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	Desc
`project_id`	A unique identifier for the proposed project. **Example:** `p0`
`project_title`	Title of the project. **Exa** • `Art Will Make You H` • `First Grad`
`project_grade_category`	Grade level of students for which the project is targeted. One of the fo enumerated v • `Grades P` • `Grade` • `Grade` • `Grades`

Feature	Desc
project_subject_categories	One or more (comma-separated) subject categories for the project fr following enumerated list of v • Applied Lea • Care & H • Health & S • History & C • Literacy & Lan • Math & Sc • Music & The • Special • W **Exan** • Music & The • Literacy & Language, Math & Sc
school_state	State where school is located (Two-letter U.S. posta (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Postal_c **Exampl**
project_subject_subcategories	One or more (comma-separated) subject subcategories for the p **Exan** • Lit • Literature & Writing, Social Sci
project_resource_summary	An explanation of the resources needed for the project. **Exa** • My students need hands on literacy materials to ma sensory needs!<
project_essay_1	First application
project_essay_2	Second application
project_essay_3	Third application
project_essay_4	Fourth application
project_submitted_datetime	Datetime when project application was submitted. **Example:** 2016-0 12:43:5
teacher_id	A unique identifier for the teacher of the proposed project. **Exa** bdf8baa8fedef6bfeec7ae4ff1c
teacher_prefix	Teacher's title. One of the following enumerated v • • • • • • Tea
teacher_number_of_previously_posted_projects	Number of project applications previously submitted by the same te **Exam**

* 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`

Feature	Description
`quantity`	Quantity of the resource required. **Example:** `3`
`price`	Price of the resource required. **Example:** `9.95`

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

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

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

Notes on the Essay Data

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

- **project_essay_1:** "Introduce us to your classroom"
- **project_essay_2:** "Tell us more about your students"
- **project_essay_3:** "Describe how your students will use the materials you're requesting"
- **project_essay_3:** "Close by sharing why your project will make a difference"

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

- **project_essay_1:** "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."
- **project_essay_2:** "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

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

In [2]:

```
%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 [3]:

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

In [4]:

```
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' 's
chool_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 [5]:

```
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[5]:

	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

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/473019

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

project_data['clean_categories'] = cat_list
project_data.drop(['project_subject_categories'], axis=1, inplace=True)

from collections import Counter
my_counter = Counter()
for word in project_data['clean_categories'].values:
    my_counter.update(word.split())

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

1.3 preprocessing of project_subject_subcategories

In [7]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/473019

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

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

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

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

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

1.4 preprocessing of project_grade_category

In [8]:

```
preproc = []
# tqdm is for printing the status bar
for sent in project_data['project_grade_category']:
    sent = sent.replace('Grades', ' ')
    sent = sent.replace('-', '_')
    preproc.append(sent)
project_data['project_grade_category']=preproc
```

In [9]:

```
my_counter = Counter()
for word in project_data['project_grade_category'].values:
    my_counter.update(word.split())

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

1.3 Text preprocessing

In [10]:

```
# 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 [11]:

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

Out[11]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	

In [12]:

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

In [13]:

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

```
My students are English learners that are working on English as their second
or third languages. We are a melting pot of refugees, immigrants, and native
-born Americans bringing the gift of language to our school. \r\n\r\n We hav
e over 24 languages represented in our English Learner program with students
at every level of mastery.  We also have over 40 countries represented with
the families within our school.  Each student brings a wealth of knowledge a
nd experiences to us that open our eyes to new cultures, beliefs, and respec
t.\"The limits of your language are the limits of your world.\"-Ludwig Wittg
enstein  Our English learner's have a strong support system at home that beg
s for more resources.  Many times our parents are learning to read and speak
English along side of their children.  Sometimes this creates barriers for p
arents to be able to help their child learn phonetics, letter recognition, a
nd other reading skills.\r\n\r\nBy providing these dvd's and players, studen
ts are able to continue their mastery of the English language even if no one
at home is able to assist.  All families with students within the Level 1 pr
oficiency status, will be a offered to be a part of this program.  These edu
cational 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 deve
lop early reading skills.\r\n\r\nParents that do not have access to a dvd pl
ayer will have the opportunity to check out a dvd player to use for the yea
r.  The plan is to use these videos and educational dvd's for the years to c
ome for other EL students.\r\nnannan
==================================================
The 51 fifth grade students that will cycle through my classroom this year a
ll love learning, at least most of the time. At our school, 97.3% of the stu
dents receive free or reduced price lunch. Of the 560 students, 97.3% are mi
nority students. \r\nThe school has a vibrant community that loves to get to
gether and celebrate. Around Halloween there is a whole school parade to sho
w 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 games. At the e
nd of the year the school hosts a carnival to celebrate the hard work put in
during the school year, with a dunk tank being the most popular activity.My
students will use these five brightly colored Hokki stools in place of regul
ar, stationary, 4-legged chairs. As I will only have a total of ten in the c
lassroom and not enough for each student to have an individual one, they wil
l 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 th
em in place of chairs at my small group tables during math and reading time
s. The rest of the day they will be used by the students who need the highes
t amount of movement in their life in order to stay focused on school.\r\n\r
\nWhenever asked what the classroom is missing, my students always say more
Hokki Stools. They can't get their fill of the 5 stools we already have. Whe
n the students are sitting in group with me on the Hokki Stools, they are al
ways moving, but at the same time doing their work. Anytime the students get
to pick where they can sit, the Hokki Stools are the first to be taken. Ther
e are always students who head over to the kidney table to get one of the st
ools 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 comprom
ise that allow my students to do desk work and move at the same time. These
stools will help students to meet their 60 minutes a day of movement by allo
wing them to activate their core muscles for balance while they sit. For man
y of my students, these chairs will take away the barrier that exists in sch
ools for a child who can't sit still.nannan
==================================================
How do you remember your days of school? Was it in a sterile environment wit
h plain walls, rows of desks, and a teacher in front of the room? A typical
day in our room is nothing like that. I work hard to create a warm inviting
themed room for my students look forward to coming to each day.\r\n\r\nMy cl
ass is made up of 28 wonderfully unique boys and girls of mixed races in Ark
ansas.\r\nThey attend a Title I school, which means there is a high enough p
ercentage of free and reduced-price lunch to qualify. Our school is an \"ope
n 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; th
ey 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 nauti
cal environment. Creating a classroom environment is very important in the s
uccess 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, hav
e them developed, and then hung in our classroom ready for their first day o
f 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\nYou
r generous donations will help me to help make our classroom a fun, invitin
g, learning environment from day one.\r\n\r\nIt costs lost of money out of m
y own pocket on resources to get our classroom ready. Please consider helpin
g with this project to make our new school year a very successful one. Thank
you!nannan
==================================================
My kindergarten students have varied disabilities ranging from speech and la
nguage delays, cognitive delays, gross/fine motor delays, to autism. They ar
e eager beavers and always strive to work their hardest working past their l
imitations. \r\n\r\nThe materials we have are the ones I seek out for my stu
dents. I teach in a Title I school where most of the students receive free o
r reduced price lunch.  Despite their disabilities and limitations, my stude
nts love coming to school and come eager to learn and explore.Have you ever
felt like you had ants in your pants and you needed to groove and move as yo
u were in a meeting? This is how my kids feel all the time. The want to be a
ble to move as they learn or so they say.Wobble chairs are the answer and I
love then because they develop their core, which enhances gross motor and in
Turn fine motor skills. \r\nThey also want to learn through games, my kids d
on't want to sit and do worksheets. They want to learn to count by jumping a
nd playing. Physical engagement is the key to our success. The number toss a
nd color and shape mats can make that happen. My students will forget they a
re doing work and just have the fun a 6 year old deserves.nannan
==================================================
The mediocre teacher tells. The good teacher explains. The superior teacher
demonstrates. The great teacher inspires. -William A. Ward\r\n\r\nMy school
has 803 students which is makeup is 97.6% African-American, making up the la
rgest segment of the student body. A typical school in Dallas is made up of
23.2% African-American students. Most of the students are on free or reduced
lunch. We aren't receiving doctors, lawyers, or engineers children from rich
backgrounds or neighborhoods. As an educator I am inspiring minds of young c
hildren and we focus not only on academics but one smart, effective, efficie
nt, and disciplined students with good character.In our classroom we can uti
lize the Bluetooth for swift transitions during class. I use a speaker which
```

```
doesn't amplify the sound enough to receive the message. Due to the volume o
f my speaker my students can't hear videos or books clearly and it isn't mak
ing the lessons as meaningful. But with the bluetooth speaker my students wi
ll be able to hear and I can stop, pause and replay it at any time.\r\nThe c
art will allow me to have more room for storage of things that are needed fo
r the day and has an extra part to it I can use.  The table top chart has al
l of the letter, words and pictures for students to learn about different le
tters and it is more accessible.nannan
==================================================
```

In [14]:

```
# 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 [15]:

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

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

In [16]:

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

```
My kindergarten students have varied disabilities ranging from speech and la
nguage delays, cognitive delays, gross/fine motor delays, to autism. They ar
e eager beavers and always strive to work their hardest working past their l
imitations.     The materials we have are the ones I seek out for my student
s. I teach in a Title I school where most of the students receive free or re
duced price lunch.  Despite their disabilities and limitations, my students
love coming to school and come eager to learn and explore.Have you ever felt
like you had ants in your pants and you needed to groove and move as you wer
e in a meeting? This is how my kids feel all the time. The want to be able t
o move as they learn or so they say.Wobble chairs are the answer and I love
then because they develop their core, which enhances gross motor and in Turn
fine motor skills.   They also want to learn through games, my kids do not w
ant to sit and do worksheets. They want to learn to count by jumping and pla
ying. Physical engagement is the key to our success. The number toss and col
or and shape mats can make that happen. My students will forget they are doi
ng work and just have the fun a 6 year old deserves.nannan
```

In [17]:

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

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

In [18]:

```
# 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'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they'
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'l
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'u
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'd
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any',
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'v
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'do
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn'
            'won', "won't", 'wouldn', "wouldn't"]
```

In [20]:

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

```
100%|████████████████████████████████████████████████████████████████
████████████████████████████████████| 109248/109248 [01:05<00:0
0, 1680.18it/s]
```

In [21]:

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

Out[21]:

```
'kindergarten students varied disabilities ranging speech language delays co
gnitive delays gross fine motor delays autism eager beavers always strive wo
rk hardest working past limitations materials ones seek students teach title
school students receive free reduced price lunch despite disabilities limita
tions students love coming school come eager learn explore ever felt like an
ts pants needed groove move meeting kids feel time want able move learn say
wobble chairs answer love develop core enhances gross motor turn fine motor
skills also want learn games kids not want sit worksheets want learn count j
umping playing physical engagement key success number toss color shape mats
make happen students forget work fun 6 year old deserves nannan'
```

1.4 Preprocessing of `project_title`

In [24]:

```
# similarly you can preprocess the titles also

preprocessed_title=[]
for tit in tqdm(project_data['project_title'].values):
    tit = decontracted(tit)
    tit = tit.replace('\\r', ' ')
    tit = tit.replace('\\"', ' ')
    tit = tit.replace('\\n', ' ')
    tit = re.sub('[^A-Za-z0-9]+', ' ', tit)
    tit = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed_title.append(tit.lower().strip())
print(preprocessed_title[23])
```

```
100%|████████████████████████████████████████████████████████████████| 109248/109248 [00:41<00:0
0, 2643.48it/s]

last time used math probably within last hour yet go school believing never
use students engage authentic math experiences routinely help understand cri
tical math truly teach small town big dreams fantastic opportunities surroun
d students ultimate goal help achieve success seeking students drive potenti
al take world storm graduation need little help according forbes magazine 20
14 companies seeking employees strong problem solvers highly developed commu
nication skills vast amount technical knowledge order students gain skills n
eed competitive today job market access digital devices critical funding pro
ject allowing kids chance engage lessons develop not skills abilities apply
real world meaning content learn communication collaboration creativity crit
ical thinking blossom help exploring authentic cross curricular topics bring
math life example rather memorizing formulas area perimeter properties vario
us quadrilaterals students utilized tech resources plan create blueprints ne
w home another teacher whose family growing took blueprints step exploring r
atios scale factors building scale models designs nannan
```

Sentiment Analysis of essays

In [25]:

```
import nltk
nltk.downloader.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer
analyser = SentimentIntensityAnalyzer()

neg = []
pos = []
neu = []
compound = []

for a in tqdm(project_data["essay"]) :
    b = analyser.polarity_scores(a)['neg']
    c = analyser.polarity_scores(a)['pos']
    d = analyser.polarity_scores(a)['neu']
    e = analyser.polarity_scores(a)['compound']
    neg.append(b)
    pos.append(c)
    neu.append(d)
    compound.append(e)
```

```
[nltk_data] Downloading package vader_lexicon to C:\Users\Hemant
[nltk_data]     Gusain\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 109248/109248 [22:00<0
0:00, 78.47it/s]
```

In [26]:

```
project_data["pos"] = pos
```

In [27]:

```
project_data["neg"] = neg
```

In [28]:

```
project_data["neu"] = neu
```

In [29]:

```
project_data["compound"] = compound
```

Number of Words in Title

In [30]:

```
title_word_count = []

for a in project_data["project_title"] :
    b = len(a.split())
    title_word_count.append(b)

project_data["title_word_count"] = title_word_count
```

Number of Words in Essays

In [31]:

```
essay_word_count = []

for a in project_data["essay"] :
    b = len(a.split())
    essay_word_count.append(b)

project_data["essay_word_count"] = essay_word_count
```

1.5 Preparing data for models

In [32]:

```
project_data.columns
```

Out[32]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', 'project_titl
e',
       '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', 'pos', 'neg', 'ne
u',
       'compound', 'title_word_count', 'essay_word_count'],
      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 [33]:

```
# 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, bina
categories_one_hot = vectorizer.fit_transform(project_data['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",categories_one_hot.shape)
```

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

In [34]:

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

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement',
'Extracurricular', 'Civics_Government', 'ForeignLanguages', 'NutritionEducat
ion', 'Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingArts', 'Characte
rEducation', 'TeamSports', 'Other', 'College_CareerPrep', 'Music', 'History_
Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL', 'Gym_Fitness',
'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig  (109248, 30)
```

In [35]:

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

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [36]:

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

```
Shape of matrix after one hot encodig  (109248, 16512)
```

In [37]:

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

1.5.2.2 TFIDF vectorizer

In [38]:

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

```
Shape of matrix after one hot encodig  (109248, 16512)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [39]:

```
'''
# 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-pickl

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


'''
```

Out[39]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4
```

```
084039\ndef (https://stackoverflow.com/a/38230349/4084039\ndef) loadGloveMod
el(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFil
e,\'r\', encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n
splitLine = line.split()\n        word = splitLine[0]\n        embedding = n
p.array([float(val) for val in splitLine[1:]])\n        model[word] = embedd
ing\n    print ("Done.",len(model)," words loaded!")\n    return model\nmode
l = loadGloveModel(\'glove.42B.300d.txt\')\n\n# ============================
\nOutput:\n    \nLoading Glove Model\n1917495it [06:32, 4879.69it/s]\nDone.
 1917495  words loaded!\n\n# ============================\n\nwords = []\nfor
i in preproced_texts:\n    words.extend(i.split(\' \'))\n\nfor i in preproce
d_titles:\n    words.extend(i.split(\' \'))\nprint("all the words in the cou
pus", len(words))\nwords = set(words)\nprint("the unique words in the coupu
s", len(words))\n\ninter_words = set(model.keys()).intersection(words)\nprin
t("The number of words that are present in both glove vectors and our coupu
s",       len(inter_words),"(",np.round(len(inter_words)/len(words)*100,
3),"%)")\n\nwords_courpus = {}\nwords_glove = set(model.keys())\nfor i in wo
rds:\n    if i in words_glove:\n        words_courpus[i] = model[i]\nprint
("word 2 vec length", len(words_courpus))\n\n\n# stronging variables into pi
ckle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-
load-variables-in-python/\n\nimport (http://www.jessicayung.com/how-to-use-p
ickle-to-save-and-load-variables-in-python/\n\nimport) pickle\nwith open(\'g
love_vectors\', \'wb\') as f:\n    pickle.dump(words_courpus, f)\n\n\n'
```

In [40]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickl
# 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 [41]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors.append(vector)

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

```
100%|████████████████████████████████████████████████████████████████
████████████████████████████████████████| 109248/109248 [00:36<00:0
0, 3007.62it/s]

109248
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [42]:

```
# 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 [43]:

```
# 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((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettir
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors.append(vector)

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

```
100%|████████████████████████████████████████████████████████████████
████████████████████████████████████████| 109248/109248 [04:32<00:
00, 400.57it/s]

109248
300
```

In [44]:

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

1.5.3 Vectorizing Numerical features

In [45]:

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

In [46]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.prepro
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.
# 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 standar
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])

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

```
Mean : 298.1193425966608, Standard deviation : 367.49634838483496
```

In [47]:

```
price_standardized
```

Out[47]:

```
array([[-0.3905327 ],
       [ 0.00239637],
       [ 0.59519138],
       ...,
       [-0.15825829],
       [-0.61243967],
       [-0.51216657]])
```

1.5.4 Merging all the above features

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

In [48]:

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

```
(109248, 9)
(109248, 30)
(109248, 16512)
(109248, 1)
```

In [49]:

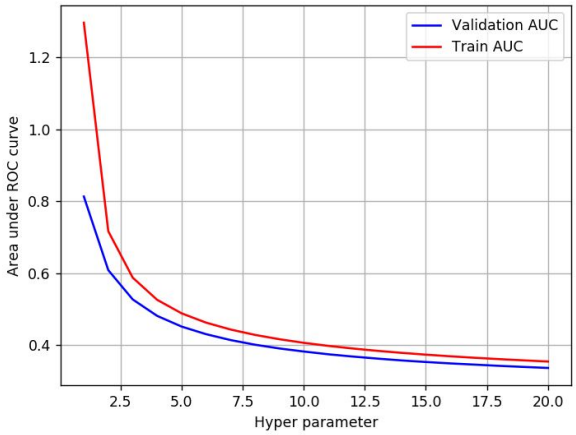
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X = hstack((categories_one_hot, sub_categories_one_hot, text_bow, price_standardized))
X.shape
```

Out[49]:

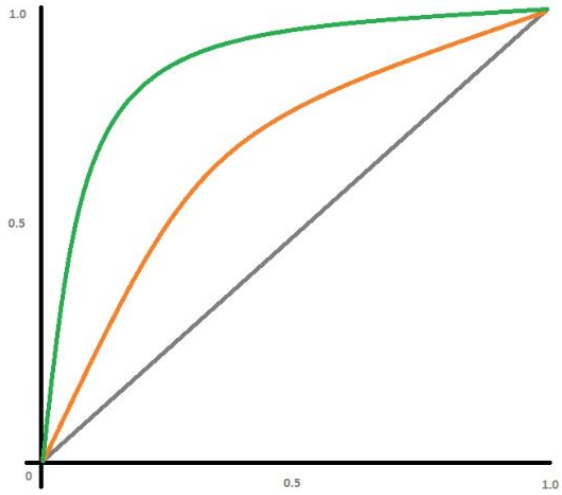
```
(109248, 16552)
```

Assignment 5: Logistic Regression

1. **[Task-1] Logistic Regression(either SGDClassifier with log loss, or LogisticRegression) on these feature sets**
   - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (`BOW with bi-grams with min_df=10 and max_features=5000`)
   - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (`TFIDF with bi-grams with min_df=10 and max_features=5000`)
   - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
   - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)
2. **Hyper paramter tuning (find best hyper parameters corresponding the algorithm that you choose)**
   - 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 or simple cross validation data
   - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning
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.



   - 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. Please visualize your confusion matrices using seaborn heatmaps.

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

(https://seaborn.pydata.org/generated/seaborn.heatmap.html)

4. **[Task-2] Apply Logistic Regression on the below feature set Set 5 by finding the best hyper parameter as suggested in step 2 and step 3.**
5. Consider these set of features Set 5 :

   - **school_state** : categorical data
   - **clean_categories** : categorical data
   - **clean_subcategories** : categorical data
   - **project_grade_category** :categorical data
   - **teacher_prefix** : categorical data
   - **quantity** : numerical data
   - **teacher_number_of_previously_posted_projects** : numerical data
   - **price** : numerical data
   - **sentiment score's of each of the essay** : numerical data
   - **number of words in the title** : numerical data
   - **number of words in the combine essays** : numerical data

   And apply the Logistic regression on these features by finding the best hyper paramter as suggested in step 2 and step 3

6. **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/)

Vectorizer	Model	Hyper parameter	AUC
BOW	Brute	7	0.78
TFIDF	Brute	12	0.79
W2V	Brute	10	0.78
TFIDFW2V	Brute	6	0.78

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-leakage, 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. Logistic Regression

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

Here we are taking 50,000 points randomly due to limited memory

In [51]:

```
data = project_data[10000:60000]
data.head()
```

Out[51]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	p
10000	76755	p258142	279e6808424d777734096694ae19a1c0	Mrs.	MI	
10001	101058	p125183	f3ca3822432094be81b8f70a1feaa8b8	Mrs.	SC	
10002	179555	p130359	990815a9ed48ec7fc3c612cdce661c03	Mrs.	NY	
10003	122148	p027189	b242ae67ce7224850c7cc722613f84b1	Ms.	TX	
10004	136901	p219223	69f536d5eb0bfcf676b373f55a408a4c	Ms.	CA	

5 rows × 26 columns

In [52]:

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

In [53]:

```
data.head()
```

Out[53]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	pr
10000	76755	p258142	279e6808424d777734096694ae19a1c0	Mrs.	MI	
10001	101058	p125183	f3ca3822432094be81b8f70a1feaa8b8	Mrs.	SC	
10002	179555	p130359	990815a9ed48ec7fc3c612cdce661c03	Mrs.	NY	
10003	122148	p027189	b242ae67ce7224850c7cc722613f84b1	Ms.	TX	
10004	136901	p219223	69f536d5eb0bfcf676b373f55a408a4c	Ms.	CA	

5 rows × 25 columns

In [54]:

```
X=data
```

In [55]:

```
# 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=0.30, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.30, stratify=
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [56]:

```
# one hot encoding for "School_state "
vectorizer = CountVectorizer(vocabulary=set(project_data.school_state),lowercase=False, bir
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_school_state_one_hot = vectorizer.transform(X_train['school_state'].values)
X_cv_school_state_one_hot = vectorizer.transform(X_cv['school_state'].values)
X_test_school_state_one_hot = vectorizer.transform(X_test['school_state'].values)

print("After vectorizations")
print("="*50)
print(X_train_school_state_one_hot.shape, y_train.shape)
print(X_cv_school_state_one_hot.shape, y_cv.shape)
print(X_test_school_state_one_hot.shape, y_test.shape)
print("="*50)
print(vectorizer.get_feature_names())
```

```
After vectorizations
==================================================
(24500, 51) (24500,)
(10500, 51) (10500,)
(15000, 51) (15000,)
==================================================
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'I
A', 'ID', 'IL', 'IN', 'KS', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO',
'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM', 'NV', 'NY', 'OH', 'OK', 'O
R', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA', 'WI', 'WV',
'WY']
```

In [57]:

```
# one hot encoding for "project_grade_category"
pattern = "(?u)\\b[\\w-]+\\b"
Project_Grade_Category = CountVectorizer(token_pattern=pattern, lowercase=False, binary=Tru
Project_Grade_Category.fit(X_train['project_grade_category'].values) # fit has to happen or

# we use the fitted CountVectorizer to convert the text to vector
X_train_project_grade_category_one_hot = Project_Grade_Category.transform(X_train['project_
X_cv_project_grade_category_one_hot = Project_Grade_Category.transform(X_cv['project_grade_
X_test_project_grade_category_one_hot = Project_Grade_Category.transform(X_test['project_gr

print("After vectorizations")
print("="*50)
print(X_train_project_grade_category_one_hot.shape, y_train.shape)
print(X_cv_project_grade_category_one_hot.shape, y_cv.shape)
print(X_test_project_grade_category_one_hot.shape, y_test.shape)
print("="*50)
print(Project_Grade_Category.get_feature_names())

type(X_train_project_grade_category_one_hot)
df = pd.DataFrame(X_train_project_grade_category_one_hot.toarray())
df.head()
```

```
After vectorizations
==================================================
(24500, 4) (24500,)
(10500, 4) (10500,)
(15000, 4) (15000,)
==================================================
['3_5', '6_8', '9_12', 'PreK_2']
```

Out[57]:

	0	1	2	3
0	1	0	0	0
1	0	0	0	1
2	0	0	0	1
3	1	0	0	0
4	0	0	0	1

In [58]:

```
# one hot encoding for "clean_categories"

vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, bina
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_categories_one_hot = vectorizer.transform(X_train['clean_categories'].values)
X_cv_clean_categories_one_hot = vectorizer.transform(X_cv['clean_categories'].values)
X_test_clean_categories_one_hot = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print("="*50)
print(X_train_clean_categories_one_hot.shape, y_train.shape)
print(X_cv_clean_categories_one_hot.shape, y_cv.shape)
print(X_test_clean_categories_one_hot.shape, y_test.shape)
print("="*50)
print(vectorizer.get_feature_names())
```

```
After vectorizations
==================================================
(24500, 9) (24500,)
(10500, 9) (10500,)
(15000, 9) (15000,)
==================================================
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning',
'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
```

In [59]:

```
# one hot encoding for "clean_subcategories"

vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False,
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen only on train dat

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_subcategories_one_hot = vectorizer.transform(X_train['clean_subcategories'].v
X_cv_clean_subcategories_one_hot = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_clean_subcategories_one_hot = vectorizer.transform(X_test['clean_subcategories'].val

print("After vectorizations")
print("="*50)
print(X_train_clean_subcategories_one_hot.shape, y_train.shape)
print(X_cv_clean_subcategories_one_hot.shape, y_cv.shape)
print(X_test_clean_subcategories_one_hot.shape, y_test.shape)
print("="*50)
print(vectorizer.get_feature_names())
```

```
After vectorizations
==================================================
(24500, 30) (24500,)
(10500, 30) (10500,)
(15000, 30) (15000,)
==================================================
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement',
'Extracurricular', 'Civics_Government', 'ForeignLanguages', 'NutritionEducat
ion', 'Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingArts', 'Characte
rEducation', 'TeamSports', 'Other', 'College_CareerPrep', 'Music', 'History_
Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL', 'Gym_Fitness',
'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
```

In [60]:

```
# one hot encoding for "teacher_prefix"

vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values.astype("U")) # fit has to happen only on tr

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_prefix_one_hot = vectorizer.transform(X_train['teacher_prefix'].values.asty
X_cv_teacher_prefix_one_hot = vectorizer.transform(X_cv['teacher_prefix'].values.astype("U"
X_test_teacher_prefix_one_hot = vectorizer.transform(X_test['teacher_prefix'].values.astype

print("After vectorizations")
print("="*50)
print(X_train_teacher_prefix_one_hot.shape, y_train.shape)
print(X_cv_teacher_prefix_one_hot.shape, y_cv.shape)
print(X_test_teacher_prefix_one_hot.shape, y_test.shape)
print("="*50)
print(vectorizer.get_feature_names())
```

```
After vectorizations
==================================================
(24500, 6) (24500,)
(10500, 6) (10500,)
(15000, 6) (15000,)
==================================================
['dr', 'mr', 'mrs', 'ms', 'nan', 'teacher']
```

In [61]:

```
# vectorizing numerical features "teacher_number_of_previously_posted_projects"

from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)

X_train_teacher_number_of_previously_posted_projects = normalizer.transform(X_train['teache
X_cv_teacher_number_of_previously_posted_projects = normalizer.transform(X_cv['teacher_numb
X_test_teacher_number_of_previously_posted_projects = normalizer.transform(X_test['teacher_

print("After vectorizations")
print("="*50)
print(X_train_teacher_number_of_previously_posted_projects.shape, y_train.shape)
print(X_cv_teacher_number_of_previously_posted_projects.shape, y_cv.shape)
print(X_test_teacher_number_of_previously_posted_projects.shape, y_test.shape)
```

```
After vectorizations
==================================================
(24500, 1) (24500,)
(10500, 1) (10500,)
(15000, 1) (15000,)
```

In [62]:

```
# vectorizing numerical features "price"

from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['price'].values.reshape(1,-1))

X_train_price = normalizer.transform(X_train['price'].values.reshape(-1,1)) #If (1,-1) is u
X_cv_price = normalizer.transform(X_cv['price'].values.reshape(-1,1))
X_test_price = normalizer.transform(X_test['price'].values.reshape(-1,1))

print("After vectorizations")
print("="*50)
print(X_train_price.shape, y_train.shape)
print(X_cv_price.shape, y_cv.shape)
print(X_test_price.shape, y_test.shape)
```

```
After vectorizations
==================================================
(24500, 1) (24500,)
(10500, 1) (10500,)
(15000, 1) (15000,)
```

In [63]:

```
# vectorizing numerical features "quantity"

from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['quantity'].values.reshape(-1,1))

X_train_quantity = normalizer.transform(X_train['quantity'].values.reshape(-1,1))
X_cv_quantity = normalizer.transform(X_cv['quantity'].values.reshape(-1,1))
X_test_quantity = normalizer.transform(X_test['quantity'].values.reshape(-1,1))

print("After vectorizations")
print("="*50)
print(X_train_quantity.shape, y_train.shape)
print(X_cv_quantity.shape, y_cv.shape)
print(X_test_quantity.shape, y_test.shape)
```

```
After vectorizations
==================================================
(24500, 1) (24500,)
(10500, 1) (10500,)
(15000, 1) (15000,)
```

In [64]:

```
#Normalizing the numerical features: Title word Count

normalizer = Normalizer()
normalizer.fit(X_train['title_word_count'].values.reshape(-1,1))
X_train_title_norm = normalizer.transform(X_train['title_word_count'].values.reshape(-1,1))
X_cv_title_norm = normalizer.transform(X_cv['title_word_count'].values.reshape(-1,1))
X_test_title_norm = normalizer.transform(X_test['title_word_count'].values.reshape(-1,1))
print("After vectorizations")
print(X_train_title_norm.shape, y_train.shape)
print(X_cv_title_norm.shape, y_cv.shape)
print(X_test_title_norm.shape, y_test.shape)
```

```
After vectorizations
(24500, 1) (24500,)
(10500, 1) (10500,)
(15000, 1) (15000,)
```

In [65]:

```
#Normalizing the numerical features: Essay word Count

normalizer = Normalizer()
normalizer.fit(X_train['essay_word_count'].values.reshape(-1,1))
X_train_essay_norm = normalizer.transform(X_train['essay_word_count'].values.reshape(-1,1))
X_cv_essay_norm = normalizer.transform(X_cv['essay_word_count'].values.reshape(-1,1))
X_test_essay_norm = normalizer.transform(X_test['essay_word_count'].values.reshape(-1,1))
print("After vectorizations")
print(X_train_essay_norm.shape, y_train.shape)
print(X_cv_essay_norm.shape, y_cv.shape)
print(X_test_essay_norm.shape, y_test.shape)
```

```
After vectorizations
(24500, 1) (24500,)
(10500, 1) (10500,)
(15000, 1) (15000,)
```

In [66]:

```
#Normalizing the numerical features: Essay Sentiments-Positive

normalizer = Normalizer()
normalizer.fit(X_train['pos'].values.reshape(-1,1))
essay_sent_pos_train = normalizer.transform(X_train['pos'].values.reshape(-1,1))
essay_sent_pos_cv = normalizer.transform(X_cv['pos'].values.reshape(-1,1))
essay_sent_pos_test = normalizer.transform(X_test['pos'].values.reshape(-1,1))
print("After vectorizations")
print(essay_sent_pos_train.shape, y_train.shape)
print(essay_sent_pos_cv.shape, y_cv.shape)
print(essay_sent_pos_test.shape, y_test.shape)
```

```
After vectorizations
(24500, 1) (24500,)
(10500, 1) (10500,)
(15000, 1) (15000,)
```

In [67]:

```
#Normalizing the numerical features: Essay Sentiments-Negative

normalizer = Normalizer()
normalizer.fit(X_train['neg'].values.reshape(-1,1))
essay_sent_neg_train = normalizer.transform(X_train['neg'].values.reshape(-1,1))
essay_sent_neg_cv = normalizer.transform(X_cv['neg'].values.reshape(-1,1))
essay_sent_neg_test = normalizer.transform(X_test['neg'].values.reshape(-1,1))
print("After vectorizations")
print(essay_sent_neg_train.shape, y_train.shape)
print(essay_sent_neg_cv.shape, y_cv.shape)
print(essay_sent_neg_test.shape, y_test.shape)
```

```
After vectorizations
(24500, 1) (24500,)
(10500, 1) (10500,)
(15000, 1) (15000,)
```

In [68]:

```
#Normalizing the numerical features: Essay Sentiments-Neutral

normalizer = Normalizer()
normalizer.fit(X_train['neu'].values.reshape(-1,1))
essay_sent_neu_train = normalizer.transform(X_train['neu'].values.reshape(-1,1))
essay_sent_neu_cv = normalizer.transform(X_cv['neu'].values.reshape(-1,1))
essay_sent_neu_test = normalizer.transform(X_test['neu'].values.reshape(-1,1))
print("After vectorizations")
print(essay_sent_neu_train.shape, y_train.shape)
print(essay_sent_neu_cv.shape, y_cv.shape)
print(essay_sent_neu_test.shape, y_test.shape)
```

```
After vectorizations
(24500, 1) (24500,)
(10500, 1) (10500,)
(15000, 1) (15000,)
```

In [69]:

```
#Normalizing the numerical features: Essay Sentiments-Compound

normalizer = Normalizer()
normalizer.fit(X_train['compound'].values.reshape(-1,1))
essay_sent_comp_train = normalizer.transform(X_train['compound'].values.reshape(-1,1))
essay_sent_comp_cv = normalizer.transform(X_cv['compound'].values.reshape(-1,1))
essay_sent_comp_test = normalizer.transform(X_test['compound'].values.reshape(-1,1))
print("After vectorizations")
print(essay_sent_comp_train.shape, y_train.shape)
print(essay_sent_comp_cv.shape, y_cv.shape)
print(essay_sent_comp_test.shape, y_test.shape)
```

```
After vectorizations
(24500, 1) (24500,)
(10500, 1) (10500,)
(15000, 1) (15000,)
```

2.3 Make Data Model Ready: encoding essay, and project_title

Bag of Words

In [70]:

```
# BOW for essay
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10, ngram_range=(2,2), max_features=5000)
vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_bow_essay = vectorizer.transform(X_train['essay'].values)
X_cv_bow_essay = vectorizer.transform(X_cv['essay'].values)
X_test_bow_essay = vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print("="*50)
print(X_train_bow_essay.shape, y_train.shape)
print(X_cv_bow_essay.shape, y_cv.shape)
print(X_test_bow_essay.shape, y_test.shape)
```

```
After vectorizations
==================================================
(24500, 5000) (24500,)
(10500, 5000) (10500,)
(15000, 5000) (15000,)
```

In [71]:

```
# BOW for "project_title"
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_title'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_bow_title = vectorizer.transform(X_train['project_title'].values)
X_cv_bow_title = vectorizer.transform(X_cv['project_title'].values)
X_test_bow_title = vectorizer.transform(X_test['project_title'].values)

print("After vectorizations")
print("="*50)
print(X_train_bow_title.shape, y_train.shape)
print(X_cv_bow_title.shape, y_cv.shape)
print(X_test_bow_title.shape, y_test.shape)
```

```
After vectorizations
==================================================
(24500, 8502) (24500,)
(10500, 8502) (10500,)
(15000, 8502) (15000,)
```

TF-IDF

In [72]:

```
#TF-idf for "essay"

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10, ngram_range=(2,2), max_features=5000)
vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_tfidf_essay = vectorizer.transform(X_train['essay'].values)
X_cv_tfidf_essay = vectorizer.transform(X_cv['essay'].values)
X_test_tfidf_essay = vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print("="*50)
print(X_train_tfidf_essay.shape, y_train.shape)
print(X_cv_tfidf_essay.shape, y_cv.shape)
print(X_test_tfidf_essay.shape, y_test.shape)
```

```
After vectorizations
==================================================
(24500, 5000) (24500,)
(10500, 5000) (10500,)
(15000, 5000) (15000,)
```

In [73]:

```
#TF-idf for "Project_title"

vectorizer = TfidfVectorizer()
vectorizer.fit(X_train['project_title'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_tfidf_title = vectorizer.transform(X_train['project_title'].values)
X_cv_tfidf_title = vectorizer.transform(X_cv['project_title'].values)
X_test_tfidf_title = vectorizer.transform(X_test['project_title'].values)

print("After vectorizations")
print("="*50)
print(X_train_tfidf_title.shape, y_train.shape)
print(X_cv_tfidf_title.shape, y_cv.shape)
print(X_test_tfidf_title.shape, y_test.shape)
```

```
After vectorizations
==================================================
(24500, 8502) (24500,)
(10500, 8502) (10500,)
(15000, 8502) (15000,)
```

Avg-W2V

In [74]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickl
# 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 [75]:

```
# average Word2Vec for "essay" in training data

X_train_avgw2v_essay = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essay']): # 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_avgw2v_essay.append(vector)

print(len(X_train_avgw2v_essay))
print(len(X_train_avgw2v_essay[1]))
```

```
100%|████████████████████████████████████████████████████████████████
████████████████████████████████████████████| 24500/24500 [00:11<00:0
0, 2164.69it/s]

24500
300
```

In [76]:

```
# average Word2Vec for "essay" in crossvalidation data

X_cv_avgw2v_essay = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['essay']): # 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_cv_avgw2v_essay.append(vector)

print(len(X_cv_avgw2v_essay))
print(len(X_cv_avgw2v_essay[1]))
```

```
100%|████████████████████████████████████████████████████████████████
████████████████████████████████████████████| 10500/10500 [00:04<00:0
0, 2105.08it/s]

10500
300
```

In [77]:

```
# average Word2Vec for "essay" in test data

X_test_avgw2v_essay = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay']): # 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_avgw2v_essay.append(vector)

print(len(X_test_avgw2v_essay))
print(len(X_test_avgw2v_essay[1]))
```

```
100%|████████████████████████████████████████████████████████████████████████| 15000/15000 [00:07<00:0
0, 2081.44it/s]

15000
300
```

In [ ]:

```

```

In [78]:

```
# average Word2Vec for "project_title" in training data

X_train_avgw2v_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['project_title']): # 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_avgw2v_title.append(vector)

print(len(X_train_avgw2v_title))
print(len(X_train_avgw2v_title[1]))
```

```
100%|████████████████████████████████████████████████████████████████████████| 24500/24500 [00:00<00:0
0, 69056.61it/s]

24500
300
```

In [79]:

```
# average Word2Vec for "project_title" in crossvalidation data

X_cv_avgw2v_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['project_title']): # 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_cv_avgw2v_title.append(vector)

print(len(X_cv_avgw2v_title))
print(len(X_cv_avgw2v_title[1]))
```

```
100%|████████████████████████████████████████████████████████████████
████████████████████████████████████████████| 10500/10500 [00:00<00:0
0, 81446.71it/s]

10500
300
```

In [80]:

```
# average Word2Vec for "project_title" in test data

X_test_avgw2v_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['project_title']): # 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_avgw2v_title.append(vector)

print(len(X_test_avgw2v_title))
print(len(X_test_avgw2v_title[1]))
```

```
100%|████████████████████████████████████████████████████████████████
████████████████████████████████████████████| 15000/15000 [00:00<00:0
0, 79413.51it/s]

15000
300
```

TF-IDF Weighted W2V

In [81]:

```
tfidf_model = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
tfidf_model.fit(X_train['essay'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [82]:

```
# TF-IDF weighted Word2Vec for "essay" in training data

X_train_weightw2v_essay = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essay']): # 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((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_train_weightw2v_essay.append(vector)

print(len(X_train_weightw2v_essay))
print(len(X_train_weightw2v_essay[0]))
```

```
100%|██████████████████████████████████████████████████████████████████
██████████████████████████████████████████| 24500/24500 [02:02<00:
00, 200.23it/s]

24500
300
```

In [83]:

```
# TF-IDF weighted Word2Vec for "essay" in cross validation data

X_cv_weightw2v_essay = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['essay']): # 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((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_cv_weightw2v_essay.append(vector)

print(len(X_cv_weightw2v_essay))
print(len(X_cv_weightw2v_essay[0]))
```

```
100%|████████████████████████████████████████████████████████████
████████████████████████████████████████| 10500/10500 [00:51<00:
00, 204.25it/s]

10500
300
```

In [84]:

```
# TF-IDF weighted Word2Vec for "essay" in cross test data

X_test_weightw2v_essay = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay']): # 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((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_weightw2v_essay.append(vector)

print(len(X_test_weightw2v_essay))
print(len(X_test_weightw2v_essay[0]))
```

```
100%|████████████████████████████████████████████████████████████
████████████████████████████████████████| 15000/15000 [01:14<00:
00, 200.36it/s]

15000
300
```

In [ ]:

In [85]:

```
tfidf_model = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
tfidf_model.fit(X_train['project_title'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary_title = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [86]:

```
# TF-IDF weighted Word2Vec for "project_title" in training data

X_train_weightw2v_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['project_title']): # 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((sentenc
            tf_idf = dictionary_title[word]*(sentence.count(word)/len(sentence.split())) #
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_train_weightw2v_title.append(vector)

print(len(X_train_weightw2v_title))
print(len(X_train_weightw2v_title[0]))
```

```
100%|████████████████████████████████████████████████████████████████
████████████████████████████████████| 24500/24500 [00:00<00:0
0, 58230.65it/s]

24500
300
```

In [87]:

```
# TF-IDF weighted Word2Vec for "project_title" in cross validation data

X_cv_weightw2v_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['project_title']): # 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((sentenc
            tf_idf = dictionary_title[word]*(sentence.count(word)/len(sentence.split())) #
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_cv_weightw2v_title.append(vector)

print(len(X_cv_weightw2v_title))
print(len(X_cv_weightw2v_title[0]))
```

```
100%|████████████████████████████████████████████████████████████
████████████████████████████████████████| 10500/10500 [00:00<00:0
0, 56184.03it/s]

10500
300
```

In [88]:

```
# TF-IDF weighted Word2Vec for "project_title" in training data

X_test_weightw2v_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['project_title']): # 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((sentenc
            tf_idf = dictionary_title[word]*(sentence.count(word)/len(sentence.split())) #
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_weightw2v_title.append(vector)

print(len(X_test_weightw2v_title))
print(len(X_test_weightw2v_title[0]))
```

```
100%|████████████████████████████████████████████████████████████
████████████████████████████████████████| 15000/15000 [00:00<00:0
0, 57286.87it/s]

15000
300
```

2.4 Appling Logistic Regression on different kind of featurization as mentioned in the instructions

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

2.4.1 Applying Logistic regression on BOW, SET 1

In [89]:

```
# concatinating all the features
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039

from scipy.sparse import hstack
X_bow_train = hstack((X_train_bow_essay, X_train_bow_title, X_train_school_state_one_hot, X
X_bow_cv = hstack((X_cv_bow_essay, X_cv_bow_title, X_cv_school_state_one_hot, X_cv_project_
X_bow_test = hstack((X_test_bow_essay, X_test_bow_title, X_test_school_state_one_hot, X_tes

print("Final Data matrix")
print("="*50)
print(X_bow_train.shape, y_train.shape)
print(X_bow_cv.shape, y_cv.shape)
print(X_bow_test.shape, y_test.shape)
```

```
Final Data matrix
==================================================
(24500, 13605) (24500,)
(10500, 13605) (10500,)
(15000, 13605) (15000,)
```

Hyper parameter tuning using simple for loop

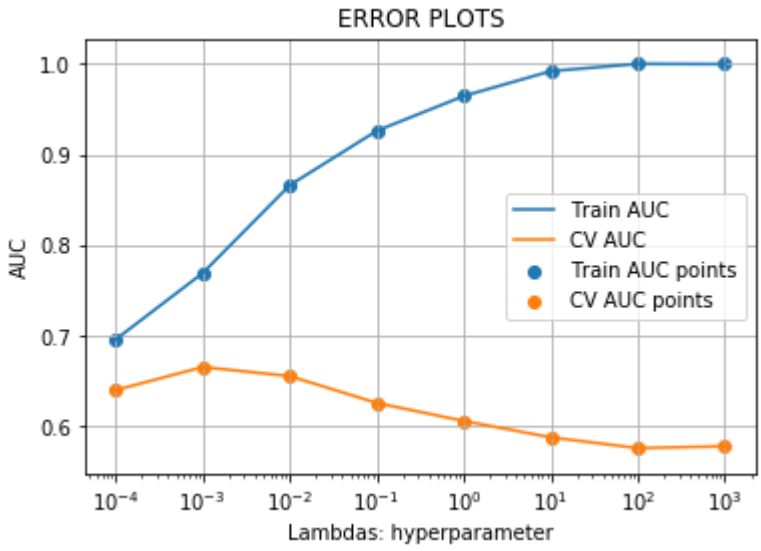
In [90]:

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 4900
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [91]:

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

bow_train_auc = []
bow_cv_auc = []
lambdas = [1000,100,10,1,0.1,0.01,0.001,0.0001]
for i in tqdm(lambdas):
    logistic_clf = LogisticRegression(C=i ,class_weight="balanced")
    logistic_clf.fit(X_bow_train, y_train)

    y_train_bow_pred = batch_predict(logistic_clf, X_bow_train)
    y_cv_bow_pred = batch_predict(logistic_clf, X_bow_cv)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # not the predicted outputs
    bow_train_auc.append(roc_auc_score(y_train,y_train_bow_pred))
    bow_cv_auc.append(roc_auc_score(y_cv, y_cv_bow_pred))

plt.plot(lambdas, bow_train_auc, label='Train AUC')
plt.plot(lambdas, bow_cv_auc, label='CV AUC')

plt.scatter(lambdas, bow_train_auc, label='Train AUC points')
plt.scatter(lambdas, bow_cv_auc, label='CV AUC points')

plt.xscale("log")
plt.legend()
plt.xlabel("Lambdas: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████| 8/8 [01:55<00:00,  6.71s/it]
```



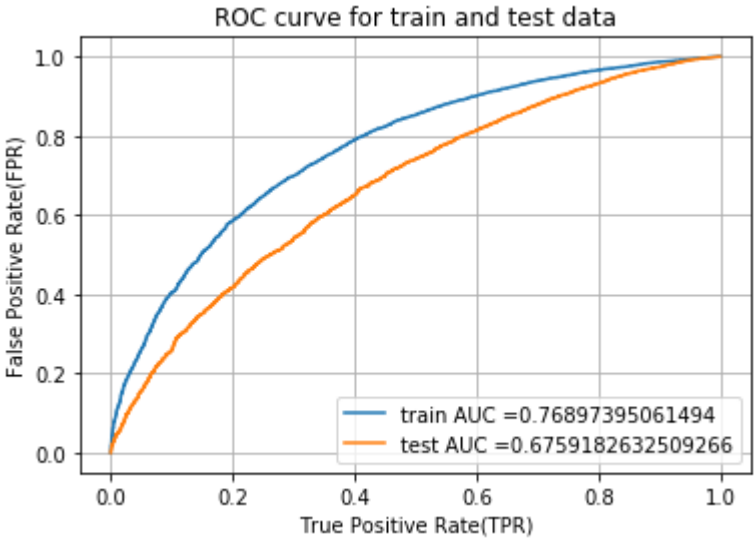
In [92]:

```
best_lambda_bow = 0.001
```

ROC

In [93]:

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

logistic_clf = LogisticRegression(C= best_lambda_bow , class_weight="balanced")
logistic_clf.fit(X_bow_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the p
# not the predicted outputs

y_train_bow_pred = logistic_clf.predict_proba(X_bow_train)[:,1]
y_test_bow_pred = logistic_clf.predict_proba(X_bow_test)[:,1]

train_bow_fpr, train_bow_tpr, train_bow_thresholds = roc_curve(y_train, y_train_bow_pred)
test_bow_fpr, test_bow_tpr, test_bow_thresholds= roc_curve(y_test, y_test_bow_pred)

bow_train_auc = auc(train_bow_fpr, train_bow_tpr)
plt.plot(train_bow_fpr, train_bow_tpr, label="train AUC ="+str(auc(train_bow_fpr, train_bow

bow_test_auc = auc(test_bow_fpr, test_bow_tpr)
plt.plot(test_bow_fpr, test_bow_tpr, label="test AUC ="+str(bow_test_auc))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("ROC curve for train and test data")
plt.grid()
plt.show()
```



Observation

From the above curve it is clearly noticable that test AUC is 10% low than train AUC but both are above 50% therefore it is sensible.

Confusion Matrix

In [94]:

```
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(fpr*(1-tpr))]

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

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

Train Data

In [95]:

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_bow_pred, train_bow_thresholds, train_bow_f
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.449
[[ 1873  1873]
 [ 3071 17683]]
```

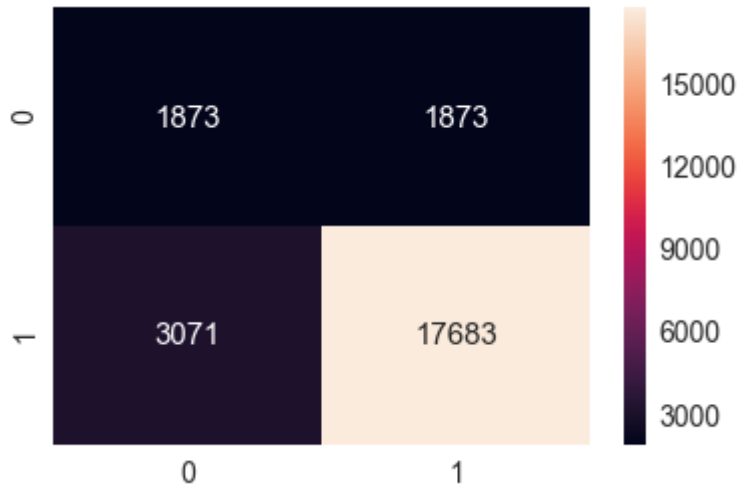
In [96]:

```
cm_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_bow_pred, train_bow_thres
sns.set(font_scale=1.4)#for label size
sns.heatmap(cm_train, annot=True, annot_kws={"size": 15}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.449
```

Out[96]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x250921567b8>
```



Test Data

In [97]:

```
print("test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_bow_pred, test_bow_thresholds, test_bow_fpr,
```

```
test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999995245203888 for threshold 0.482
[[1147 1146]
 [3292 9415]]
```

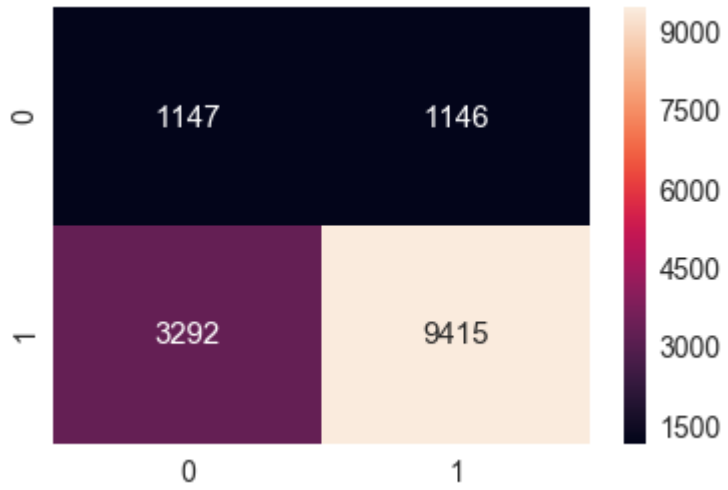
In [98]:

```
cm_test = pd.DataFrame(confusion_matrix(y_test, predict(y_test_bow_pred, test_bow_threshold
sns.set(font_scale=1.4)#for label size
sns.heatmap(cm_test, annot=True, annot_kws={"size": 15}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24999995245203888 for threshold 0.482
```

Out[98]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x250952dfb38>
```



2.4.2 Applying Logistic regression on TFIDF, SET 1

In [99]:

```
# concatinating all the features
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039

from scipy.sparse import hstack
X_tfidf_train = hstack((X_train_tfidf_essay, X_train_tfidf_title, X_train_school_state_one_
X_tfidf_cv = hstack((X_cv_tfidf_essay, X_cv_tfidf_title, X_cv_school_state_one_hot, X_cv_pr
X_tfidf_test = hstack((X_test_tfidf_essay, X_test_tfidf_title, X_test_school_state_one_hot,

print("Final Data matrix")
print("="*50)
print(X_tfidf_train.shape, y_train.shape)
print(X_tfidf_cv.shape, y_cv.shape)
print(X_tfidf_test.shape, y_test.shape)
```

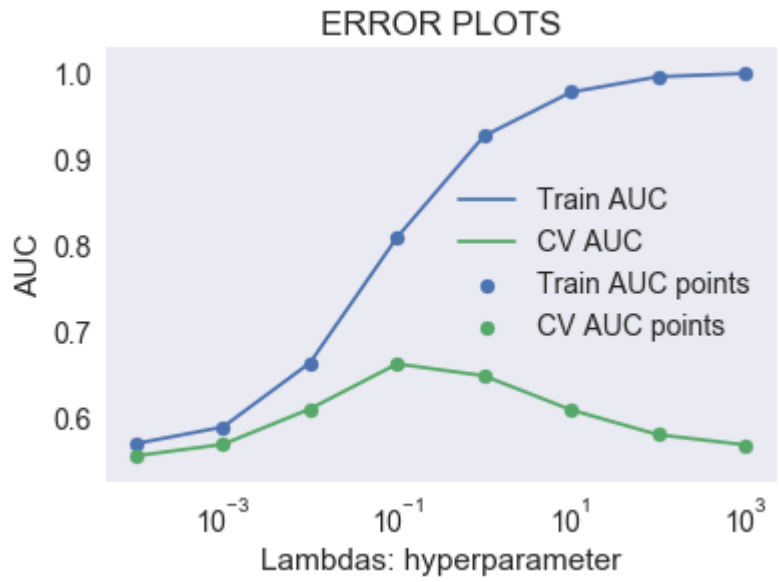
```
Final Data matrix
==================================================
(24500, 13605) (24500,)
(10500, 13605) (10500,)
(15000, 13605) (15000,)
```

Hyper parameter tuning using simple for loop

In [100]:

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

tfidf_train_auc = []
tfidf_cv_auc = []
lambdas = [1000,100,10,1,0.1,0.01,0.001,0.0001]
for i in tqdm(lambdas):
    logistic_clf = LogisticRegression(C=i ,class_weight="balanced")
    logistic_clf.fit(X_tfidf_train, y_train)

    y_train_tfidf_pred = batch_predict(logistic_clf, X_tfidf_train)
    y_cv_tfidf_pred = batch_predict(logistic_clf, X_tfidf_cv)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # not the predicted outputs
    tfidf_train_auc.append(roc_auc_score(y_train,y_train_tfidf_pred))
    tfidf_cv_auc.append(roc_auc_score(y_cv, y_cv_tfidf_pred))

plt.plot(lambdas, tfidf_train_auc, label='Train AUC')
plt.plot(lambdas, tfidf_cv_auc, label='CV AUC')

plt.scatter(lambdas, tfidf_train_auc, label='Train AUC points')
plt.scatter(lambdas, tfidf_cv_auc, label='CV AUC points')

plt.xscale("log")
plt.legend()
plt.xlabel("Lambdas: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████████████████████████████████████████████████████████████████████| 8/8 [01:14<00:00,  5.22s/it]
```



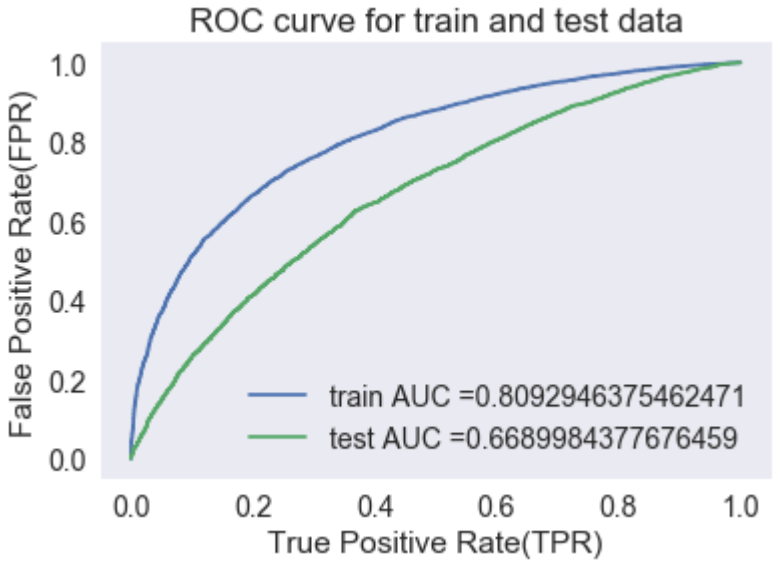
In [101]:

```
best_lambda_tfidf = 0.1
```

ROC

In [102]:

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

logistic_clf = LogisticRegression(C= best_lambda_tfidf , class_weight="balanced")
logistic_clf.fit(X_tfidf_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the p
# not the predicted outputs

y_train_tfidf_pred = logistic_clf.predict_proba(X_tfidf_train)[:,1]
y_test_tfidf_pred = logistic_clf.predict_proba(X_tfidf_test)[:,1]

train_tfidf_fpr, train_tfidf_tpr, train_tfidf_thresholds = roc_curve(y_train, y_train_tfidf
test_tfidf_fpr, test_tfidf_tpr, test_tfidf_thresholds= roc_curve(y_test, y_test_tfidf_pred)

tfidf_train_auc = auc(train_tfidf_fpr, train_tfidf_tpr)
plt.plot(train_tfidf_fpr, train_tfidf_tpr, label="train AUC ="+str(auc(train_tfidf_fpr, tra

tfidf_test_auc = auc(test_tfidf_fpr, test_tfidf_tpr)
plt.plot(test_tfidf_fpr, test_tfidf_tpr, label="test AUC ="+str(tfidf_test_auc))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("ROC curve for train and test data")
plt.grid()
plt.show()
```



Observation

From the above curve it is clearly noticable that test AUC is 15% low than train AUC but both are above 50% therefore it is sensible & also the Train AUC is very High.

Confusion Matrix

Train Data

In [103]:

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_tfidf_pred, train_tfidf_thresholds, train_t
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.432
[[ 1873  1873]
 [ 2486 18268]]
```

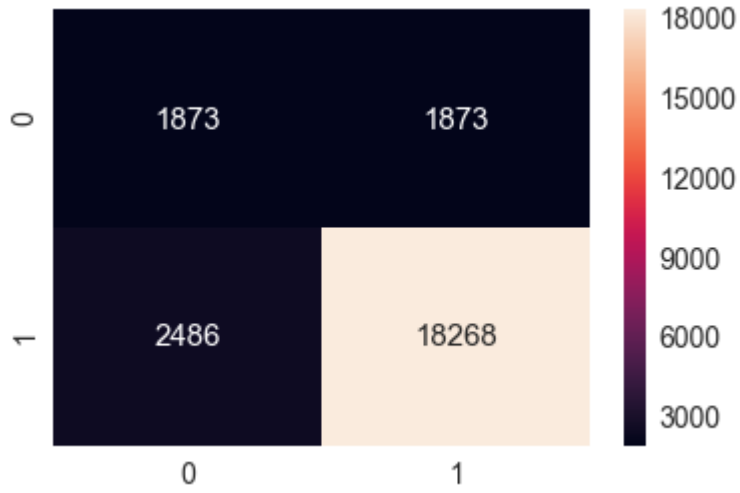
In [104]:

```
cm_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_tfidf_pred, train_tfidf_t
sns.set(font_scale=1.4)#for label size
sns.heatmap(cm_train, annot=True, annot_kws={"size": 15}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.432
```

Out[104]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x250930b59b0>
```



Test Data

In [105]:

```
from sklearn.metrics import confusion_matrix
print("test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_tfidf_pred, test_tfidf_thresholds, test_tfidf
```

```
test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999995245203888 for threshold 0.486
[[1147 1146]
 [3454 9253]]
```

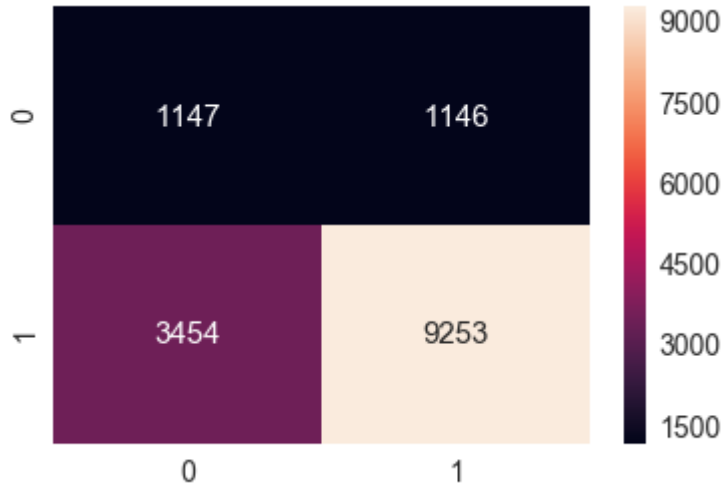
In [106]:

```
cm_test = pd.DataFrame(confusion_matrix(y_test, predict(y_test_tfidf_pred, test_tfidf_thres
sns.set(font_scale=1.4)#for label size
sns.heatmap(cm_test, annot=True, annot_kws={"size": 15}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24999995245203888 for threshold 0.486
```

Out[106]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2509509f198>
```



2.4.3 Applying Logistic regression on AVGW2V, SET 1

In [107]:

```
# concatinating all the features
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039

from scipy.sparse import hstack
X_avgw2v_train = hstack((X_train_avgw2v_essay, X_train_avgw2v_title, X_train_school_state_c
X_avgw2v_cv = hstack((X_cv_avgw2v_essay, X_cv_avgw2v_title, X_cv_school_state_one_hot, X_cv
X_avgw2v_test = hstack((X_test_avgw2v_essay, X_test_avgw2v_title, X_test_school_state_one_h

print("Final Data matrix")
print("="*50)
print(X_avgw2v_train.shape, y_train.shape)
print(X_avgw2v_cv.shape, y_cv.shape)
print(X_avgw2v_test.shape, y_test.shape)
```

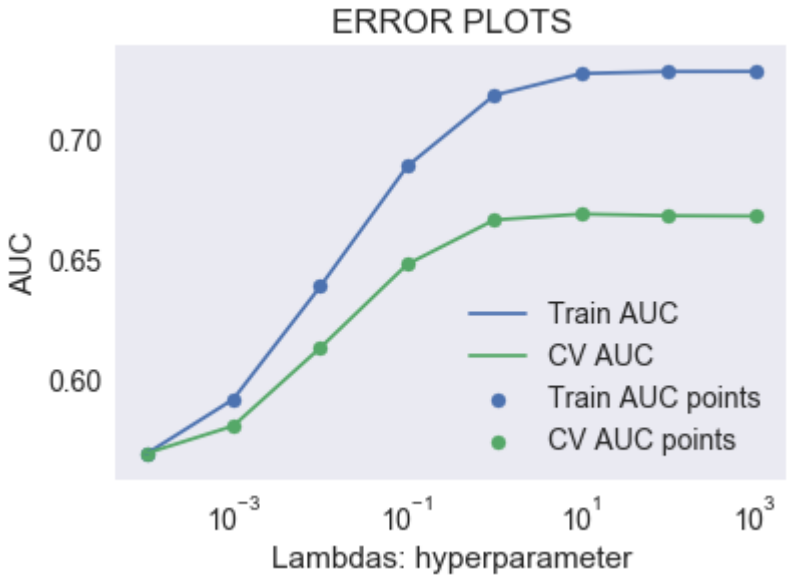
```
Final Data matrix
==================================================
(24500, 703) (24500,)
(10500, 703) (10500,)
(15000, 703) (15000,)
```

Hyper parameter tuning using simple for loop

In [108]:

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

avgw2v_train_auc = []
avgw2v_cv_auc = []
lambdas = [1000,100,10,1,0.1,0.01,0.001,0.0001]
for i in tqdm(lambdas):
    logistic_clf = LogisticRegression(C=i ,class_weight="balanced")
    logistic_clf.fit(X_avgw2v_train, y_train)

    y_train_avgw2v_pred = batch_predict(logistic_clf, X_avgw2v_train)
    y_cv_avgw2v_pred = batch_predict(logistic_clf, X_avgw2v_cv)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # not the predicted outputs
    avgw2v_train_auc.append(roc_auc_score(y_train,y_train_avgw2v_pred))
    avgw2v_cv_auc.append(roc_auc_score(y_cv, y_cv_avgw2v_pred))

plt.plot(lambdas, avgw2v_train_auc, label='Train AUC')
plt.plot(lambdas, avgw2v_cv_auc, label='CV AUC')

plt.scatter(lambdas, avgw2v_train_auc, label='Train AUC points')
plt.scatter(lambdas, avgw2v_cv_auc, label='CV AUC points')

plt.xscale("log")
plt.legend()
plt.xlabel("Lambdas: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████| 8/8 [03:55<00:00, 16.08s/it]
```



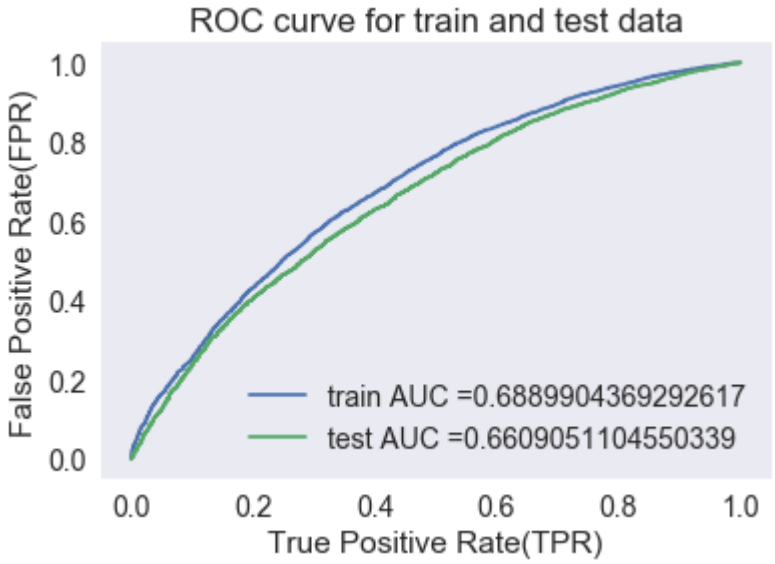
In [109]:

```
best_lambda_avgw2v = 0.1
```

ROC

In [110]:

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

logistic_clf = LogisticRegression(C= best_lambda_avgw2v , class_weight="balanced")
logistic_clf.fit(X_avgw2v_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the p
# not the predicted outputs

y_train_avgw2v_pred = logistic_clf.predict_proba(X_avgw2v_train)[:,1]
y_test_avgw2v_pred = logistic_clf.predict_proba(X_avgw2v_test)[:,1]

train_avgw2v_fpr, train_avgw2v_tpr, train_avgw2v_thresholds = roc_curve(y_train, y_train_av
test_avgw2v_fpr, test_avgw2v_tpr, test_avgw2v_thresholds= roc_curve(y_test, y_test_avgw2v_p

avgw2v_train_auc = auc(train_avgw2v_fpr, train_avgw2v_tpr)
plt.plot(train_avgw2v_fpr, train_avgw2v_tpr, label="train AUC ="+str(auc(train_avgw2v_fpr,

avgw2v_test_auc = auc(test_avgw2v_fpr, test_avgw2v_tpr)
plt.plot(test_avgw2v_fpr, test_avgw2v_tpr, label="test AUC ="+str(avgw2v_test_auc))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("ROC curve for train and test data")
plt.grid()
plt.show()
```



Observation

From the above curve it is clearly noticable that test AUC is 4% low than train AUC but both are above 50% therefore it is sensible.

Confusion Matrix

Train Data

In [111]:

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_avgw2v_pred, train_avgw2v_thresholds, trair
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.458
[[ 1873  1873]
 [ 4908 15846]]
```

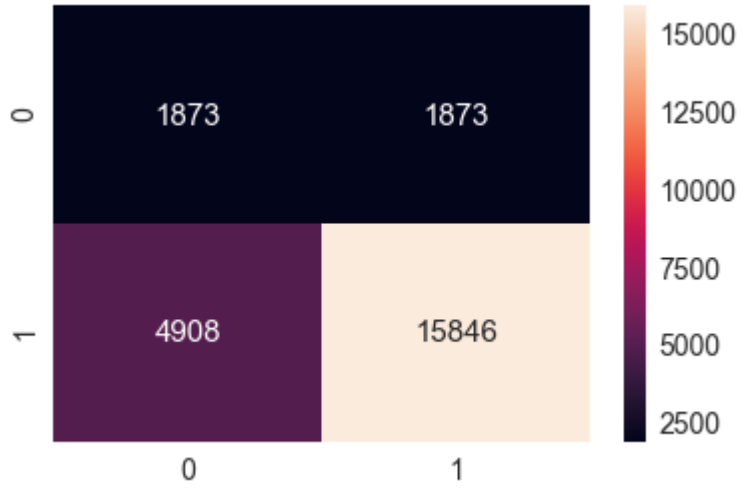
In [112]:

```
cm_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_avgw2v_pred, train_avgw2v
sns.set(font_scale=1.4)#for label size
sns.heatmap(cm_train, annot=True, annot_kws={"size": 15}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.458
```

Out[112]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x250952a8160>
```



Test Data

In [113]:

```
from sklearn.metrics import confusion_matrix
print("test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_avgw2v_pred, test_avgw2v_thresholds, test_avg
```

```
test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999995245203888 for threshold 0.473
[[1147 1146]
 [3554 9153]]
```

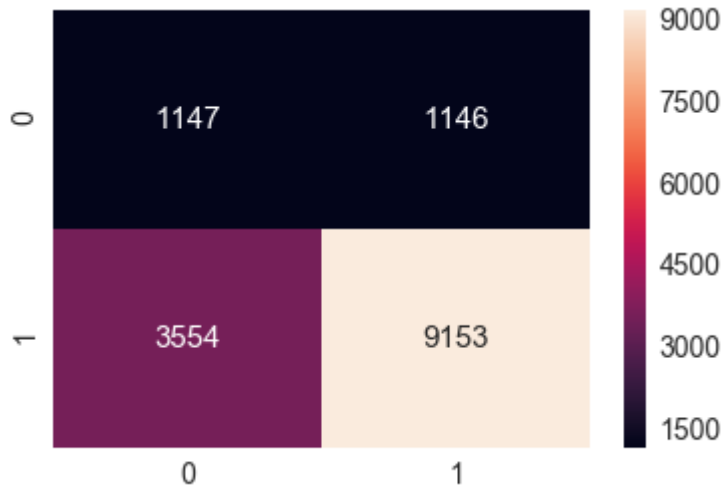
In [114]:

```
cm_test = pd.DataFrame(confusion_matrix(y_test, predict(y_test_avgw2v_pred, test_avgw2v_thr
sns.set(font_scale=1.4)#for label size
sns.heatmap(cm_test, annot=True, annot_kws={"size": 15}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24999995245203888 for threshold 0.473
```

Out[114]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x250c5b543c8>
```



2.4.4 Applying Logistic regression on TFIDF W2V, SET 1

In [115]:

```
# concatinating all the features
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039

from scipy.sparse import hstack
X_weightw2v_train = hstack((X_train_weightw2v_essay, X_train_weightw2v_title, X_train_schoc
X_weightw2v_cv = hstack((X_cv_weightw2v_essay, X_cv_weightw2v_title, X_cv_school_state_one_
X_weightw2v_test = hstack((X_test_weightw2v_essay, X_test_weightw2v_title, X_test_school_st

print("Final Data matrix")
print("="*50)
print(X_weightw2v_train.shape, y_train.shape)
print(X_weightw2v_cv.shape, y_cv.shape)
print(X_weightw2v_test.shape, y_test.shape)
```

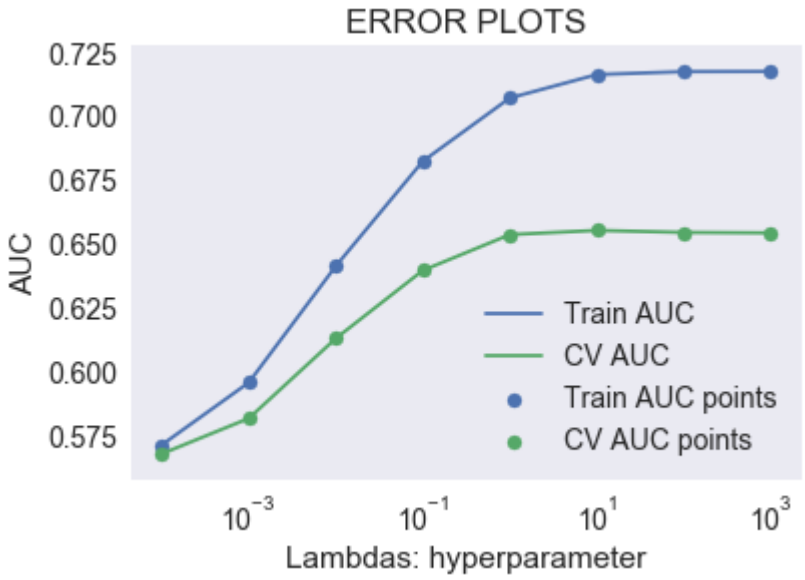
```
Final Data matrix
==================================================
(24500, 703) (24500,)
(10500, 703) (10500,)
(15000, 703) (15000,)
```

Hyper parameter tuning using simple for loop

In [116]:

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

weightw2v_train_auc = []
weightw2v_cv_auc = []
lambdas = [1000,100,10,1,0.1,0.01,0.001,0.0001]
for i in tqdm(lambdas):
    logistic_clf = LogisticRegression(C=i ,class_weight="balanced")
    logistic_clf.fit(X_weightw2v_train, y_train)

    y_train_weightw2v_pred = batch_predict(logistic_clf, X_weightw2v_train)
    y_cv_weightw2v_pred = batch_predict(logistic_clf, X_weightw2v_cv)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # not the predicted outputs
    weightw2v_train_auc.append(roc_auc_score(y_train,y_train_weightw2v_pred))
    weightw2v_cv_auc.append(roc_auc_score(y_cv, y_cv_weightw2v_pred))

plt.plot(lambdas, weightw2v_train_auc, label='Train AUC')
plt.plot(lambdas, weightw2v_cv_auc, label='CV AUC')

plt.scatter(lambdas, weightw2v_train_auc, label='Train AUC points')
plt.scatter(lambdas, weightw2v_cv_auc, label='CV AUC points')

plt.xscale("log")
plt.legend()
plt.xlabel("Lambdas: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████| 8/8 [03:58<00:00, 16.25s/it]
```



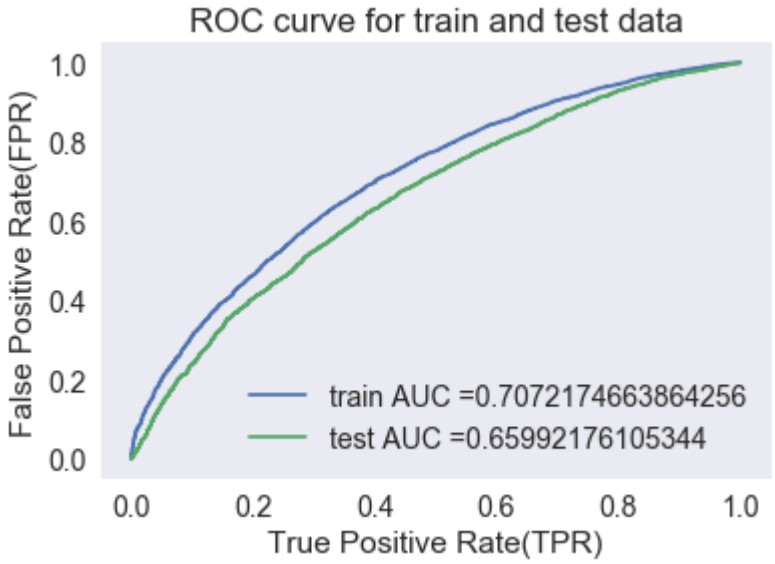
In [117]:

```
best_lambda_weightw2v = 1
```

ROC

In [118]:

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

logistic_clf = LogisticRegression(C= best_lambda_weightw2v , class_weight="balanced")
logistic_clf.fit(X_weightw2v_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the p
# not the predicted outputs

y_train_weightw2v_pred = logistic_clf.predict_proba(X_weightw2v_train)[:,1]
y_test_weightw2v_pred = logistic_clf.predict_proba(X_weightw2v_test)[:,1]

train_weightw2v_fpr, train_weightw2v_tpr, train_weightw2v_thresholds = roc_curve(y_train, y
test_weightw2v_fpr, test_weightw2v_tpr, test_weightw2v_thresholds= roc_curve(y_test, y_test

weightw2v_train_auc = auc(train_weightw2v_fpr, train_weightw2v_tpr)
plt.plot(train_weightw2v_fpr, train_weightw2v_tpr, label="train AUC ="+str(auc(train_weight

weightw2v_test_auc = auc(test_weightw2v_fpr, test_weightw2v_tpr)
plt.plot(test_weightw2v_fpr, test_weightw2v_tpr, label="test AUC ="+str(weightw2v_test_auc)
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("ROC curve for train and test data")
plt.grid()
plt.show()
```



Observation

From the above curve it is clearly noticable that test AUC is 6% low than train AUC but both are above 50% therefore it is sensible.

Confusion Matrix

Train Data

In [119]:

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_weightw2v_pred, train_weightw2v_thresholds,
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.444
[[ 1873  1873]
 [ 4635 16119]]
```

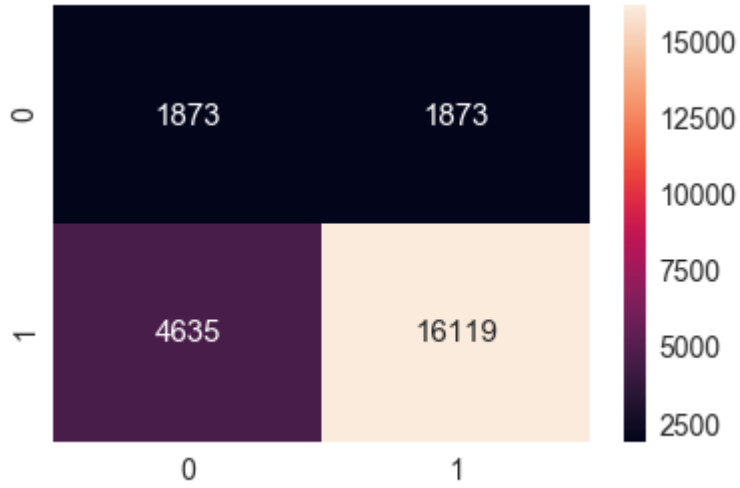
In [120]:

```
cm_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_weightw2v_pred, train_wei
sns.set(font_scale=1.4)#for label size
sns.heatmap(cm_train, annot=True, annot_kws={"size": 15}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.444
```

Out[120]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x250922085f8>
```



Test Data

In [121]:

```
print("test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_weightw2v_pred, test_weightw2v_thresholds, te
```

```
test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999995245203888 for threshold 0.469
[[1147 1146]
 [3567 9140]]
```

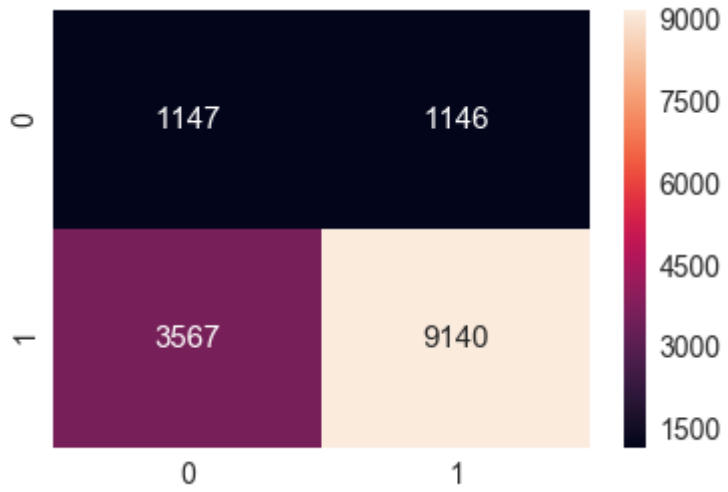
In [122]:

```
cm_test = pd.DataFrame(confusion_matrix(y_test, predict(y_test_weightw2v_pred, test_weightw
sns.set(font_scale=1.4)#for label size
sns.heatmap(cm_test, annot=True, annot_kws={"size": 15}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24999995245203888 for threshold 0.469
```

Out[122]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x25094761160>
```



2.5 Logistic Regression with added Features `Set 5`

In [123]:

```
# concatinating all the features
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039

from scipy.sparse import hstack
X_added_train = hstack((X_train_school_state_one_hot, X_train_project_grade_category_one_hc
X_added_cv = hstack((X_cv_school_state_one_hot, X_cv_project_grade_category_one_hot, X_cv_c
X_added_test = hstack((X_test_school_state_one_hot, X_test_project_grade_category_one_hot,

print("Final Data matrix")
print("="*50)
print(X_added_train.shape, y_train.shape)
print(X_added_cv.shape, y_cv.shape)
print(X_added_test.shape, y_test.shape)
```

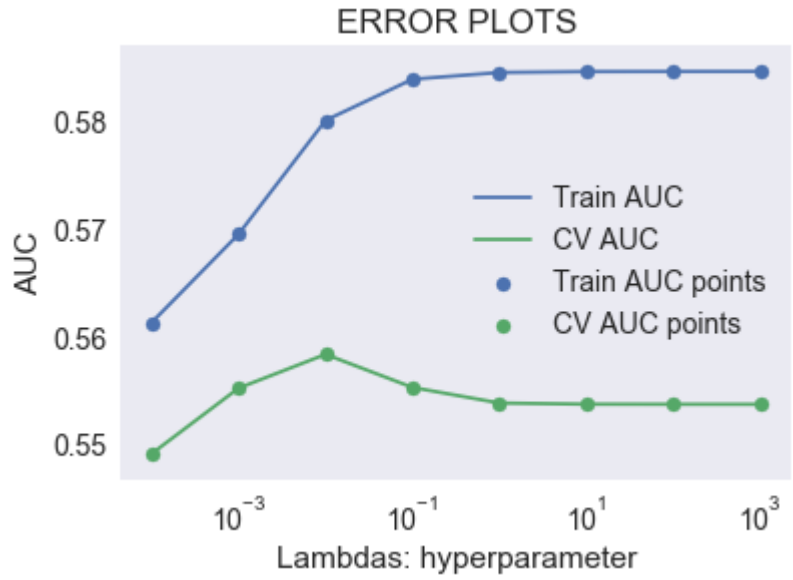
```
Final Data matrix
==================================================
(24500, 109) (24500,)
(10500, 109) (10500,)
(15000, 109) (15000,)
```

Hyper parameter tuning using simple for loop

In [124]:

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

added_train_auc = []
added_cv_auc = []
lambdas = [1000,100,10,1,0.1,0.01,0.001,0.0001]
for i in tqdm(lambdas):
    logistic_clf = LogisticRegression(C=i ,class_weight="balanced")
    logistic_clf.fit(X_added_train, y_train)

    y_train_added_pred = batch_predict(logistic_clf, X_added_train)
    y_cv_added_pred = batch_predict(logistic_clf, X_added_cv)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # not the predicted outputs
    added_train_auc.append(roc_auc_score(y_train,y_train_added_pred))
    added_cv_auc.append(roc_auc_score(y_cv, y_cv_added_pred))

plt.plot(lambdas, added_train_auc, label='Train AUC')
plt.plot(lambdas, added_cv_auc, label='CV AUC')

plt.scatter(lambdas, added_train_auc, label='Train AUC points')
plt.scatter(lambdas, added_cv_auc, label='CV AUC points')

plt.xscale("log")
plt.legend()
plt.xlabel("Lambdas: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|████████████████████████████████████████████████████████████████████████| 8/8 [00:02<0
0:00,  3.37it/s]
```



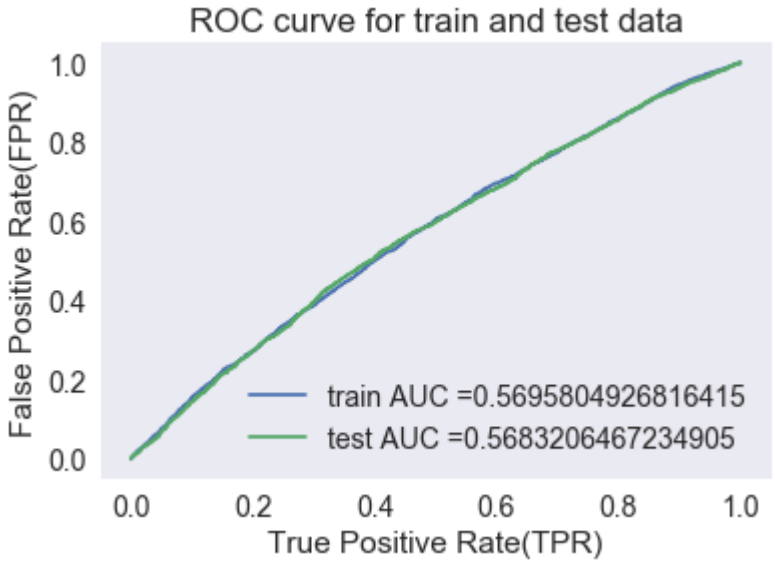
In [125]:

```
best_lambda_added = 0.001
```

ROC

In [126]:

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

logistic_clf = LogisticRegression(C= best_lambda_added , class_weight="balanced")
logistic_clf.fit(X_added_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the p
# not the predicted outputs

y_train_added_pred = logistic_clf.predict_proba(X_added_train)[:,1]
y_test_added_pred = logistic_clf.predict_proba(X_added_test)[:,1]

train_added_fpr, train_added_tpr, train_added_thresholds = roc_curve(y_train, y_train_added
test_added_fpr, test_added_tpr, test_added_thresholds= roc_curve(y_test, y_test_added_pred)

added_train_auc = auc(train_added_fpr, train_added_tpr)
plt.plot(train_added_fpr, train_added_tpr, label="train AUC ="+str(auc(train_added_fpr, tra

added_test_auc = auc(test_added_fpr, test_added_tpr)
plt.plot(test_added_fpr, test_added_tpr, label="test AUC ="+str(added_test_auc))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("ROC curve for train and test data")
plt.grid()
plt.show()
```



Observation

From the above curve it is clearly noticable that test AUC is equivalent to train AUC and both are above 50% therefore Model is sensible.

Confusion Matrix

Train Data

In [127]:

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_added_pred, train_added_thresholds, train_a
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.494
[[ 1873  1873]
 [ 8231 12523]]
```

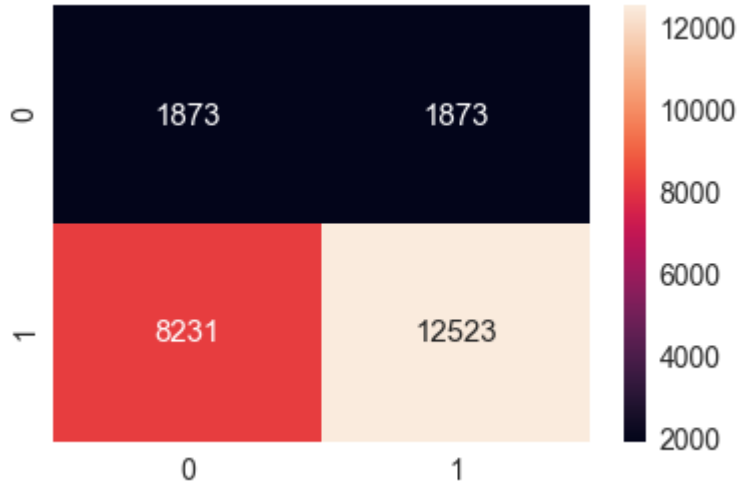
In [128]:

```
cm_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_added_pred, train_added_t
sns.set(font_scale=1.4)#for label size
sns.heatmap(cm_train, annot=True, annot_kws={"size": 15}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.494
```

Out[128]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x25095303c88>
```



Test Data

In [129]:

```
print("test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_added_pred, test_added_thresholds, test_addec
```

```
test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999995245203888 for threshold 0.495
[[1147 1146]
 [5108 7599]]
```

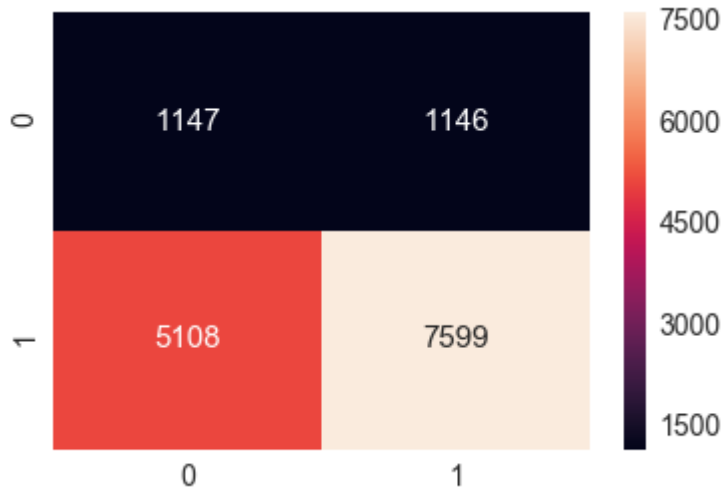
In [130]:

```
cm_test = pd.DataFrame(confusion_matrix(y_test, predict(y_test_added_pred, test_added_thres
sns.set(font_scale=1.4)#for label size
sns.heatmap(cm_test, annot=True, annot_kws={"size": 15}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24999995245203888 for threshold 0.495
```

Out[130]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x25093afa0b8>
```



3. Conclusion

In [131]:

```
# Please compare all your models using Prettytable library
# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Hyper Parameter", "Train AUC", "Test AUC"]

x.add_row(["BOW", "Brute", str(best_lambda_bow), str((bow_train_auc)), str((bow_test_auc))]
x.add_row(["TFIDF", "Brute", str(best_lambda_tfidf), str((tfidf_train_auc)),str((tfidf_test
x.add_row(["AVG-W2V", "Brute", str(best_lambda_avgw2v), str((avgw2v_train_auc)),str((avgw2v
x.add_row(["TFIDFW2V","Brute", str(best_lambda_weightw2v), str((weightw2v_train_auc)),str((
x.add_row(["SET-5 added features","Brute", str(best_lambda_added),str((added_train_auc)), s

print(x)
```

```
+----------------------+-------+----------------+--------------------+-----
---------------+
|      Vectorizer      | Model | Hyper Parameter |     Train AUC      |
Test AUC      |
+----------------------+-------+----------------+--------------------+-----
---------------+
|         BOW          | Brute |      0.001      |  0.76897395061494  | 0.67
59182632509266 |
|        TFIDF         | Brute |       0.1       | 0.8092946375462471 | 0.66
89984377676459 |
|       AVG-W2V        | Brute |       0.1       | 0.6889904369292617 | 0.66
09051104550339 |
|       TFIDFW2V       | Brute |        1        | 0.7072174663864256 |  0.6
5992176105344  |
| SET-5 added features | Brute |      0.001      | 0.5695804926816415 | 0.56
83206467234905 |
+----------------------+-------+----------------+--------------------+-----
---------------+
```

1. Here all the model except set-5 are perfoming reasonably well.
2. Test AUC and Train Auc is high in case of TFIDF with 81% and 66%.
3. There is nothing much difference even after removing text data as the performance even decreses
4. So overall TFIDF performs better with 66% AUC

In [ ]:

In [ ]: