71065896, 0.74756383, 0.86530365,
       0.91497735])
```

In [91]:

```
#making dataframe for train_auc values with max_depth & min_sample_split:
d = {'n_estimators': [5, 10, 50, 100],
    'max_depth = 2': [0.59307484, 0.61180976, 0.66877343, 0.69872358],
    'max_depth = 4': [0.63623711, 0.6526147 , 0.72503972, 0.76906201],
    'max_depth = 6': [0.67279492, 0.6963472, 0.79358579, 0.84731312],
    'max_depth = 8': [0.71065896, 0.74756383, 0.86530365, 0.91497735]}
df = pd.DataFrame(d).set_index('n_estimators')
df
```

Out[91]:

n_estimators	max_depth = 2	max_depth = 4	max_depth = 6	max_depth = 8
5	0.593075	0.636237	0.672795	0.710659
10	0.611810	0.652615	0.696347	0.747564
50	0.668773	0.725040	0.793586	0.865304
100	0.698724	0.769062	0.847313	0.914977

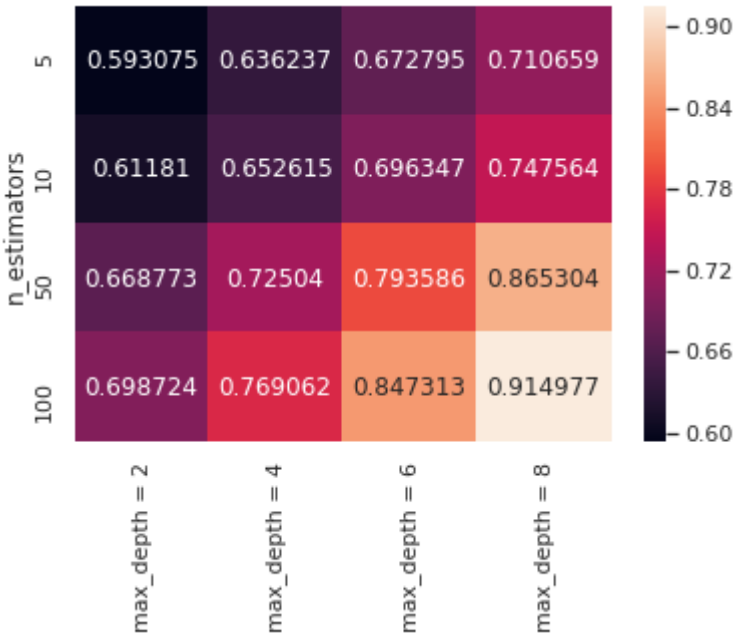
In [92]:

```
#heatmap for train_auc in each cases:
import seaborn as sns; sns.set()

sns.set(font_scale = 1.0)
sns.heatmap(df, annot=True, fmt='g')
```

Out[92]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe0400931d0>
```



In [93]:

```
cv_auc
```

Out[93]:

```
array([0.58790668, 0.60720974, 0.65661936, 0.67899818, 0.62054062,
       0.63218887, 0.67904531, 0.69950358, 0.63417177, 0.64205152,
       0.68810446, 0.70460002, 0.63681729, 0.64772985, 0.69332023,
       0.70837328])
```

In [94]:

```
#cv_auc values in each cases:
d = {'n_estimators': [5, 10, 50, 100],
     'max_depth = 2': [0.58790668, 0.60720974, 0.65661936, 0.67899818],
     'max_depth = 4': [0.62054062, 0.63218887, 0.67904531, 0.69950358],
     'max_depth = 6': [0.63417177, 0.64205152, 0.68810446, 0.70460002],
     'max_depth = 8': [0.63681729, 0.64772985, 0.69332023, 0.70837328]}
df = pd.DataFrame(d).set_index('n_estimators')
df
```

Out[94]:

n_estimators	max_depth = 2	max_depth = 4	max_depth = 6	max_depth = 8
5	0.587907	0.620541	0.634172	0.636817
10	0.607210	0.632189	0.642052	0.647730
50	0.656619	0.679045	0.688104	0.693320
100	0.678998	0.699504	0.704600	0.708373

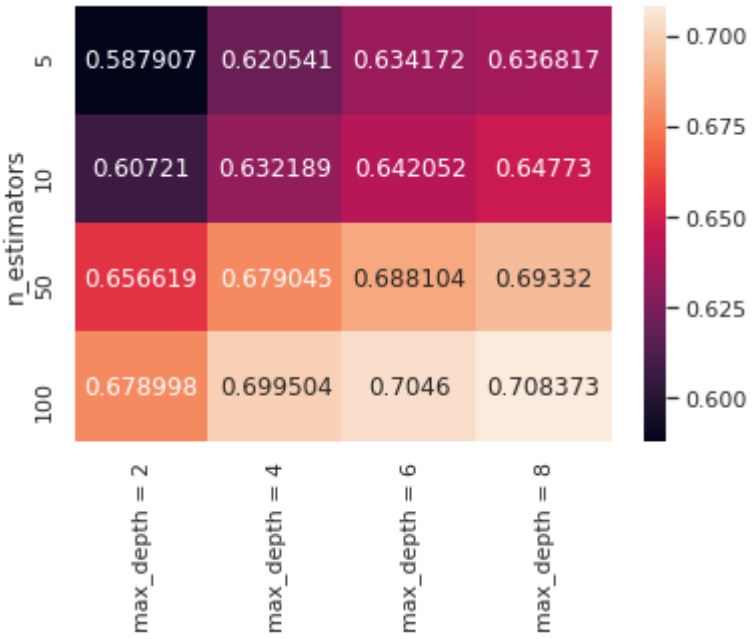
In [95]:

```
#heatmap for cv_auc in each cases:
import seaborn as sns; sns.set()

sns.set(font_scale = 1.0)
sns.heatmap(df, annot=True, fmt='g')
```

Out[95]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe0456b9c50>
```



In [134]:

```
#1.from the heatmap plot we choose max_depth and  n_estimators such that we will have m
aximum AUC on cv data.
#2.Gap between cv_auc and train_auc should be less.

best_max_depth = 4
best_n_estimators = 100
```

In [96]:

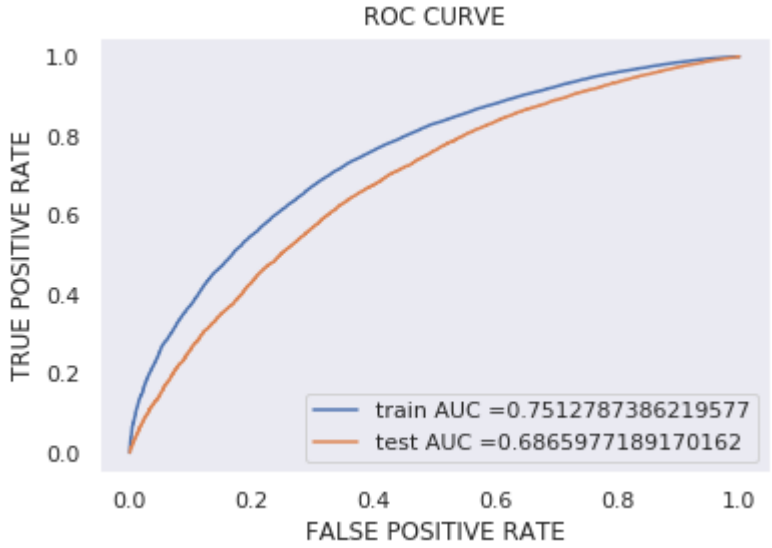
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#skle
arn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

neigh = xgb.XGBClassifier(max_depth = 4, n_estimators = 100, n_jobs = -1)
neigh.fit(X_tr_m1, y_train)

y_train_pred = neigh.predict_proba(X_tr_m1)[:,1]
y_test_pred = neigh.predict_proba(X_te_m1)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE")
plt.ylabel("TRUE POSITIVE RATE")
plt.title("ROC CURVE")
plt.grid()
plt.show()
```



In [97]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, 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.rou
nd(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

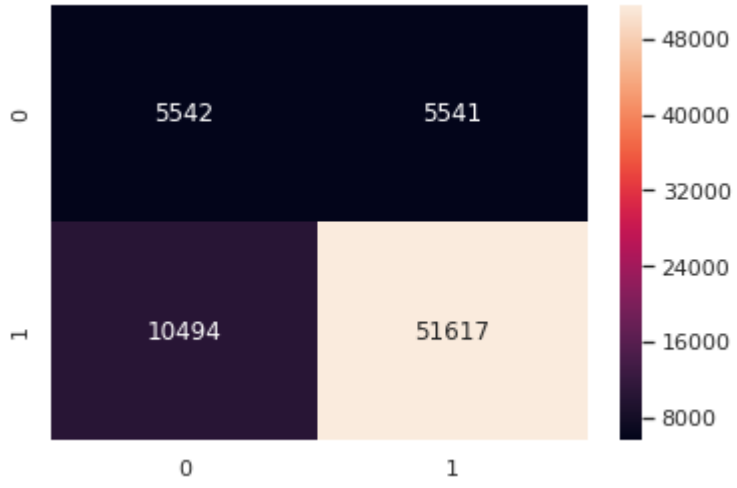
In [98]:

```
import seaborn as sns; sns.set()
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
conf_mat_data = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fp
r, train_fpr))
sns.set(font_scale = 1.0)
sns.heatmap(conf_mat_data, annot=True, fmt='g')
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2499999979647145 for threshold 0.802
```

Out[98]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe04582f748>
```



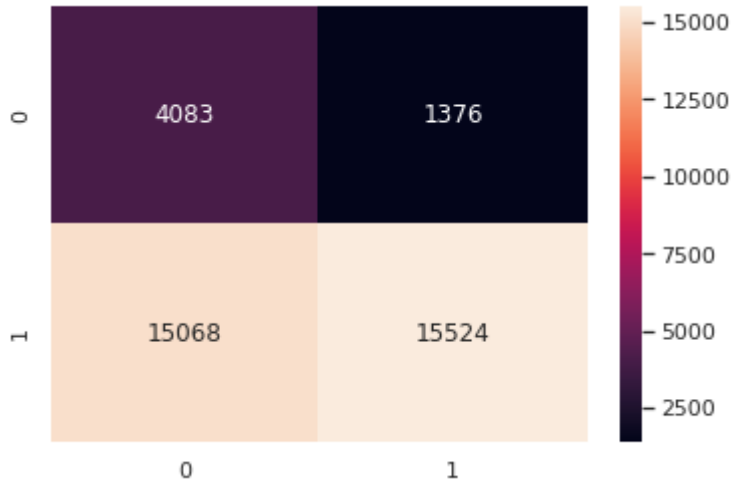
In [99]:

```
print("Test confusion matrix")
sns.set(font_scale = 1.0)
conf_mat_data = confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr,
test_fpr))
sns.heatmap(conf_mat_data, annot=True, fmt='g')
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.864
```

Out[99]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe04b22ed68>
```



2.5.2 Applying XGBOOST on TFIDF, SET 2

In [100]:

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

X_tr_m2 = csr_matrix(X_tr_m1)
X_te_m2 = csr_matrix(X_te_m1)
```

In [101]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchC
V.html
from sklearn.model_selection import GridSearchCV
import xgboost as xgb

neigh = xgb.XGBClassifier(n_jobs=-1)
parameters = {'max_depth': [2, 4, 6, 8], 'n_estimators': [5, 10, 50, 100]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc', return_train_score=True)
clf.fit(X_tr_m2, y_train)

train_auc= clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']
```

In [102]:

```
train_auc
```

Out[102]:

```
array([0.59307484, 0.61180976, 0.66877343, 0.69872358, 0.63623711,
       0.6526147 , 0.72503972, 0.76906201, 0.67279492, 0.6963472 ,
       0.79358579, 0.84731312, 0.71065896, 0.74756383, 0.86530365,
       0.91497735])
```

In [103]:

```
#making dataframe for train_auc values with max_depth & min_sample_split:
d = {'n_estimators': [5, 10, 50, 100],
    'max_depth = 2': [0.59307484, 0.61180976, 0.66877343, 0.69872358],
    'max_depth = 4': [0.63623711, 0.6526147 , 0.72503972, 0.76906201],
    'max_depth = 6': [0.67279492, 0.6963472, 0.79358579, 0.84731312],
    'max_depth = 8': [0.71065896, 0.74756383, 0.86530365, 0.91497735]}
df = pd.DataFrame(d).set_index('n_estimators')
df
```

Out[103]:

n_estimators	max_depth = 2	max_depth = 4	max_depth = 6	max_depth = 8
5	0.593075	0.636237	0.672795	0.710659
10	0.611810	0.652615	0.696347	0.747564
50	0.668773	0.725040	0.793586	0.865304
100	0.698724	0.769062	0.847313	0.914977

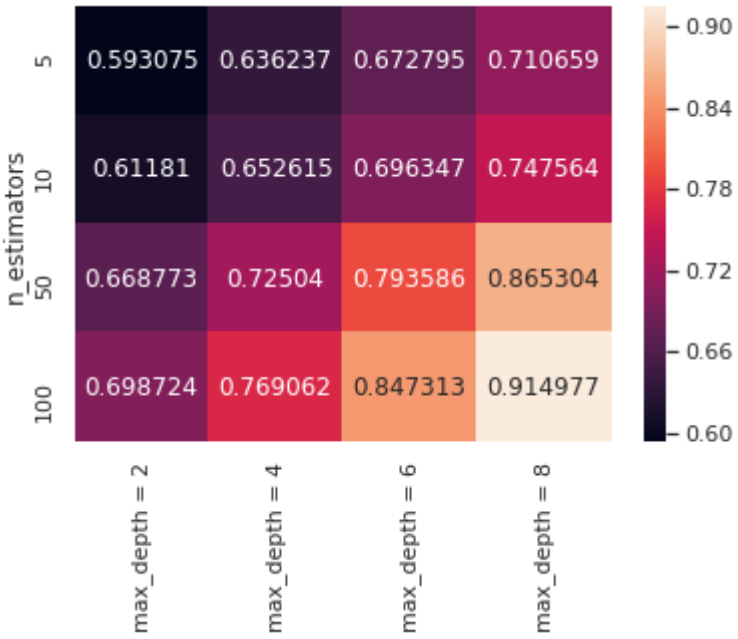
In [104]:

```
#heatmap for train_auc in each cases:
import seaborn as sns; sns.set()

sns.set(font_scale = 1.0)
sns.heatmap(df, annot=True, fmt='g')
```

Out[104]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe04b2d1940>
```



In [105]:

```
cv_auc
```

Out[105]:

```
array([0.58790668, 0.60720974, 0.65661936, 0.67899818, 0.62054062,
       0.63218887, 0.67904531, 0.69950358, 0.63417177, 0.64205152,
       0.68810446, 0.70460002, 0.63681729, 0.64772985, 0.69332023,
       0.70837328])
```

In [106]:

```
#cv_auc values in each cases:
d = {'n_estimators': [5, 10, 50, 100],
     'max_depth = 2': [0.58790668, 0.60720974, 0.65661936, 0.67899818],
     'max_depth = 4': [0.62054062, 0.63218887, 0.67904531, 0.69950358],
     'max_depth = 6': [0.63417177, 0.64205152, 0.68810446, 0.70460002],
     'max_depth = 8': [0.63681729, 0.64772985, 0.69332023, 0.70837328]}
df = pd.DataFrame(d).set_index('n_estimators')
df
```

Out[106]:

n_estimators	max_depth = 2	max_depth = 4	max_depth = 6	max_depth = 8
5	0.587907	0.620541	0.634172	0.636817
10	0.607210	0.632189	0.642052	0.647730
50	0.656619	0.679045	0.688104	0.693320
100	0.678998	0.699504	0.704600	0.708373

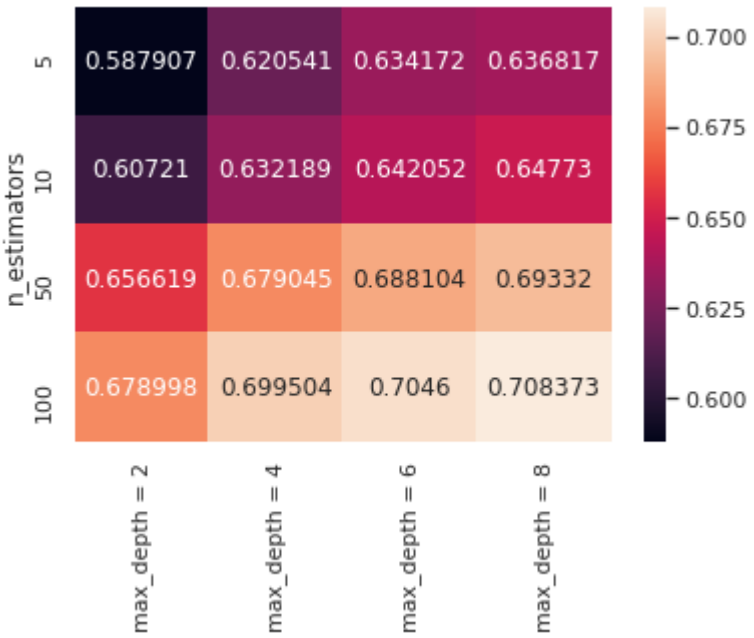
In [107]:

```
#heatmap for cv_auc in each cases:
import seaborn as sns; sns.set()

sns.set(font_scale = 1.0)
sns.heatmap(df, annot=True, fmt='g')
```

Out[107]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe045780b70>
```



In [108]:

```
#1.from the heatmap plot we choose max_depth and  n_estimators such that we will have m
aximum AUC on cv data.
#2.Gap between cv_auc and train_auc should be less.

best_max_depth = 4
best_n_estimators = 100
```

In [109]:

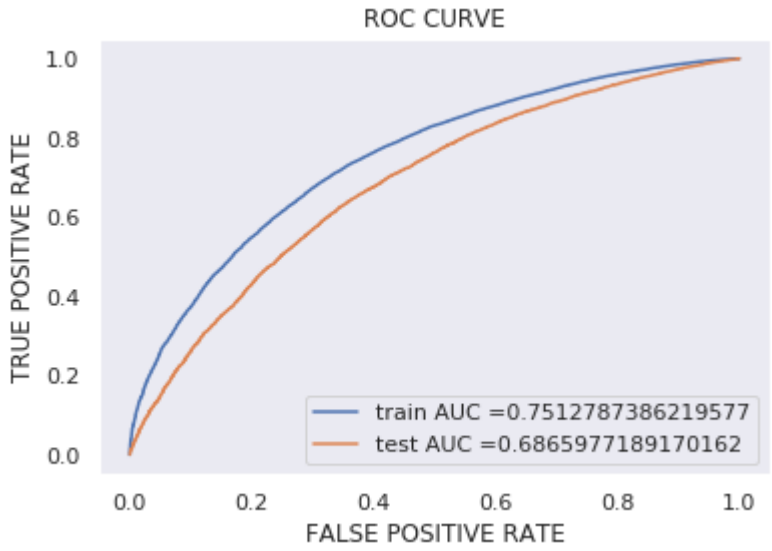
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#skle
arn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

neigh = xgb.XGBClassifier(max_depth = 4, n_estimators = 100, n_jobs = -1)
neigh.fit(X_tr_m2, y_train)

y_train_pred = neigh.predict_proba(X_tr_m2)[:,1]
y_test_pred = neigh.predict_proba(X_te_m2)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE")
plt.ylabel("TRUE POSITIVE RATE")
plt.title("ROC CURVE")
plt.grid()
plt.show()
```



In [110]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, 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.rou
nd(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

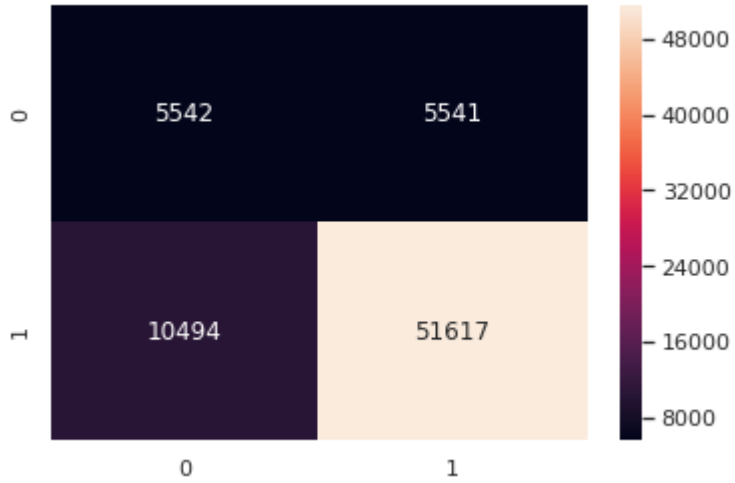
In [111]:

```
import seaborn as sns; sns.set()
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
conf_mat_data = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fp
r, train_fpr))
sns.set(font_scale = 1.0)
sns.heatmap(conf_mat_data, annot=True, fmt='g')
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2499999979647145 for threshold 0.802
```

Out[111]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe04575f358>
```



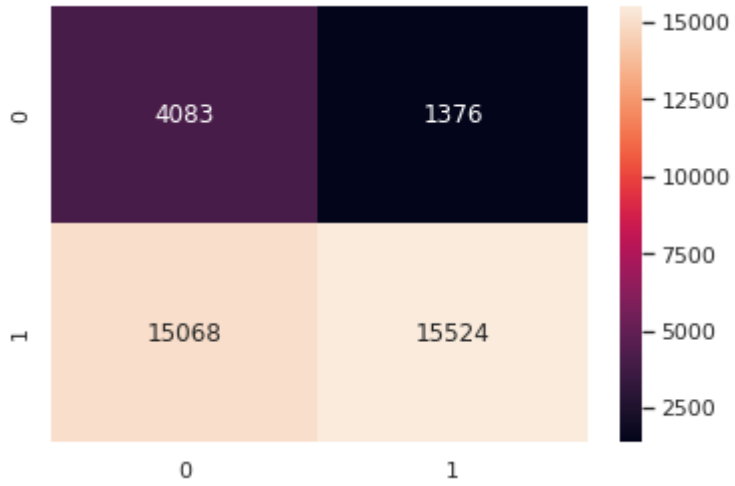
In [112]:

```
print("Test confusion matrix")
sns.set(font_scale = 1.0)
conf_mat_data = confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr,
test_fpr))
sns.heatmap(conf_mat_data, annot=True, fmt='g')
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.864
```

Out[112]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe04b8dc908>
```



2.5.3 Applying XGBOOST on AVG W2V, SET 3

In [113]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchC
V.html
from sklearn.model_selection import GridSearchCV
import xgboost as xgb

neigh = xgb.XGBClassifier(n_jobs=-1)
parameters = {'max_depth': [2, 4, 6, 8], 'n_estimators': [5, 10, 50, 100]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc', return_train_score=True)
clf.fit(X_tr_m3, y_train)

train_auc= clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']
```

In [114]:

```
train_auc
```

Out[114]:

```
array([0.63191268, 0.64501455, 0.68949684, 0.71563448, 0.66695297,
       0.68390783, 0.75814519, 0.81067357, 0.71671126, 0.75139685,
       0.87791953, 0.94385349, 0.78875754, 0.85062568, 0.98247087,
       0.99843297])
```

In [115]:

```
#making dataframe for train_auc values with max_depth & min_sample_split:
d = {'n_estimators': [5, 10, 50, 100],
     'max_depth = 2': [0.63191268, 0.64501455, 0.68949684, 0.71563448],
     'max_depth = 4': [0.66695297, 0.68390783, 0.75814519, 0.81067357],
     'max_depth = 6': [0.71671126, 0.75139685, 0.87791953, 0.94385349],
     'max_depth = 8': [0.78875754, 0.85062568, 0.98247087, 0.99843297]}
df = pd.DataFrame(d).set_index('n_estimators')
df
```

Out[115]:

n_estimators	max_depth = 2	max_depth = 4	max_depth = 6	max_depth = 8
5	0.631913	0.666953	0.716711	0.788758
10	0.645015	0.683908	0.751397	0.850626
50	0.689497	0.758145	0.877920	0.982471
100	0.715634	0.810674	0.943853	0.998433

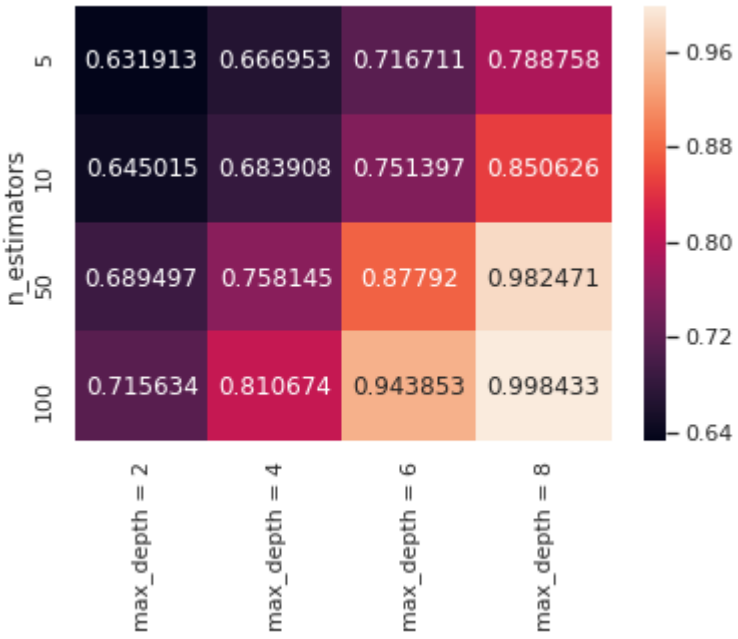
In [116]:

```
#heatmap for train_auc in each cases:
import seaborn as sns; sns.set()

sns.set(font_scale = 1.0)
sns.heatmap(df, annot=True, fmt='g')
```

Out[116]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe04567f898>
```



In [117]:

```
cv_auc
```

Out[117]:

```
array([0.61457543, 0.62976189, 0.66526735, 0.68120381, 0.63591053,
       0.64802779, 0.68336554, 0.69480754, 0.63897206, 0.65266797,
       0.68782546, 0.69426925, 0.6336304 , 0.64915086, 0.68039182,
       0.68485867])
```

In [118]:

```
#cv_auc values in each cases:
d = {'n_estimators': [5, 10, 50, 100],
    'max_depth = 2': [0.61457543, 0.62976189, 0.66526735, 0.68120381],
    'max_depth = 4': [0.63591053, 0.64802779, 0.68336554, 0.69480754],
    'max_depth = 6': [0.63897206, 0.65266797, 0.68782546, 0.69426925],
    'max_depth = 8': [0.6336304 , 0.64915086, 0.68039182, 0.68485867]}
df = pd.DataFrame(d).set_index('n_estimators')
df
```

Out[118]:

n_estimators	max_depth = 2	max_depth = 4	max_depth = 6	max_depth = 8
5	0.614575	0.635911	0.638972	0.633630
10	0.629762	0.648028	0.652668	0.649151
50	0.665267	0.683366	0.687825	0.680392
100	0.681204	0.694808	0.694269	0.684859

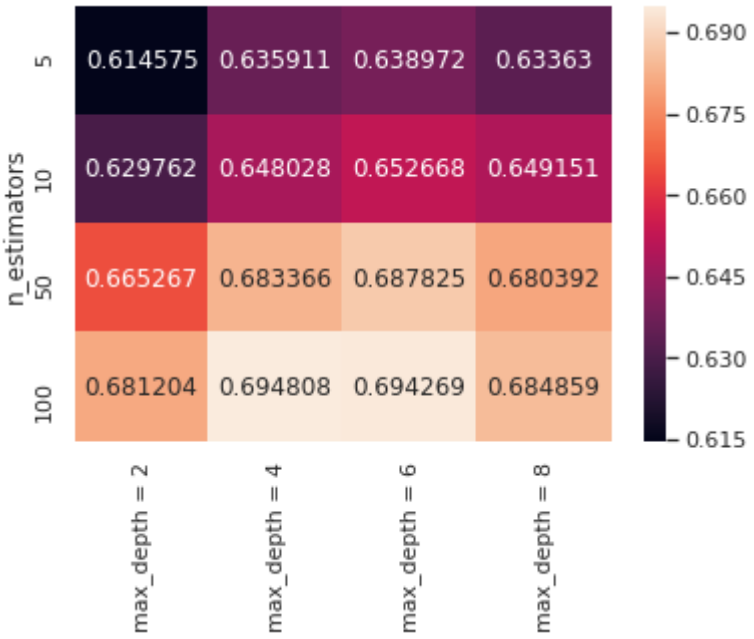
In [119]:

```
#heatmap for cv_auc in each cases:
import seaborn as sns; sns.set()

sns.set(font_scale = 1.0)
sns.heatmap(df, annot=True, fmt='g')
```

Out[119]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe04b97d3c8>
```



In [120]:

```
#1.from the heatmap plot we choose max_depth and  n_estimators such that we will have m
aximum AUC on cv data.
#2.Gap between cv_auc and train_auc should be less.

best_max_depth = 2
best_n_estimators = 100
```

In [121]:

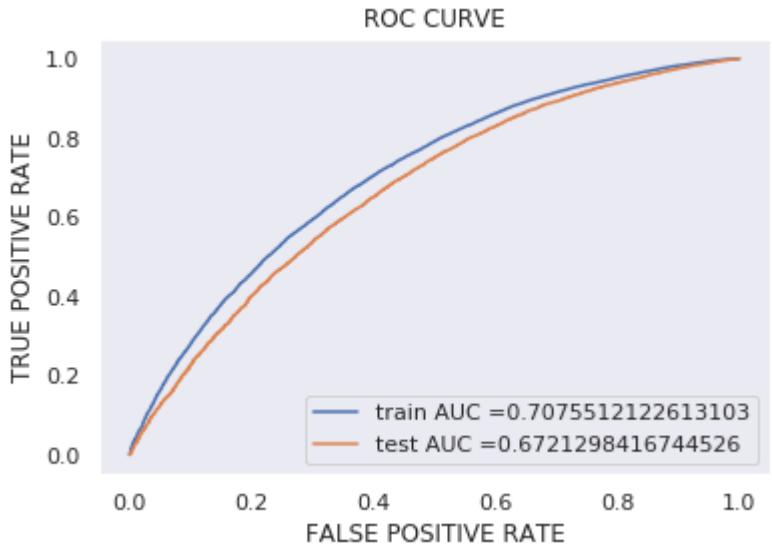
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

neigh = xgb.XGBClassifier(max_depth = 2, n_estimators = 100, n_jobs = -1)
neigh.fit(X_tr_m3, y_train)

y_train_pred = neigh.predict_proba(X_tr_m3)[:,1]
y_test_pred = neigh.predict_proba(X_te_m3)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE")
plt.ylabel("TRUE POSITIVE RATE")
plt.title("ROC CURVE")
plt.grid()
plt.show()
```



In [122]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, 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.rou
nd(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

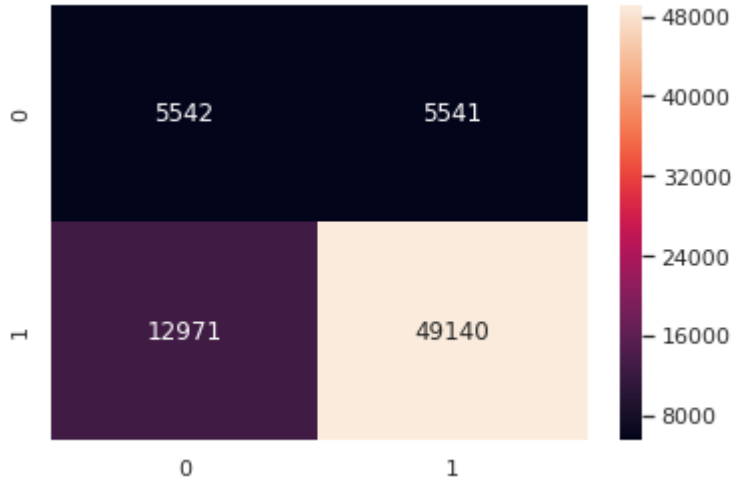
In [123]:

```
import seaborn as sns; sns.set()
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
conf_mat_data = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fp
r, train_fpr))
sns.set(font_scale = 1.0)
sns.heatmap(conf_mat_data, annot=True, fmt='g')
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2499999979647145 for threshold 0.816
```

Out[123]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe04047ed30>
```



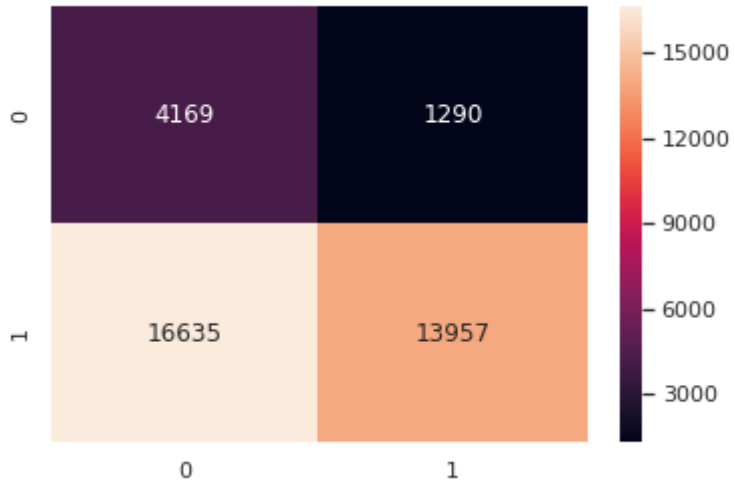
In [124]:

```
print("Test confusion matrix")
sns.set(font_scale = 1.0)
conf_mat_data = confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr,
test_fpr))
sns.heatmap(conf_mat_data, annot=True, fmt='g')
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.875
```

Out[124]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe04b3e0080>
```



2.5.4 Applying XGBOOST on TFIDF W2V, SET 4

In [125]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchC
V.html
from sklearn.model_selection import GridSearchCV
import xgboost as xgb

neigh = xgb.XGBClassifier(n_jobs=-1)
parameters = {'max_depth': [2, 4, 6, 8], 'n_estimators': [5, 10, 50, 100]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc', return_train_score=True)
clf.fit(X_tr_m4, y_train)

train_auc= clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']
```

In [126]:

```
train_auc
```

Out[126]:

```
array([0.63867299, 0.65221645, 0.69454917, 0.71844469, 0.67258752,
       0.69014227, 0.75922259, 0.8051326 , 0.7283429 , 0.75856502,
       0.87505137, 0.9379527 , 0.79964246, 0.85437922, 0.98023781,
       0.99748375])
```

In [127]:

```
#making dataframe for train_auc values with max_depth & min_sample_split:
d = {'n_estimators': [5, 10, 50, 100],
     'max_depth = 2': [0.63867299, 0.65221645, 0.69454917, 0.71844469],
     'max_depth = 4': [0.67258752, 0.69014227, 0.75922259, 0.8051326],
     'max_depth = 6': [0.7283429 , 0.75856502, 0.87505137, 0.9379527],
     'max_depth = 8': [0.79964246, 0.85437922, 0.98023781, 0.99748375]}
df = pd.DataFrame(d).set_index('n_estimators')
df
```

Out[127]:

n_estimators	max_depth = 2	max_depth = 4	max_depth = 6	max_depth = 8
5	0.638673	0.672588	0.728343	0.799642
10	0.652216	0.690142	0.758565	0.854379
50	0.694549	0.759223	0.875051	0.980238
100	0.718445	0.805133	0.937953	0.997484

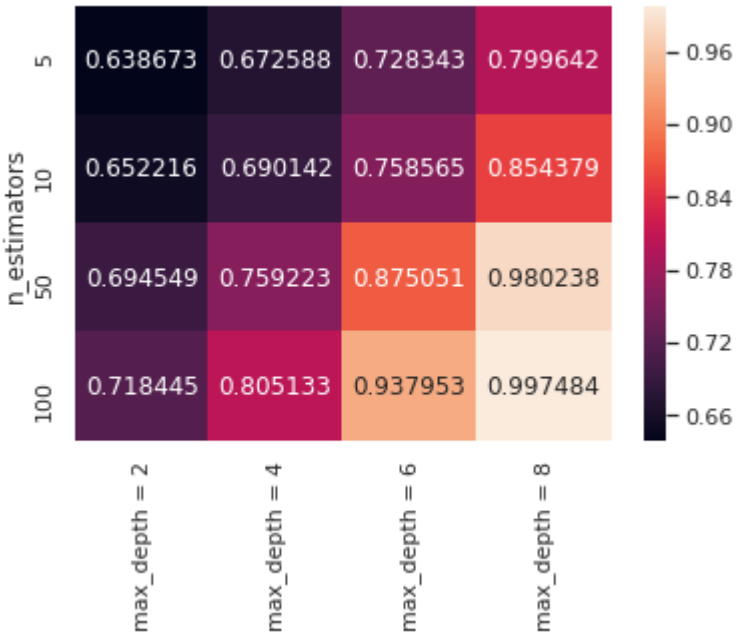
In [128]:

```
#heatmap for train_auc in each cases:
import seaborn as sns; sns.set()

sns.set(font_scale = 1.0)
sns.heatmap(df, annot=True, fmt='g')
```

Out[128]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe04b4f7780>
```



In [129]:

```
cv_auc
```

Out[129]:

```
array([0.62644669, 0.64047626, 0.6725003 , 0.6872128 , 0.64467989,
       0.65616703, 0.68875531, 0.69797807, 0.64985773, 0.66327629,
       0.69208806, 0.69701586, 0.64498734, 0.6591422 , 0.68868132,
       0.69117662])
```

In [130]:

```
#cv_auc values in each cases:
d = {'n_estimators': [5, 10, 50, 100],
    'max_depth = 2': [0.62644669, 0.64047626, 0.6725003, 0.6872128],
    'max_depth = 4': [0.64467989, 0.65616703, 0.68875531, 0.69797807],
    'max_depth = 6': [0.64985773, 0.66327629, 0.69208806, 0.69701586],
    'max_depth = 8': [0.64498734, 0.6591422 , 0.68868132, 0.69117662]}
df = pd.DataFrame(d).set_index('n_estimators')
df
```

Out[130]:

n_estimators	max_depth = 2	max_depth = 4	max_depth = 6	max_depth = 8
5	0.626447	0.644680	0.649858	0.644987
10	0.640476	0.656167	0.663276	0.659142
50	0.672500	0.688755	0.692088	0.688681
100	0.687213	0.697978	0.697016	0.691177

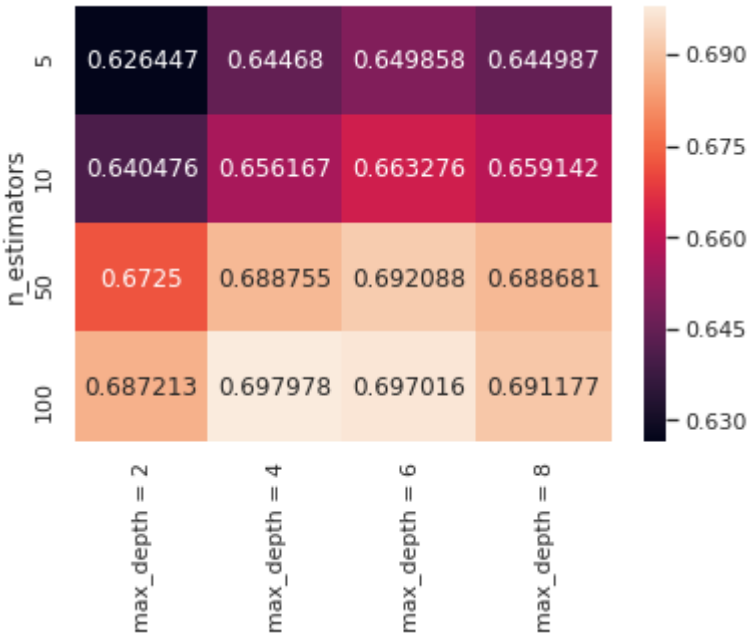
In [131]:

```
#heatmap for cv_auc in each cases:
import seaborn as sns; sns.set()

sns.set(font_scale = 1.0)
sns.heatmap(df, annot=True, fmt='g')
```

Out[131]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe044d424a8>
```



In [132]:

```
#1.from the heatmap plot we choose max_depth and  n_estimators such that we will have m
aximum AUC on cv data.
#2.Gap between cv_auc and train_auc should be less.

best_max_depth = 2
best_n_estimators = 100
```

In [133]:

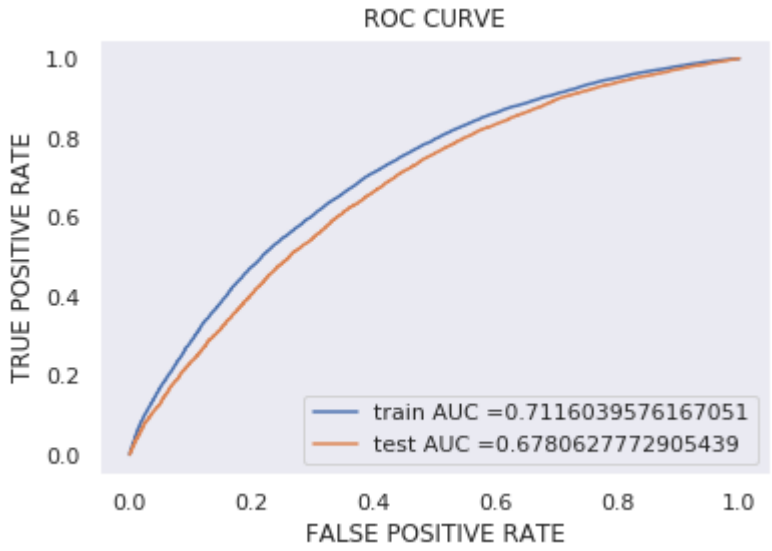
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

neigh = xgb.XGBClassifier(max_depth = 2, n_estimators = 100, n_jobs = -1)
neigh.fit(X_tr_m4, y_train)

y_train_pred = neigh.predict_proba(X_tr_m4)[:,1]
y_test_pred = neigh.predict_proba(X_te_m4)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE")
plt.ylabel("TRUE POSITIVE RATE")
plt.title("ROC CURVE")
plt.grid()
plt.show()
```



In [134]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, 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.rou
nd(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

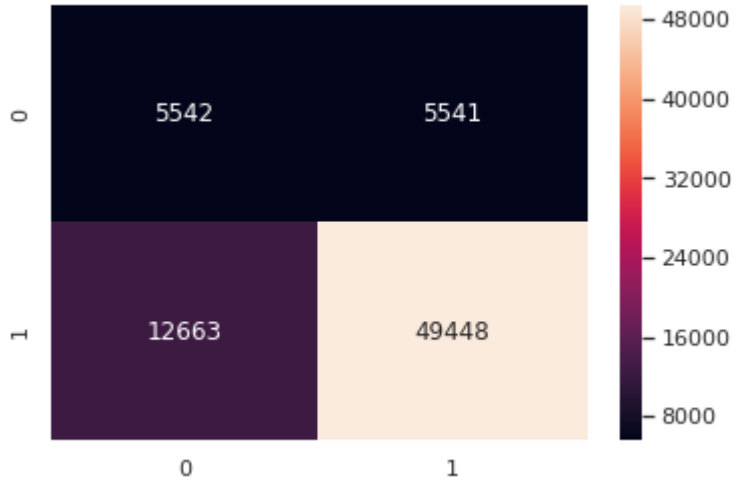
In [135]:

```
import seaborn as sns; sns.set()
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
conf_mat_data = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fp
r, train_fpr))
sns.set(font_scale = 1.0)
sns.heatmap(conf_mat_data, annot=True, fmt='g')
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2499999979647145 for threshold 0.81
```

Out[135]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe044db86a0>
```



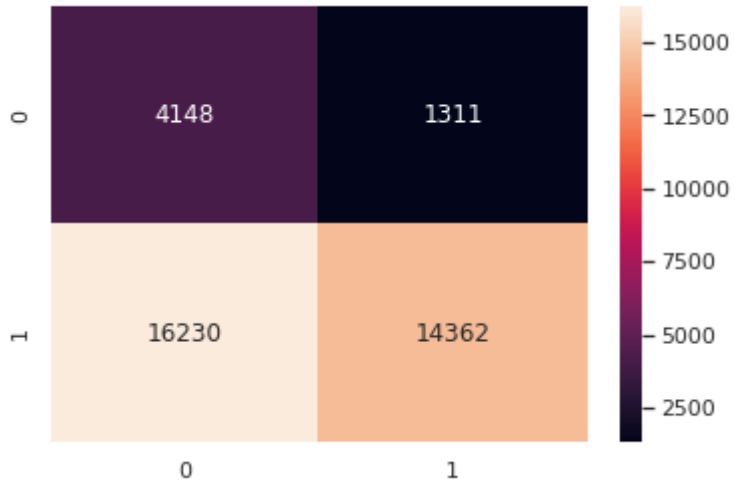
In [136]:

```
print("Test confusion matrix")
sns.set(font_scale = 1.0)
conf_mat_data = confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr,
test_fpr))
sns.heatmap(conf_mat_data, annot=True, fmt='g')
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.872
```

Out[136]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe045832400>
```



3. Conclusion

In [137]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Max_Depth", "n_estimators", "AUC"]

x.add_row(["set1(BOW)", "Random Forest", 20, 10, 0.64])
x.add_row(["set2(TFIDF)", "Random Forest", 20, 10, 0.62])
x.add_row(["set3(W2V)", "Random Forest", 10, 10, 0.63])
x.add_row(["set4(TFIDFW2V)", "Random Forest", 10, 10, 0.65])
print(x)
```

```
+---------------+---------------+-----------+--------------+------+
|   Vectorizer  |     Model     | Max_Depth | n_estimators | AUC  |
+---------------+---------------+-----------+--------------+------+
|   set1(BOW)   | Random Forest |     20    |      10      | 0.64 |
|  set2(TFIDF)  | Random Forest |     20    |      10      | 0.62 |
|   set3(W2V)   | Random Forest |     10    |      10      | 0.63 |
| set4(TFIDFW2V) | Random Forest |     10    |      10      | 0.65 |
+---------------+---------------+-----------+--------------+------+
```

In [138]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Max_Depth", "n_estimators", "AUC"]

x.add_row(["set1(BOW)", "XGBOOST Decision Trees", 4, 100, 0.68])
x.add_row(["set2(TFIDF)", "XGBOOST Decision Trees", 4, 100, 0.68])
x.add_row(["set3(W2V)", "XGBOOST Decision Trees", 4, 100, 0.67])
x.add_row(["set4(TFIDFW2V)", "XGBOOST Decision Trees", 4, 100, 0.67])
print(x)
```

```
+---------------+-----------------------+-----------+--------------+----
--+
|   Vectorizer  |         Model         | Max_Depth | n_estimators | AUC
|
+---------------+-----------------------+-----------+--------------+----
--+
|   set1(BOW)   | XGBOOST Decision Trees |     4     |     100      | 0.6
8 |
|  set2(TFIDF)  | XGBOOST Decision Trees |     4     |     100      | 0.6
8 |
|   set3(W2V)   | XGBOOST Decision Trees |     4     |     100      | 0.6
7 |
| set4(TFIDFW2V) | XGBOOST Decision Trees |     4     |     100      | 0.6
7 |
+---------------+-----------------------+-----------+--------------+----
--+
```