DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

Feature	De
`project_id`	A unique identifier for the proposed project. **Example:**
`project_title`	Title of the project. **E** • `Art Will Make You` • `First Gr`
`project_grade_category`	Grade level of students for which the project is targeted. One of the enumerate • `Grades` • `Gra` • `Gra` • `Grad`
`project_subject_categories`	One or more (comma-separated) subject categories for the projec following enumerated list • `Applied L` • `Care &` • `Health &` • `History &` • `Literacy & L` • `Math &` • `Music & T` • `Specia` • **Ex** • `Music & T` • `Literacy & Language, Math &`
`school_state`	State where school is located (Two-letter U.S. p (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Posta **Exam**
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for th **Ex** • `L` • `Literature & Writing, Social S`
`project_resource_summary`	An explanation of the resources needed for the project. **E** • `My students need hands on literacy materials to sensory`
`project_essay_1`	First applicat
`project_essay_2`	Second applicat
`project_essay_3`	Third applicat
`project_essay_4`	Fourth applicat
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `201 12:43`
`teacher_id`	A unique identifier for the teacher of the proposed project. `bdf8baa8fedef6bfeec7ae4f`

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

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

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

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

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

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

Label	Description
`project_is_approved`	A binary flag indicating whether DonorsChoose approved the project. A value of `0` indicates the

Notes on the Essay Data

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

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

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

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

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

In [1]:

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

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

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

from tqdm import tqdm
import os

from chart_studio import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

1.1 Reading Data

In [2]:

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

In [3]:

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

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

In [4]:

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

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

Out[4]:

	id	description	quantity	price
0	p233245	LC652 - Lakeshore Double-Space Mobile Drying Rack	1	149.00
1	p069063	Bouncy Bands for Desks (Blue support pipes)	3	14.95

1.2 preprocessing of `project_subject_categories`

In [5]:

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

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

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [6]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47
301924/4084039

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

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

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

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

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

1.2.3 preprocessing of `School State`

In [7]:

```
project_data['school_state'].unique()
```

Out[7]:

```
array(['IN', 'FL', 'AZ', 'KY', 'TX', 'CT', 'GA', 'SC', 'NC', 'CA', 'NY',
       'OK', 'MA', 'NV', 'OH', 'PA', 'AL', 'LA', 'VA', 'AR', 'WA', 'WV',
       'ID', 'TN', 'MS', 'CO', 'UT', 'IL', 'MI', 'HI', 'IA', 'RI', 'NJ',
       'MO', 'DE', 'MN', 'ME', 'WY', 'ND', 'OR', 'AK', 'MD', 'WI', 'SD',
       'NE', 'NM', 'DC', 'KS', 'MT', 'NH', 'VT'], dtype=object)
```

In [8]:

```
project_data['school_state'][project_data['school_state'].isnull()==True]
```

Out[8]:

```
Series([], Name: school_state, dtype: object)
```

In [9]:

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

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

In [10]:

```
sorted_school_state_dict.keys()
```

Out[10]:

```
dict_keys(['VT', 'WY', 'ND', 'MT', 'RI', 'SD', 'NE', 'DE', 'AK', 'NH', 'W
V', 'ME', 'HI', 'DC', 'NM', 'KS', 'IA', 'ID', 'AR', 'CO', 'MN', 'OR', 'K
Y', 'MS', 'NV', 'MD', 'CT', 'TN', 'UT', 'AL', 'WI', 'VA', 'AZ', 'NJ', 'O
K', 'WA', 'MA', 'LA', 'OH', 'MO', 'IN', 'PA', 'MI', 'SC', 'GA', 'IL', 'N
C', 'FL', 'NY', 'TX', 'CA'])
```

1.2.4 preprocessing of `Teacher Prefix`

In [11]:

```
project_data.groupby(['teacher_prefix'])['teacher_prefix'].count()
```

Out[11]:

```
teacher_prefix
Dr.            13
Mr.         10648
Mrs.        57269
Ms.         38955
Teacher      2360
Name: teacher_prefix, dtype: int64
```

In [12]:

```
project_data['teacher_prefix'][project_data['teacher_prefix'].isnull()==True]
```

Out[12]:

```
7820     NaN
30368    NaN
57654    NaN
Name: teacher_prefix, dtype: object
```

In [13]:

```
project_data['teacher_prefix'].fillna(project_data['teacher_prefix'].mode()[0],inplace=
True)
```

In [14]:

```
project_data['teacher_prefix'][project_data['teacher_prefix'].isnull()==True]
```

Out[14]:

```
Series([], Name: teacher_prefix, dtype: object)
```

In [15]:

```
project_data['teacher_prefix'].unique()
```

Out[15]:

```
array(['Mrs.', 'Mr.', 'Ms.', 'Teacher', 'Dr.'], dtype=object)
```

In [16]:

```
teacher_prefix = list(project_data['teacher_prefix'].values)

teacher_prefix_list = []
for i in teacher_prefix:
    temp = ""
    temp = i.split('.')
    temp = i.replace('.','')
    teacher_prefix_list.append(temp)

project_data['clean_teacher_prefix'] = teacher_prefix_list
project_data.drop(['teacher_prefix'], 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_teacher_prefix'].values:
    my_counter.update(word.split())

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

In [17]:

```
sorted_teacher_prefix_dict.keys()
```

Out[17]:

```
dict_keys(['Dr', 'Teacher', 'Mr', 'Ms', 'Mrs'])
```

In [18]:

```
project_data.groupby(['clean_teacher_prefix'])['clean_teacher_prefix'].count()
```

Out[18]:

```
clean_teacher_prefix
Dr            13
Mr         10648
Mrs        57272
Ms         38955
Teacher     2360
Name: clean_teacher_prefix, dtype: int64
```

1.2.5 preprocessing of `Project Grade Category`

In [19]:

```
project_data.groupby(['project_grade_category'])['project_grade_category'].count()
```

Out[19]:

```
project_grade_category
Grades 3-5       37137
Grades 6-8       16923
Grades 9-12      10963
Grades PreK-2    44225
Name: project_grade_category, dtype: int64
```

In [20]:

```
project_data['project_grade_category'][project_data['project_grade_category'].isnull()=
=True]
```

Out[20]:

```
Series([], Name: project_grade_category, dtype: object)
```

In [21]:

```
project_grade_category = list(project_data['project_grade_category'].values)

project_grade_category_list = []
for i in project_grade_category:
    temp = ""
    temp = i.split(' ')
    temp = i.replace('Grades ','')
    project_grade_category_list.append(temp)

project_data['clean_project_grade_category'] = project_grade_category_list
project_data.drop(['project_grade_category'], 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_project_grade_category'].values:
    my_counter.update(word.split())

project_grade_category_dict = dict(my_counter)
sorted_project_grade_category_dict = dict(sorted(project_grade_category_dict.items(), k
ey=lambda kv: kv[1]))
```

In [22]:

```
sorted_project_grade_category_dict.keys()
```

Out[22]:

```
dict_keys(['9-12', '6-8', '3-5', 'PreK-2'])
```

In [23]:

```
project_data.groupby(['clean_project_grade_category'])['clean_project_grade_category'].
count()
```

Out[23]:

```
clean_project_grade_category
3-5       37137
6-8       16923
9-12      10963
PreK-2    44225
Name: clean_project_grade_category, dtype: int64
```

1.3 Text preprocessing

In [24]:

```
# 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 [25]:

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

Out[25]:

	Unnamed: 0	id	teacher_id	school_state	project_submitted_da
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	IN	2016-12-05 1
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	FL	2016-10-25 0

1.4.2.3 Using Pretrained Models: TFIDF weighted W2V

In [26]:

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

```
My students are English learners that are working on English as their seco
nd or third languages. We are a melting pot of refugees, immigrants, and n
ative-born Americans bringing the gift of language to our school. \r\n\r\n
We have over 24 languages represented in our English Learner program with 
students at every level of mastery.  We also have over 40 countries repres
ented with the families within our school.  Each student brings a wealth o
f knowledge and experiences to us that open our eyes to new cultures, beli
efs, and respect.\"The limits of your language are the limits of your worl
d.\"-Ludwig Wittgenstein  Our English learner's have a strong support syst
em at home that begs for more resources.  Many times our parents are learn
ing to read and speak English along side of their children.  Sometimes thi
s creates barriers for parents to be able to help their child learn phonet
ics, letter recognition, and other reading skills.\r\n\r\nBy providing the
se dvd's and players, students are able to continue their mastery of the E
nglish language even if no one at home is able to assist.  All families wi
th students within the Level 1 proficiency status, will be a offered to be
a part of this program.  These educational videos will be specially chosen
by the English Learner Teacher and will be sent home regularly to watch.  
The videos are to help the child develop early reading skills.\r\n\r\nPare
nts that do not have access to a dvd player will have the opportunity to c
heck out a dvd player to use for the year.  The plan is to use these video
s and educational dvd's for the years to come for other EL students.\r\nna
nnan
==================================================
The 51 fifth grade students that will cycle through my classroom this year
all love learning, at least most of the time. At our school, 97.3% of the 
students receive free or reduced price lunch. Of the 560 students, 97.3% a
re minority students. \r\nThe school has a vibrant community that loves to
get together and celebrate. Around Halloween there is a whole school parad
e to show off the beautiful costumes that students wear. On Cinco de Mayo 
we put on a big festival with crafts made by the students, dances, and gam
es. At the end of the year the school hosts a carnival to celebrate the ha
rd work put in during the school year, with a dunk tank being the most pop
ular activity.My students will use these five brightly colored Hokki stool
s in place of regular, stationary, 4-legged chairs. As I will only have a 
total of ten in the classroom and not enough for each student to have an i
ndividual one, they will be used in a variety of ways. During independent 
reading time they will be used as special chairs students will each use on
occasion. I will utilize them in place of chairs at my small group tables 
during math and reading times. The rest of the day they will be used by th
e students who need the highest amount of movement in their life in order 
to stay focused on school.\r\n\r\nWhenever asked what the classroom is mis
sing, my students always say more Hokki Stools. They can't get their fill 
of the 5 stools we already have. When the students are sitting in group wi
th me on the Hokki Stools, they are always moving, but at the same time do
ing their work. Anytime the students get to pick where they can sit, the H
okki Stools are the first to be taken. There are always students who head 
over to the kidney table to get one of the stools who are disappointed as 
there are not enough of them. \r\n\r\nWe ask a lot of students to sit for 
7 hours a day. The Hokki stools will be a compromise that allow my student
s to do desk work and move at the same time. These stools will help studen
ts to meet their 60 minutes a day of movement by allowing them to activate
their core muscles for balance while they sit. For many of my students, th
ese chairs will take away the barrier that exists in schools for a child w
ho can't sit still.nannan
==================================================
How do you remember your days of school? Was it in a sterile environment w
ith plain walls, rows of desks, and a teacher in front of the room? A typi
cal day in our room is nothing like that. I work hard to create a warm inv
iting themed room for my students look forward to coming to each day.\r\n
\r\nMy class is made up of 28 wonderfully unique boys and girls of mixed r
```

```
aces in Arkansas.\r\nThey attend a Title I school, which means there is a
high enough percentage of free and reduced-price lunch to qualify. Our sch
ool is an \"open classroom\" concept, which is very unique as there are no
walls separating the classrooms. These 9 and 10 year-old students are very
eager learners; they are like sponges, absorbing all the information and e
xperiences and keep on wanting more.With these resources such as the comfy
red throw pillows and the whimsical nautical hanging decor and the blue fi
sh nets, I will be able to help create the mood in our classroom setting t
o be one of a themed nautical environment. Creating a classroom environmen
t is very important in the success in each and every child's education. Th
e 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 pi
ctures of each child with them, have them developed, and then hung in our
classroom ready for their first day of 4th grade.  This kind gesture will
set the tone before even the first day of school! The nautical thank you c
ards will be used throughout the year by the students as they create thank
you cards to their team groups.\r\n\r\nYour generous donations will help m
e to help make our classroom a fun, inviting, learning environment from da
y one.\r\n\r\nIt costs lost of money out of my own pocket on resources to
get our classroom ready. Please consider helping with this project to make
our new school year a very successful one. Thank you!nannan
====================================================
My kindergarten students have varied disabilities ranging from speech and
language delays, cognitive delays, gross/fine motor delays, to autism. The
y are eager beavers and always strive to work their hardest working past t
heir limitations. \r\n\r\nThe materials we have are the ones I seek out fo
r my students. I teach in a Title I school where most of the students rece
ive free or reduced price lunch.  Despite their disabilities and limitatio
ns, 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 groov
e and move as you were in a meeting? This is how my kids feel all the tim
e. The want to be able to move as they learn or so they say.Wobble chairs
are the answer and I love then because they develop their core, which enha
nces gross motor and in Turn fine motor skills. \r\nThey also want to lear
n through games, my kids don't want to sit and do worksheets. They want to
learn to count by jumping and playing. Physical engagement is the key to o
ur 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
====================================================
The mediocre teacher tells. The good teacher explains. The superior teache
r demonstrates. The great teacher inspires. -William A. Ward\r\n\r\nMy sch
ool has 803 students which is makeup is 97.6% African-American, making up
the largest segment of the student body. A typical school in Dallas is mad
e 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 child
ren from rich backgrounds or neighborhoods. As an educator I am inspiring
minds of young children and we focus not only on academics but one smart,
effective, efficient, and disciplined students with good character.In our
classroom we can utilize the Bluetooth for swift transitions during class.
I use a speaker which doesn't amplify the sound enough to receive the mess
age. Due to the volume of my speaker my students can't hear videos or book
s clearly and it isn't making the lessons as meaningful. But with the blue
tooth speaker my students will be able to hear and I can stop, pause and r
eplay it at any time.\r\nThe cart will allow me to have more room for stor
age of things that are needed for the day and has an extra part to it I ca
n use.  The table top chart has all of the letter, words and pictures for
students to learn about different letters and it is more accessible.nannan
====================================================
```

In [27]:

```
# 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 [28]:

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

```
My kindergarten students have varied disabilities ranging from speech and
language delays, cognitive delays, gross/fine motor delays, to autism. The
y are eager beavers and always strive to work their hardest working past t
heir limitations. \r\n\r\nThe materials we have are the ones I seek out fo
r my students. I teach in a Title I school where most of the students rece
ive free or reduced price lunch.  Despite their disabilities and limitatio
ns, 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 groov
e and move as you were in a meeting? This is how my kids feel all the tim
e. The want to be able to move as they learn or so they say.Wobble chairs
are the answer and I love then because they develop their core, which enha
nces gross motor and in Turn fine motor skills. \r\nThey also want to lear
n through games, my kids do not want to sit and do worksheets. They want t
o 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 happe
n. My students will forget they are doing work and just have the fun a 6 y
ear old deserves.nannan
==================================================
```

In [29]:

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

```
My kindergarten students have varied disabilities ranging from speech and
language delays, cognitive delays, gross/fine motor delays, to autism. The
y are eager beavers and always strive to work their hardest working past t
heir limitations.     The materials we have are the ones I seek out for my
students. I teach in a Title I school where most of the students receive f
ree or reduced price lunch.  Despite their disabilities and limitations, m
y 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 were in a meeting? This is how my kids feel all the time. The
want to be able to move as they learn or so they say.Wobble chairs are the
answer and I love then because they develop their core, which enhances gro
ss motor and in Turn fine motor skills.   They also want to learn through
games, my kids do not want to sit and do worksheets. They want to learn to
count by jumping and playing. Physical engagement is the key to our succes
s. The number toss and color and shape mats can make that happen. My stude
nts will forget they are doing work and just have the fun a 6 year old des
erves.nannan
```

In [30]:

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

```
My kindergarten students have varied disabilities ranging from speech and
language delays cognitive delays gross fine motor delays to autism They ar
e eager beavers and always strive to work their hardest working past their
limitations The materials we have are the ones I seek out for my students
I teach in a Title I school where most of the students receive free or red
uced price lunch Despite their disabilities and limitations my students lo
ve 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 w
ere in a meeting This is how my kids feel all the time The want to be able
to move as they learn or so they say Wobble chairs are the answer and I lo
ve 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 no
t 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 [31]:

```
# 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'r
e", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 't
hey', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
at'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
d', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as'
, 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through'
, 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'ov
er', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
y', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too'
, 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
w', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'migh
tn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
asn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [32]:

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

```
100%|██████████| 109248/109248 [02:04<00:00, 874.40it/s]
```

In [33]:

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

Out[33]:

```
'my kindergarten students varied disabilities ranging speech language dela
ys cognitive delays gross fine motor delays autism they eager beavers alwa
ys strive work hardest working past limitations the materials ones i seek
students i teach title i school students receive free reduced price lunch
despite disabilities limitations students love coming school come eager le
arn explore have ever felt like ants pants needed groove move meeting this
kids feel time the want able move learn say wobble chairs answer i love de
velop core enhances gross motor turn fine motor skills they also want lear
n games kids not want sit worksheets they want learn count jumping playing
physical engagement key success the number toss color shape mats make happ
en my students forget work fun 6 year old deserves nannan'
```

In [34]:

```
project_data['preprocessed_essays'] = preprocessed_essays
```

1.4 Preprocessing of `project_title`

In [35]:

```
project_data['project_title'][2000:2010]
```

Out[35]:

```
2000                    Steady Stools for Active Learning
2001                                   Classroom Supplies
2002    Kindergarten Students Deserve Quality  Books a...
2003                                Listen to Understand!
2004                             iPads to iGnite Learning
2005                                  Tablets For Learning
2006                                              Go P.E.!
2007                                  Making Learning Fun!
2008    Empowerment Through Silk Screen Designed Tee S...
2009                                  Let's Play Together!
Name: project_title, dtype: object
```

In [36]:

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

```
100%|██████████| 109248/109248 [00:06<00:00, 15643.92it/s]
```

In [37]:

```
preprocessed_titles[2000:2010]
```

Out[37]:

```
['steady stools active learning',
 'classroom supplies',
 'kindergarten students deserve quality books vibrant rug',
 'listen understand',
 'ipads ignite learning',
 'tablets for learning',
 'go p e',
 'making learning fun',
 'empowerment through silk screen designed tee shirts',
 'let play together']
```

In [38]:

```
project_data['preprocessed_titles'] = preprocessed_titles
project_data.drop(['project_title'], axis=1, inplace=True)
```

1.5 Preparing data for models

In [39]:

```
project_data.columns
```

Out[39]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'school_state',
       'project_submitted_datetime', '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', 'clean_teacher_prefix',
       'clean_project_grade_category', 'essay', 'preprocessed_essays',
       'preprocessed_titles'],
      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
```

In [40]:

```
project_data = project_data.drop(['Unnamed: 0','project_submitted_datetime','project_essay_1','project_essay_2','project_essay_3','project_essay_4','project_resource_summary','essay','teacher_id'], axis = 1)
```

In [41]:

```
project_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 109248 entries, 0 to 109247
Data columns (total 10 columns):
 #   Column                                        Non-Null Count   Dtype 
---  ------                                        --------------   ----- 
 0   id                                            109248 non-null  object
 1   school_state                                  109248 non-null  object
 2   teacher_number_of_previously_posted_projects  109248 non-null  int64 
 3   project_is_approved                           109248 non-null  int64 
 4   clean_categories                              109248 non-null  object
 5   clean_subcategories                           109248 non-null  object
 6   clean_teacher_prefix                          109248 non-null  object
 7   clean_project_grade_category                  109248 non-null  object
 8   preprocessed_essays                           109248 non-null  object
 9   preprocessed_titles                           109248 non-null  object
dtypes: int64(2), object(8)
memory usage: 8.3+ MB
```

In [ ]:

```

```

1.6 Merging Numerical data in Resources to project_data

In [42]:

```
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')
```

Assignment 7: SVM

1. **[Task-1] Apply Support Vector Machines(SGDClassifier with hinge loss: Linear SVM) on these feature sets**

    - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (BOW)
    - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
    - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
    - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)

2. **The hyper paramter tuning (best alpha in range [10^-4 to 10^4], and the best penalty among 'l1', 'l2')**

    - 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.

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

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

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

4. **[Task-2] Apply the Support Vector Machines on these features by finding the best hyper paramter as suggested in step 2 and step 3** (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

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

    - Consider these set of features Set 5 : (https://seaborn.pydata.org/generated/seaborn.heatmap.html)
        - **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

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

- **Apply** (https://seaborn.pydata.org/generated/seaborn.heatmap.html)**TruncatedSVD (http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html) on TfidfVectorizer (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html) of essay text, choose the number of components (`n_components`) using elbow method (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/pca-code-example-using-non-visualization/)** : numerical data

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

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-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. Support Vector Machines

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

In [43]:

```
project_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 109248 entries, 0 to 109247
Data columns (total 12 columns):
 #   Column                                        Non-Null Count   Dtype 
---  ------                                        --------------   ----- 
 0   id                                            109248 non-null  object
 1   school_state                                  109248 non-null  object
 2   teacher_number_of_previously_posted_projects  109248 non-null  int64 
 3   project_is_approved                           109248 non-null  int64 
 4   clean_categories                              109248 non-null  object
 5   clean_subcategories                           109248 non-null  object
 6   clean_teacher_prefix                          109248 non-null  object
 7   clean_project_grade_category                  109248 non-null  object
 8   preprocessed_essays                           109248 non-null  object
 9   preprocessed_titles                           109248 non-null  object
 10  price                                         109248 non-null  float6
4
 11  quantity                                      109248 non-null  int64 
dtypes: float64(1), int64(3), object(8)
memory usage: 10.8+ MB
```

In [44]:

```
project_data = project_data[:50000]
```

In [45]:

```
y = project_data['project_is_approved'].values
X = project_data.drop(['project_is_approved'], axis=1)
X.head(1)
```

Out[45]:

	id	school_state	teacher_number_of_previously_posted_projects	clean_categories	c
0	p253737	IN	0	Literacy_Language	

In [46]:

```
# train test split

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(project_data, project_data['project
_is_approved'], test_size=0.33, stratify = project_data['project_is_approved'])
```

In [47]:

```
print("Split ratio")
print('-'*50)
print('Train dataset:',len(X_train)/len(project_data)*100,'%\n','size:',len(X_train))

print('Test dataset:',len(X_test)/len(project_data)*100,'%\n','size:',len(X_test))
```

```
Split ratio
--------------------------------------------------
Train dataset: 67.0 %
 size: 33500
Test dataset: 33.0 %
 size: 16500
```

In [48]:

```
#Features
X_train.drop(['project_is_approved'], axis=1, inplace=True)

X_test.drop(['project_is_approved'], axis=1, inplace=True)
```

In [ ]:

```

```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [49]:

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

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

2.1 Numerical features

1. teacher_number_of_previously_posted_projects
2. price
3. quantity

2.2.1.1 Teacher number of previously posted projects

In [50]:

```
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_TPPP_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_p
rojects'].values.reshape(1,-1))
X_test_TPPP_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_pro
jects'].values.reshape(1,-1))

print("After vectorizations")
print(X_train_TPPP_norm.shape, y_train.shape)
print(X_test_TPPP_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(1, 33500) (33500,)
(1, 16500) (16500,)
==========================================================================
==========================
```

In [51]:

```
print("Transpose of teacher number of previously posted projects")

X_train_TPPP_norm = X_train_TPPP_norm.transpose()
X_test_TPPP_norm = X_test_TPPP_norm.transpose()

print("After transpose")
print(X_train_TPPP_norm.shape, y_train.shape)
print(X_test_TPPP_norm.shape, y_test.shape)
print("="*100)
```

```
Transpose of teacher number of previously posted projects
After transpose
(33500, 1) (33500,)
(16500, 1) (16500,)
==========================================================================
==========================
```

2.2.1.2 price

In [52]:

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

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

```
After vectorizations
(1, 33500) (33500,)
(1, 16500) (16500,)
==========================================================================
==========================
```

In [53]:

```
print("Transpose of price")

X_train_price_norm = X_train_price_norm.transpose()
X_test_price_norm = X_test_price_norm.transpose()


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

```
Transpose of price
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
==========================================================================
==========================
```

2.2.1.3 quantity

In [54]:

```
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_norm = normalizer.transform(X_train['quantity'].values.reshape(1,-1))
X_test_quantity_norm = normalizer.transform(X_test['quantity'].values.reshape(1,-1))

print("After vectorizations")
print(X_train_quantity_norm.shape, y_train.shape)
print(X_test_quantity_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(1, 33500) (33500,)
(1, 16500) (16500,)
==========================================================================
==========================
```

In [55]:

```
print("Transpose of Quantity")

X_train_quantity_norm = X_train_quantity_norm.transpose()
X_test_quantity_norm = X_test_quantity_norm.transpose()


print("After vectorizations")
print(X_train_quantity_norm.shape, y_train.shape)
print(X_test_quantity_norm.shape, y_test.shape)
print("="*100)
```

```
Transpose of Quantity
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
==========================================================================
==========================
```

2.2.2 Categorical Data

Categorical Features for vectorization

1. Clean Categories
2. Clean Sub Categories
3. School State
4. Teacher Prefix
5. Project grade category

2.2.2.1 Clean Categories

In [56]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True)
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_CC_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_test_CC_ohe = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_CC_ohe.shape, y_train.shape)
print(X_test_CC_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(33500, 9) (33500,)
(16500, 9) (16500,)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
====================================================================================================
```

2.2.2.2 Clean Sub Categories

In [57]:

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_CSC_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
X_test_CSC_ohe = vectorizer.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_CSC_ohe.shape, y_train.shape)
print(X_test_CSC_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(33500, 30) (33500,)
(16500, 30) (16500,)
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvemen
t', 'Extracurricular', 'Civics_Government', 'ForeignLanguages', 'Nutrition
Education', 'Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingArts',
'CharacterEducation', 'TeamSports', 'Other', 'College_CareerPrep', 'Musi
c', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL',
'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'A
ppliedSciences', 'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Lit
eracy']
==========================================================================
==========================
```

2.2.2.3 School State

In [58]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_school_state_dict.keys()), lowercase=False, binary=True)
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_state_ohe = vectorizer.transform(X_train['school_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(33500, 51) (33500,)
(16500, 51) (16500,)
['VT', 'WY', 'ND', 'MT', 'RI', 'SD', 'NE', 'DE', 'AK', 'NH', 'WV', 'ME',
'HI', 'DC', 'NM', 'KS', 'IA', 'ID', 'AR', 'CO', 'MN', 'OR', 'KY', 'MS', 'N
V', 'MD', 'CT', 'TN', 'UT', 'AL', 'WI', 'VA', 'AZ', 'NJ', 'OK', 'WA', 'M
A', 'LA', 'OH', 'MO', 'IN', 'PA', 'MI', 'SC', 'GA', 'IL', 'NC', 'FL', 'N
Y', 'TX', 'CA']
==========================================================================
==========================
```

2.2.2.4 Teacher prefix

In [59]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_teacher_prefix_dict.keys()), lowercase=False, binary=True)
vectorizer.fit(X_train['clean_teacher_prefix'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer.transform(X_train['clean_teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['clean_teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(33500, 5) (33500,)
(16500, 5) (16500,)
['Dr', 'Teacher', 'Mr', 'Ms', 'Mrs']
====================================================================================================
```

2.2.2.5 Project Grade category

In [60]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_project_grade_category_dict.keys()), lowercase=False, binary=True)
vectorizer.fit(X_train['clean_project_grade_category'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer.transform(X_train['clean_project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['clean_project_grade_category'].values)

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(33500, 4) (33500,)
(16500, 4) (16500,)
['9-12', '6-8', '3-5', 'PreK-2']
====================================================================================================
```

In [ ]:

```
```

2.3 Make Data Model Ready: encoding eassay, and project_title

In [61]:

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

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

Ecoding Essay and Project title

```
2.3.1 BOW
2.3.2 TFIDF
2.3.3 AVG W2V
2.3.4 TFIDF W2V
```

2.3.1 BOW Essays and Title

2.3.1.1 BOW Essay

In [62]:

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

print("="*100)


vectorizer = CountVectorizer(min_df=10,ngram_range=(2,2), max_features=5000)
vectorizer.fit(X_train['preprocessed_essays'].values) # fit has to happen only on train
data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['preprocessed_essays'].values)
X_test_essay_bow = vectorizer.transform(X_test['preprocessed_essays'].values)

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

```
(33500, 11) (33500,)
(16500, 11) (16500,)
==========================================================================
==========================
After vectorizations
(33500, 5000) (33500,)
(16500, 5000) (16500,)
==========================================================================
==========================
```

2.3.1.2 BOW Title

In [63]:

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

print("="*100)


vectorizer = CountVectorizer(min_df=10,ngram_range=(2,2), max_features=5000)
vectorizer.fit(X_train['preprocessed_titles'].values) # fit has to happen only on train
data

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_bow = vectorizer.transform(X_train['preprocessed_titles'].values)
X_test_title_bow = vectorizer.transform(X_test['preprocessed_titles'].values)

print("After vectorizations")
print(X_train_title_bow.shape, y_train.shape)
print(X_test_title_bow.shape, y_test.shape)
print("="*100)
```

```
(33500, 11) (33500,)
(16500, 11) (16500,)
==========================================================================
==========================
After vectorizations
(33500, 1067) (33500,)
(16500, 1067) (16500,)
==========================================================================
==========================
```

2.3.2 TF IDF Essay and Title

2.3.2.1 TF IDF Essay

In [64]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

print("="*100)


vectorizer = TfidfVectorizer(min_df=10,ngram_range=(2,2), max_features=5000)
vectorizer.fit(X_train['preprocessed_essays'].values) # fit has to happen only on train
data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer.transform(X_train['preprocessed_essays'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['preprocessed_essays'].values)

print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
```

```
(33500, 11) (33500,)
(16500, 11) (16500,)
==========================================================================
==========================
After vectorizations
(33500, 5000) (33500,)
(16500, 5000) (16500,)
==========================================================================
==========================
```

2.3.2.2 TF IDF Title

In [65]:

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

print("="*100)


vectorizer = TfidfVectorizer(min_df=10,ngram_range=(2,2), max_features=5000)
vectorizer.fit(X_train['preprocessed_titles'].values) # fit has to happen only on train
data

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_tfidf = vectorizer.transform(X_train['preprocessed_titles'].values)
X_test_title_tfidf = vectorizer.transform(X_test['preprocessed_titles'].values)

print("After vectorizations")
print(X_train_title_tfidf.shape, y_train.shape)
print(X_test_title_tfidf.shape, y_test.shape)
print("="*100)
```

```
(33500, 11) (33500,)
(16500, 11) (16500,)
==========================================================================
==========================
After vectorizations
(33500, 1067) (33500,)
(16500, 1067) (16500,)
==========================================================================
==========================
```

2.3.3 AVG W2V Essay and Title

2.3.3.1 AVG W2V Essay

In [66]:

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

In [67]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['preprocessed_essays'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train.append(vector)

print(len(avg_w2v_vectors_train))
print(len(avg_w2v_vectors_train[0]))
print(avg_w2v_vectors_train[0])
```

```
100%|██████████| 33500/33500 [00:22<00:00, 1514.90it/s]
```

```
33500
300
[ 7.47763441e-03  3.24815648e-02  1.40538226e-02 -9.45587796e-02
  8.26109140e-03  5.52618828e-02 -3.19990763e+00  1.35346043e-01
  1.30357527e-03 -6.87951935e-02  2.12703548e-03 -1.80045460e-02
  7.74376344e-02 -1.03384885e-01 -1.56060538e-03 -8.20022333e-02
  5.13309882e-02 -3.17261527e-02  1.07267728e-01  4.22213656e-02
  3.97784496e-02  2.50448495e-02  1.82753871e-03 -3.85450140e-02
 -7.09182634e-03 -7.71782898e-02  6.62676989e-02 -5.76049484e-02
 -1.31208349e-01 -2.86166849e-02 -1.33508480e-01 -1.35610619e-01
  3.43358237e-02  6.30589204e-02 -6.11299095e-02 -6.98158172e-03
 -5.04001505e-02 -9.34818957e-02  8.36861290e-04 -8.70693011e-03
 -9.87632581e-02  7.39766118e-02 -6.96004956e-02 -1.59581677e-01
  2.95462366e-03 -9.69948398e-02  7.44700527e-02 -2.55813344e-02
 -2.94010890e-02 -1.48634137e-01 -8.67572892e-02 -6.02233011e-02
  7.42663753e-02 -8.38146968e-02 -2.28815215e-02 -7.55495570e-02
  6.68076344e-02 -3.26049032e-02 -1.25272753e-01  4.82043651e-02
  1.26285341e-02 -2.60918925e-02  1.32493860e-01 -2.95161075e-02
 -8.15603548e-02  1.45979116e-01  1.65919677e-03 -7.67239581e-02
  6.62840108e-02 -1.16111290e-01 -1.24917621e-01  1.31873849e-02
  3.92304072e-02 -1.37828790e-01 -9.14113333e-03 -1.89197677e-01
  4.55871774e-02  1.32157753e-02  7.16432151e-02 -4.32377677e-02
  5.52988090e-02 -4.69275516e-01  3.46757430e-02 -8.24901978e-02
 -6.36553323e-02 -1.71697613e-02  1.15173770e-01 -9.50612581e-03
  1.67827189e-01  1.90121301e-02  1.36003942e-02 -2.32871656e-02
 -5.88358441e-02  5.51833151e-02 -4.77140839e-02 -9.71375699e-02
 -2.21744710e+00  1.98780772e-02  1.04256345e-01  1.41811796e-01
 -1.19295533e-01  3.04478677e-02  1.31429065e-01 -5.62528484e-02
  4.94337516e-02 -7.31553441e-03  4.48490702e-02 -1.91491070e-01
  1.44711000e-02  6.92469667e-02 -8.08244598e-02  2.64333763e-03
  3.88492796e-03  1.62331908e-01  4.92222419e-02  1.86030434e-01
 -1.42495172e-01 -6.59080753e-03  9.41347000e-02  4.37618903e-02
  6.10910968e-03 -4.43203695e-02  6.15733120e-02 -1.10175405e-01
  8.07123581e-02 -1.05179516e-01  7.09793785e-03 -1.29995118e-02
 -3.26615097e-02  1.50006083e-01  1.98302322e-02 -2.80532366e-03
 -1.84740376e-02 -9.11210720e-02  1.15101226e-02 -8.64197075e-02
  1.73317611e-01  5.72118484e-03  1.31911315e-01  2.96129572e-01
  5.45111237e-02  4.21949866e-02  1.21001048e-01 -6.45147172e-02
  9.72915290e-03 -7.62875892e-02  9.75091920e-02 -9.53130774e-02
  1.97365624e-01 -2.46851613e-03  1.98355249e-02 -5.61593344e-02
  2.18345763e-03 -7.47370028e-02  2.20775280e-02  3.63157957e-02
  4.71806452e-04 -3.68837935e-02 -9.95260463e-02 -1.94454731e-03
  4.27380172e-02 -1.71743032e-02 -7.48803495e-02 -8.10384215e-02
  8.03563613e-03  1.93307634e-02 -7.75595109e-02  2.92398280e-02
  1.14427946e-01 -4.24344011e-02 -7.66153000e-02 -3.87259226e-02
 -9.04667323e-02 -1.14410104e-01 -3.47527913e-02  4.41513518e-02
 -1.36118710e-02 -6.08619355e-03 -1.87996732e-01 -1.02525752e-01
  7.60559785e-02  2.88450847e-01 -5.43069720e-02 -4.55620968e-03
 -1.08671561e-01 -8.71264462e-02 -2.33445505e-02 -6.20754484e-02
  6.31808333e-02 -1.20747204e-02 -1.65148387e-02 -8.80050817e-02
 -7.49520882e-02 -2.69448538e-02  4.05031505e-03 -5.54862108e-02
 -6.45621989e-02 -1.32626688e-02  8.27026226e-02 -3.74155373e-02
  2.19842197e-01 -2.11410022e-02 -7.13869634e-02  8.93889563e-02
 -1.70088427e-01  3.15302333e-02  5.53346366e-02 -8.74090237e-02
  2.22148098e-01  1.07026989e-02  1.06834448e-01 -7.11345699e-03
 -2.32427753e-02 -2.56819706e-01 -1.21545673e-01  1.37719753e-02
 -7.52784602e-02 -9.92675077e-02 -1.83084043e-02  3.77961172e-02
 -1.01257366e-01 -1.23097391e-01 -2.22091742e-02 -3.66320129e-02
 -1.96290022e+00  4.54361591e-02  1.83930989e-02 -3.72521387e-02
 -6.45725054e-03 -1.28588914e-01  7.20212688e-03  4.89670495e-02
 -2.57513763e-03 -4.50568398e-02 -8.41537161e-02 -3.27721659e-03
 -1.27135075e-02 -4.69595172e-02 -1.89231806e-02  6.11892333e-02
 -4.01589969e-02  7.88378960e-02 -2.83993467e-01  6.85979570e-02
 -5.83950828e-02  1.78270892e-02 -5.48652151e-02  1.56225624e-03
 -4.51048132e-02 -2.47181912e-02  6.12058237e-02  1.05221098e-01
  4.11198441e-02 -5.13640000e-02  1.88190570e-01 -1.07557656e-02
  1.16473129e-01 -4.95246946e-02  6.65514183e-02 -9.45026576e-02
 -5.88624301e-03 -3.60195194e-02 -1.39037118e-03  6.78447527e-02
  4.24824022e-02 -1.35618770e-01 -1.32675877e-01  3.36318387e-03
  4.37980667e-02  7.62908581e-03 -1.71380559e-02  3.60395624e-02
 -1.45714396e-01  7.78462312e-02 -9.47608495e-03  9.10159742e-02
  1.30110501e-01 -5.17754839e-03  1.51858065e-03  1.28979180e-01
  1.64525882e-01  7.67301935e-03  5.39518430e-02  1.29777880e-01
 -3.81845591e-03  1.65934201e-01  1.58837000e-02 -5.13541538e-02
  1.94370215e-02  1.07474086e-02 -3.69860247e-02 -1.35969230e-02
  1.54686559e-03 -1.42438467e-01  8.69545387e-02  8.08119538e-02
 -5.45541946e-02  1.30318422e-01  7.05853733e-02  8.15273752e-02]
```

In [68]:

```
avg_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this lis
t
for sentence in tqdm(X_test['preprocessed_essays'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test.append(vector)
```

```
100%|██████████| 16500/16500 [00:11<00:00, 1499.90it/s]
```

2.3.3.2 AVG W2V Title

In [69]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_train_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['preprocessed_titles'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train_title.append(vector)

print(len(avg_w2v_vectors_train_title))
print(len(avg_w2v_vectors_train_title[0]))
print(avg_w2v_vectors_train_title[0])
```

```
100%|██████████| 33500/33500 [00:01<00:00, 26458.35it/s]

33500
300
[-0.08471     0.22536     0.21678333 -0.23669667  0.33664    -0.07625433
 -1.60026667  0.32971267  0.23029333 -0.02039033  0.173716    0.034518
  0.21767667  0.11648133  0.06687333  0.13098433  0.08590133 -0.093217
  0.045353    0.187353   -0.41111667  0.37221333  0.22340133  0.068277
 -0.27576667 -0.03305833 -0.02978833 -0.133595   -0.24863167 -0.39198667
  0.27897667  0.14929033  0.01517467  0.23193333  0.30121333  0.19303333
  0.13563333  0.40875     0.09765667  0.44435    -0.13309667  0.12115167
 -0.07216267 -0.06676333  0.09222     0.10183     0.14598633  0.09836667
 -0.05561567 -0.17097333 -0.06539     0.0580193   0.07858667  0.132976
  0.35891333 -0.12058333 -0.10583333  0.14658333 -0.01038867 -0.14973333
  0.11678     0.46119333  0.02231333  0.012653   -0.29664167  0.35494
 -0.14260833 -0.06184667  0.105125   -0.11459167 -0.1892      0.071876
 -0.46166667  0.07811     0.04702677 -0.006208   -0.46525667 -0.13521667
 -0.26539933  0.03773667  0.10129333  0.17953667 -0.00211233 -0.30017667
  0.17390933  0.35584333  0.02420133  0.08721667 -0.09218333  0.1977
  0.15044    -0.3157     -0.201058    0.25024867  0.01761    -0.45349667
 -1.24755    -0.03962667 -0.418741    0.13082333 -0.16304667  0.07218667
  0.145454   -0.03793667  0.289654    0.254692   -0.16547633  0.31212667
 -0.00974433  0.33728533 -0.33599     0.27017    -0.09151867  0.05367667
  0.22523667 -0.13453    -0.29108267 -0.186118   -0.00644333  0.028694
  0.04327533 -0.354       0.00891333  0.0099419   0.17248067 -0.24584333
 -0.28773667 -0.28044667 -0.01171167  0.03323667 -0.07895667  0.18470333
 -0.072258   -0.148896    0.10101333  0.44221667 -0.18192     0.21175867
  0.47252     0.446899    0.14833433 -0.23325667  0.34939333  0.10530067
 -0.25859    -0.101343    0.01175     0.13639833 -0.20696333 -0.025448
  0.15789133  0.00240333  0.06240867 -0.039043   -0.09942433  0.01122667
  0.31226    -0.02308667 -0.37348667 -0.22424617 -0.33428     0.067451
 -0.10417333 -0.14476767 -0.00685    -0.26247467  0.19548     0.38911333
  0.227828    0.00272667 -0.091792    0.101875   -0.01928667 -0.21171157
  0.0412      0.30981    -0.02059607  0.11883333 -0.26563333 -0.43233333
  0.069239    0.14124267  0.02700833  0.33695667 -0.25077667 -0.18936133
  0.38566667  0.07349     0.09341    -0.01930333 -0.072927    0.33328833
  0.120765   -0.20534933  0.15338567 -0.31723333  0.106092   -0.101453
 -0.32994927 -0.03499833  0.11981667 -0.04109667 -0.30489667  0.21086867
 -0.13378333 -0.08452633 -0.10673667 -0.43787     0.03051667  0.009855
  0.044996    0.05584567  0.08181833 -0.28031333 -0.27906    -0.10686
 -0.00701767 -0.04484867 -0.025605   -0.45800333  0.12345     0.14449887
  0.60565667  0.0617293  -1.75370667  0.40073667 -0.06518733  0.25106267
 -0.15816233 -0.09395    -0.03524833  0.16677     0.21961733  0.237842
  0.29154667 -0.10614433  0.01225    -0.33268    -0.22451667 -0.35642667
 -0.04558333 -0.01863967  0.09676    -0.23740133 -0.06754     0.34189
 -0.17708333  0.21758133 -0.10498097 -0.022166    0.48960333  0.28737667
  0.11809433  0.27723667 -0.02232833 -0.06673667  0.066072    0.22057667
 -0.00465333 -0.02299467  0.12548533 -0.0137      0.14137667 -0.11609667
 -0.0759     -0.06179333  0.07765367  0.25878333  0.4152     -0.22042967
 -0.159195    0.22892     0.19315667  0.21978633  0.33014633 -0.020303
  0.18615167  0.20399667  0.04230433  0.30095667  0.34288333  0.28589
  0.27658333  0.32418667  0.00619867  0.449411   -0.04997267  0.13120067
  0.26932267  0.39833833 -0.06307533  0.00667467  0.23844467  0.28473267
  0.05516333  0.26536167  0.02581933 -0.46285333 -0.32274    -0.15855333]
```

In [70]:

```
avg_w2v_vectors_test_title = []; # the avg-w2v for each sentence/review is stored in th
is list
for sentence in tqdm(X_test['preprocessed_titles'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test_title.append(vector)
```

```
100%|██████████| 16500/16500 [00:00<00:00, 25604.50it/s]
```

2.3.4 TF IDF W2V Essay and Title

2.3.4.1 TF IDF W2V Essay

In [71]:

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

In [72]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['preprocessed_essays'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_train.append(vector)

print(len(tfidf_w2v_vectors_train))
print(len(tfidf_w2v_vectors_train[0]))
```

```
100%|██████████| 33500/33500 [02:34<00:00, 216.53it/s]

33500
300
```

In [73]:

```
tfidf_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['preprocessed_essays'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_test.append(vector)
```

```
100%|██████████| 16500/16500 [01:15<00:00, 219.75it/s]
```

2.3.4.2 TF IDF W2V Title

In [74]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['preprocessed_titles'].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 [75]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_train_title = []; # the avg-w2v for each sentence/review is stored in
this list
for sentence in tqdm(X_train['preprocessed_titles'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_train_title.append(vector)

print(len(tfidf_w2v_vectors_train_title))
print(len(tfidf_w2v_vectors_train_title[0]))
```

```
100%|██████████| 33500/33500 [00:02<00:00, 14189.87it/s]

33500
300
```

In [76]:

```
tfidf_w2v_vectors_test_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['preprocessed_titles'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_test_title.append(vector)
```

```
100%|██████████| 16500/16500 [00:01<00:00, 15020.16it/s]
```

Concatinating all the features

1. SET 1 BOW

In [77]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_BOW = hstack((X_train_essay_bow, X_train_title_bow, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_CSC_ohe, X_train_CC_ohe, X_train_price_norm, X_train_quantity_norm, X_train_TPPP_norm)).tocsr()
X_te_BOW = hstack((X_test_essay_bow, X_test_title_bow, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_CSC_ohe, X_test_CC_ohe, X_test_price_norm, X_test_quantity_norm, X_test_TPPP_norm)).tocsr()

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

```
Final Data matrix
(33500, 6169) (33500,)
(16500, 6169) (16500,)
==========================================================================
==========================
```

2. SET 2 TF IDF

In [78]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_TFIDF = hstack((X_train_essay_tfidf, X_train_title_tfidf, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_CSC_ohe, X_train_CC_ohe, X_train_price_norm, X_train_quantity_norm, X_train_TPPP_norm)).tocsr()
X_te_TFIDF = hstack((X_test_essay_tfidf, X_test_title_tfidf, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_CSC_ohe, X_test_CC_ohe, X_test_price_norm, X_test_quantity_norm, X_test_TPPP_norm)).tocsr()

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

```
Final Data matrix
(33500, 6169) (33500,)
(16500, 6169) (16500,)
====================================================================================================
```

3. SET 3 AVG W2V

In [79]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_AVG_W2V = hstack((avg_w2v_vectors_train, avg_w2v_vectors_train_title, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_CSC_ohe, X_train_CC_ohe, X_train_price_norm, X_train_quantity_norm, X_train_TPPP_norm)).tocsr()
X_te_AVG_W2V = hstack((avg_w2v_vectors_test, avg_w2v_vectors_test_title, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_CSC_ohe, X_test_CC_ohe, X_test_price_norm, X_test_quantity_norm, X_test_TPPP_norm)).tocsr()

print("Final Data matrix")
print(X_tr_AVG_W2V.shape, y_train.shape)
print(X_te_AVG_W2V.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(33500, 702) (33500,)
(16500, 702) (16500,)
====================================================================================================
```

4. SET 4 TF IDF W2V

```
In [80]:
```

```python
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_TFIDF_W2V = hstack((tfidf_w2v_vectors_train, tfidf_w2v_vectors_train_title, X_trai
n_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_CSC_ohe, X_train_CC_ohe, X
_train_price_norm, X_train_quantity_norm, X_train_TPPP_norm)).tocsr()
X_te_TFIDF_W2V = hstack((tfidf_w2v_vectors_test, tfidf_w2v_vectors_test_title, X_test_s
tate_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_CSC_ohe, X_test_CC_ohe, X_test_p
rice_norm, X_test_quantity_norm, X_test_TPPP_norm)).tocsr()

print("Final Data matrix")
print(X_tr_TFIDF_W2V.shape, y_train.shape)
print(X_te_TFIDF_W2V.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(33500, 702) (33500,)
(16500, 702) (16500,)
==========================================================================
==========================
```

Computing sentiment scores

In [81]:

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

# import nltk
nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()
for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest
students with the biggest enthusiasm \
for learning my students learn in many different ways using all of our senses and multi
ple intelligences i use a wide range\
of techniques to help all my students succeed students in my class come from a variety
 of different backgrounds which makes\
for wonderful sharing of experiences and cultures including native americans our school
is a caring community of successful \
learners which can be seen through collaborative student project based learning in and
 out of the classroom kindergarteners \
in my class love to work with hands on materials and have many different opportunities
 to practice a skill before it is\
mastered having the social skills to work cooperatively with friends is a crucial aspec
t of the kindergarten curriculum\
montana is the perfect place to learn about agriculture and nutrition my students love
 to role play in our pretend kitchen\
in the early childhood classroom i have had several kids ask me can we try cooking with
real food i will take their idea \
and create common core cooking lessons where we learn important math and writing concep
ts while cooking delicious healthy \
food for snack time my students will have a grounded appreciation for the work that wen
t into making the food and knowledge \
of where the ingredients came from as well as how it is healthy for their bodies this p
roject would expand our learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to make hom
emade applesauce make our own bread \
and mix up healthy plants from our classroom garden in the spring we will also create o
ur own cookbooks to be printed and \
shared with families students will gain math and literature skills as well as a life lo
ng enjoyment for healthy cooking \
nannan'
ss = sid.polarity_scores(for_sentiment)
for k in ss:
    print('{0}: {1}, '.format(k, ss[k]), end='')

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

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

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

In [ ]:

```

```

2.4 Appling Support Vector Machines on different kind of featurization as mentioned in the instructions

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

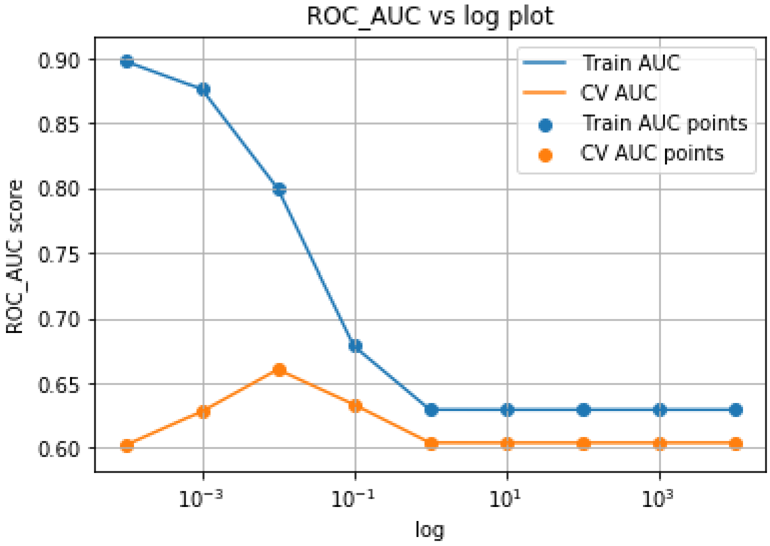
In [82]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

2.4.1 Applying SVM on BOW, SET 1

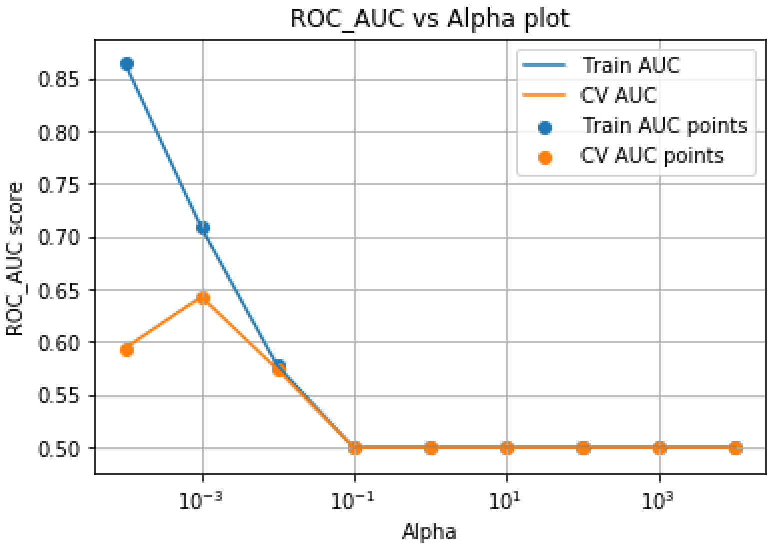
In [83]:

```
## By using "l2" Regulrizer
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
#from sklearn.datasets import *
from sklearn import linear_model
from sklearn.linear_model import SGDClassifier
from sklearn import svm
# hyperparameter tuning with l2 reg
parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
sd = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced')
classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=True)
classifier.fit(X_tr_BOW, y_train)

train_auc = classifier.cv_results_['mean_train_score']
cv_auc= classifier.cv_results_['mean_test_score']

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("log")
plt.xscale('log')
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs log plot")
plt.grid()
plt.show()
```



In [84]:

```
#By using  "l1" Regularization

# hyperparameter tuning with l1 reg
#parameters = {'alpha':[0.007,0.009,0.01,0.05,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,1.
2,1.4,1.6,1.8,2,2.2,2.4,2.6,2.8,3,3,3.5,4,4.5,5]}
import warnings
warnings.filterwarnings("ignore")

parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**
4]}
sd = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced')
classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=T
rue)
classifier.fit(X_tr_BOW, y_train)
train_auc = classifier.cv_results_['mean_train_score']
cv_auc= classifier.cv_results_['mean_test_score']

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
plt.plot(parameters['alpha'], cv_auc, label='CV AUC')

plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("Alpha")
plt.xscale('log')
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs Alpha plot")
plt.grid()
plt.show()
```



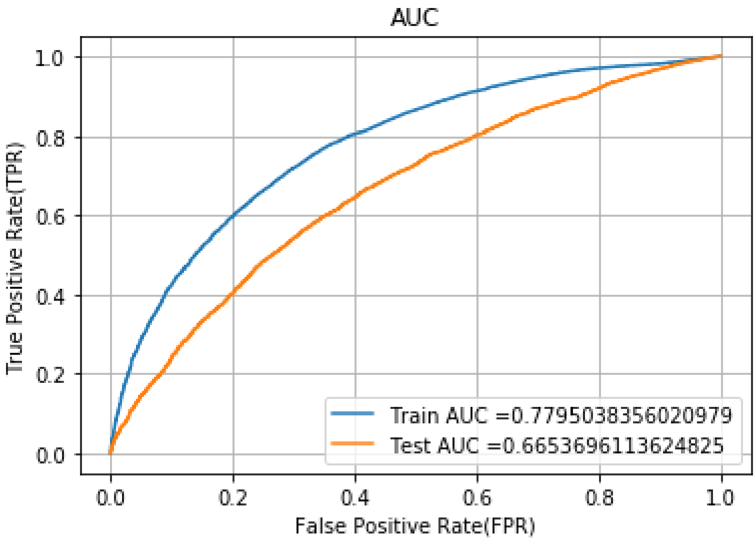
Observation:

l2 regularizatin works better than l1 and best alpha is 0.01.

Fitting Model to Hyper parameter curve

In [85]:

```
# https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l2', alpha = 0.01, class_weight = 'balanced')
Classifier_bow.fit(X_tr_BOW ,y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positiveclass
#https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGDClassifier.decision_function
y_train_pred = Classifier_bow.decision_function(X_tr_BOW)
y_test_pred = Classifier_bow.decision_function(X_te_BOW)
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



In [86]:

```
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.rou
nd(t,3))
    return t

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

Confusion Matrix

In [87]:

```
#our objective here is to make auc the maximum
#so we find  the best threshold that will give the least fpr
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 0.5048257962093863 for threshold -0.094
Train confusion matrix
[[ 3652  1516]
 [ 8092 20240]]
```

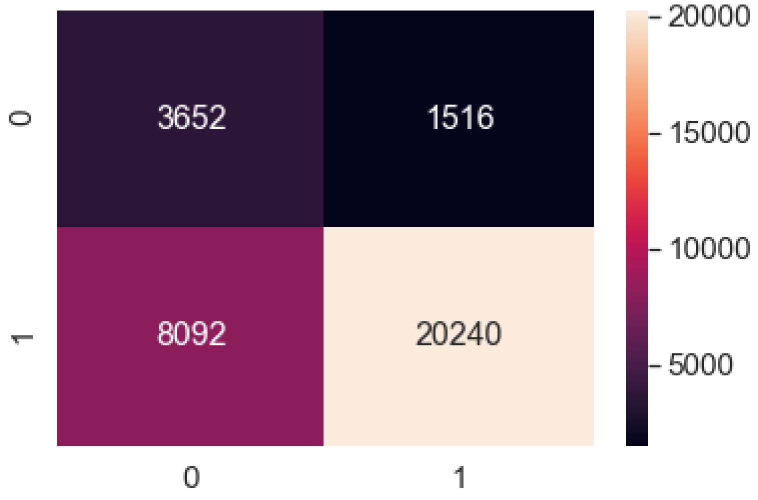
In [88]:

```
#plotting confusion matrix using seaborn's heatmap
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

print("Train data confusion matrix")

confusion_matrix_df_train = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(
y_train_pred, best_t)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(confusion_matrix_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Train data confusion matrix
```

Out[88]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ed23fa8048>
```



In [89]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
Test confusion matrix
[[1411 1135]
 [4371 9583]]
```

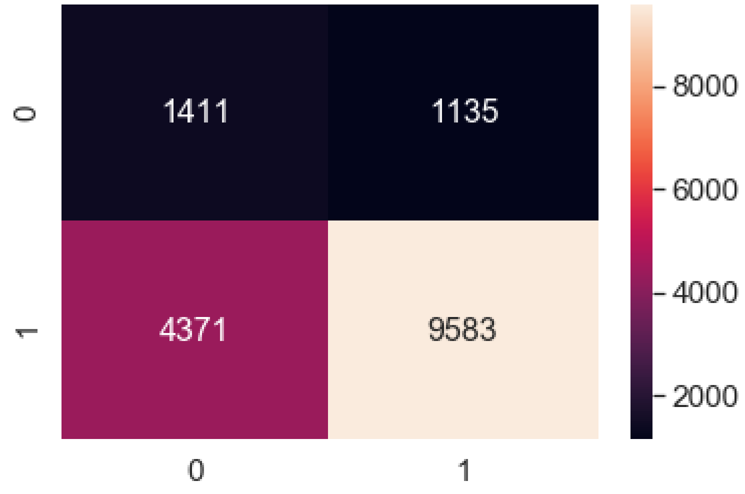
In [90]:

```
print("Test data confusion matrix")

confusion_matrix_df_test = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_
test_pred, best_t)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(confusion_matrix_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Test data confusion matrix
```

Out[90]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ed2969cc08>
```



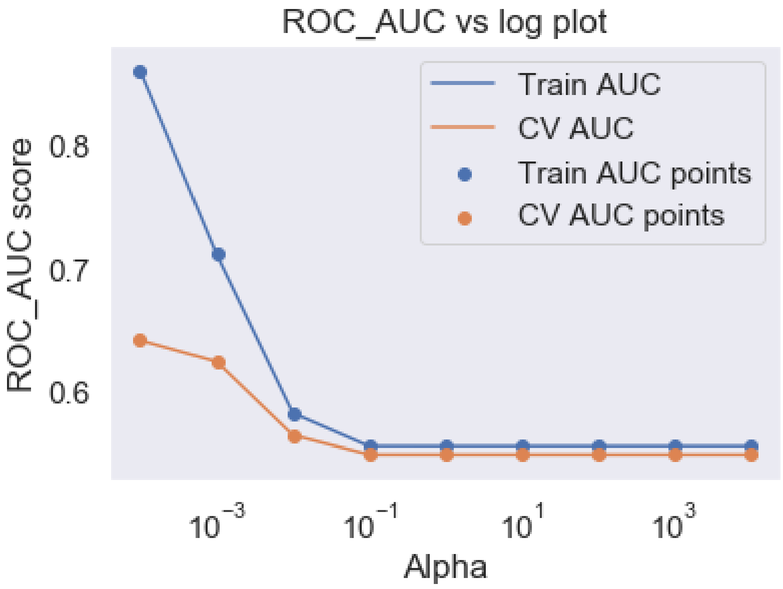
Observation

1. The number of true positives in train and test data are predominantly high

2.4.2 Applying SVM on TFIDF, SET 2

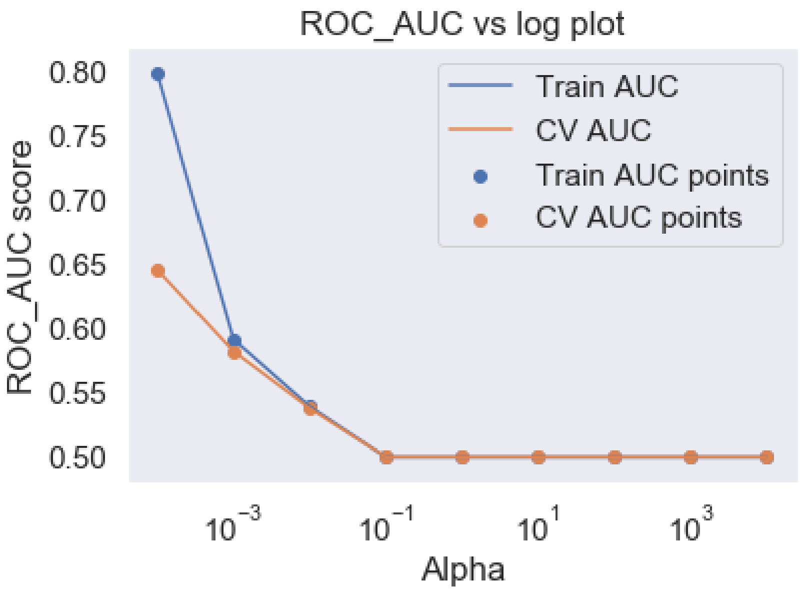
In [91]:

```
#BY USING L2 REGULARISER
# hyperparameter tuning with l2 reg
parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
sd = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced')
classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=True)
classifier.fit(X_tr_TFIDF, y_train)

train_auc = classifier.cv_results_['mean_train_score']
cv_auc= classifier.cv_results_['mean_test_score']

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')

plt.legend()
plt.xscale('log')
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs log plot")
plt.grid()
plt.show()
```


ROC_AUC vs log plot

In [92]:

```
#BY USING "L1" REGULARISER
# hyperparameter tuning with l2 reg reduce the alpha values in list
parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**
4]}
sd = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced')
classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=T
rue)
classifier.fit(X_tr_TFIDF, y_train)
train_auc = classifier.cv_results_['mean_train_score']
cv_auc= classifier.cv_results_['mean_test_score']
plt.plot(parameters['alpha'], train_auc, label='Train AUC')
plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
plt.legend()
plt.xscale('log')
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs log plot")
plt.grid()
plt.show()
```



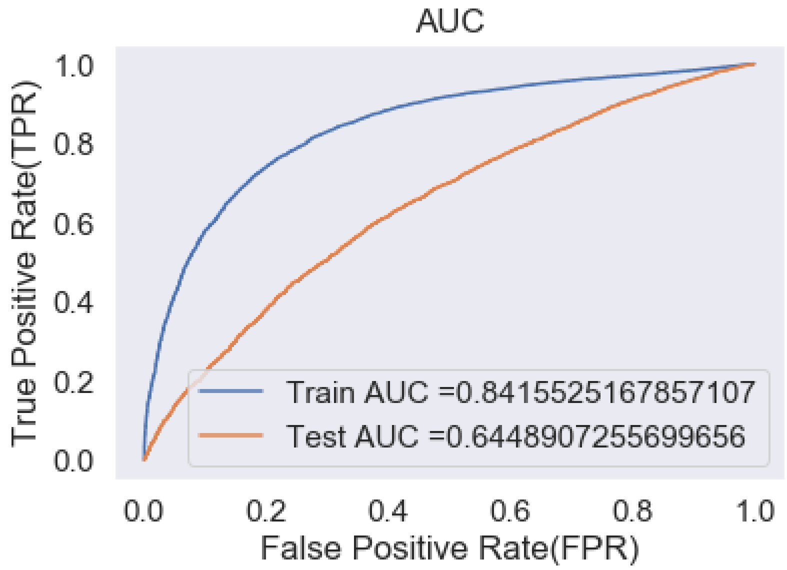
Observation:

l2 regularization works better than l1 and best alpha is 0.0001

Fitting Model to Hyper parameter curve

In [94]:

```
# https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklea
rn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l2', alpha = 0.0001,  class_w
eight = 'balanced')
Classifier_bow.fit(X_tr_TFIDF ,y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positiveclass
#https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.ht
ml#sklearn.linear_model.SGDClassifier.decision_function
y_train_pred = Classifier_bow.decision_function(X_tr_TFIDF)
y_test_pred = Classifier_bow.decision_function(X_te_TFIDF)

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



Confusion Matrix

In [95]:

```
#our objective here is to make auc the maximum
#so we find  the best threshold that will give the least fpr
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 0.5958182178888185 for threshold -0.011
Train confusion matrix
[[ 4037  1131]
 [ 6722 21610]]
```

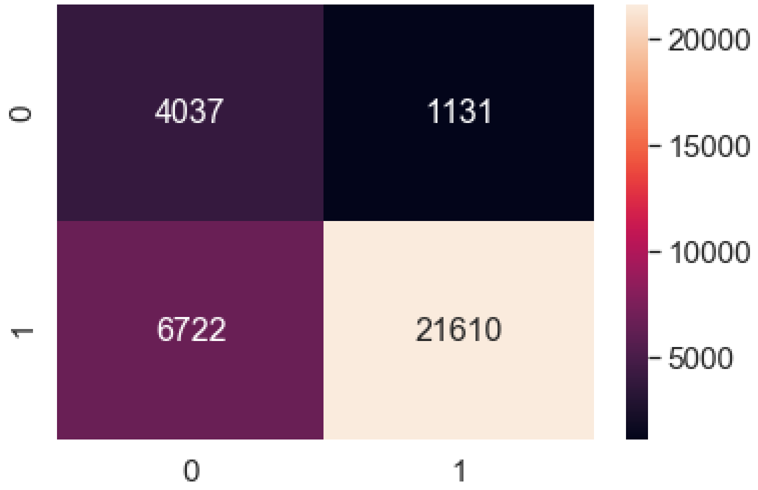
In [96]:

```
#plotting confusion matrix using seaborn's heatmap
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

print("Train data confusion matrix")

confusion_matrix_df_train = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(
y_train_pred, best_t)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(confusion_matrix_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Train data confusion matrix
```

Out[96]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ed23f76c48>
```



In [97]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
Test confusion matrix
[[1238 1308]
 [4035 9919]]
```

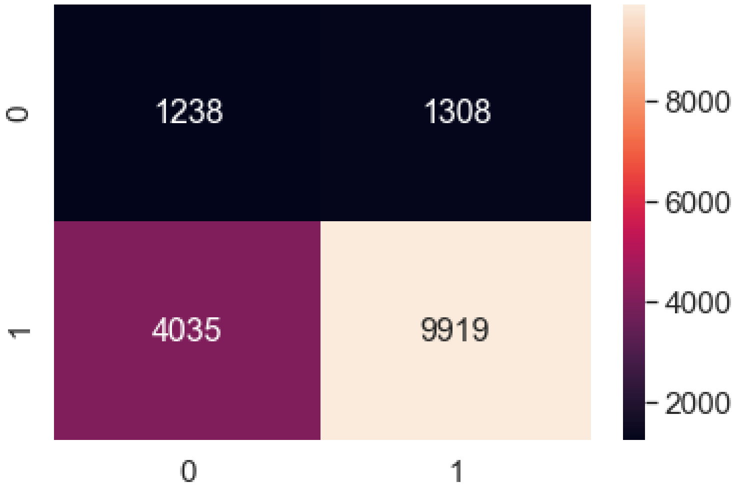
In [98]:

```
print("Test data confusion matrix")

confusion_matrix_df_test = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_
test_pred, best_t)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(confusion_matrix_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Test data confusion matrix
```

Out[98]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ed25de3248>
```



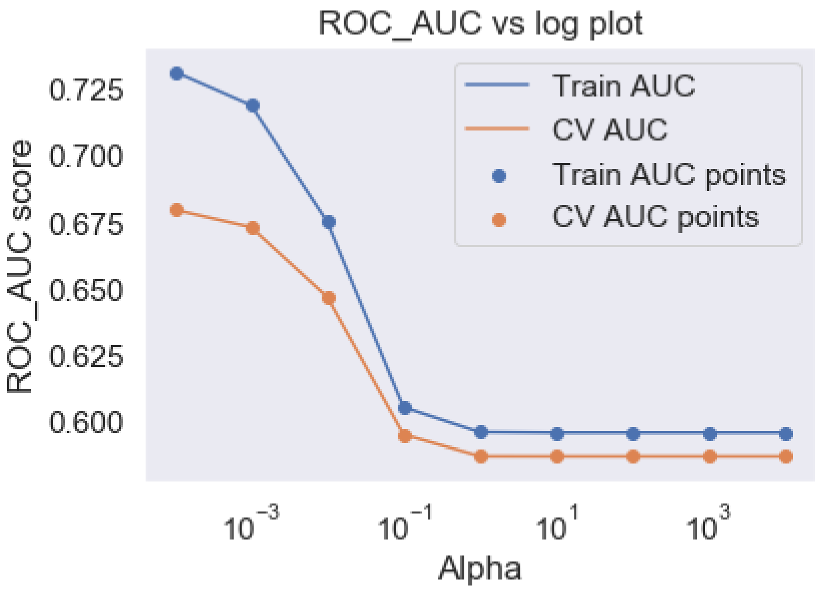
Observation

The number of true positives in train and test data are predominantly high

2.4.3 Applying SVM on AVG W2V, SET 3

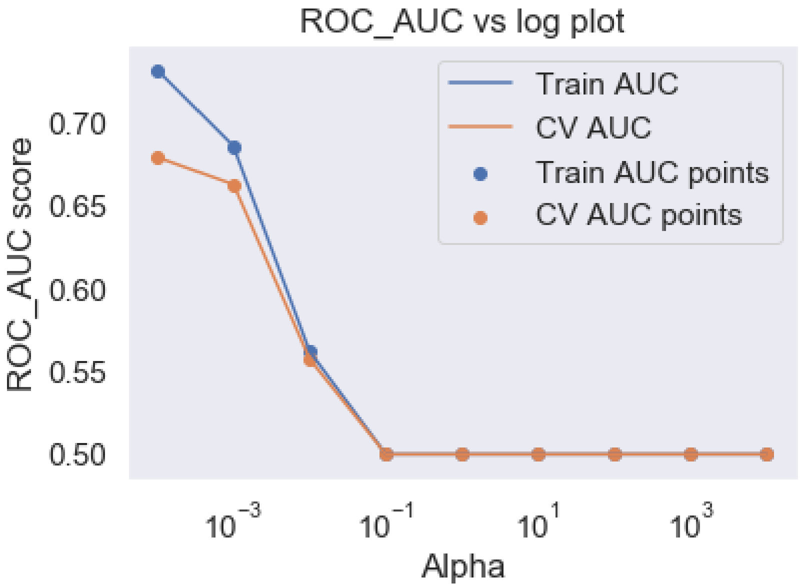
In [99]:

```
#BY USING "L2" REGULARISER
# hyperparameter tuning with l2 reg
parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
sd = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced')

classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=True)
classifier.fit(X_tr_AVG_W2V, y_train)

train_auc = classifier.cv_results_['mean_train_score']
cv_auc= classifier.cv_results_['mean_test_score']

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
plt.legend()
plt.xscale('log')
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs log plot")
plt.grid()
plt.show()
```



In [100]:

```
#BY USING "L1" REGULARISER
# hyperparameter tuning with l2 reg
parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**
4]}
sd = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced')

classifier = GridSearchCV(sd, parameters, cv= 5, scoring='roc_auc',return_train_score=T
rue)
classifier.fit(X_tr_AVG_W2V, y_train)

train_auc = classifier.cv_results_['mean_train_score']
cv_auc= classifier.cv_results_['mean_test_score']

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
plt.legend()
plt.xscale('log')
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs log plot")
plt.grid()
plt.show()
```



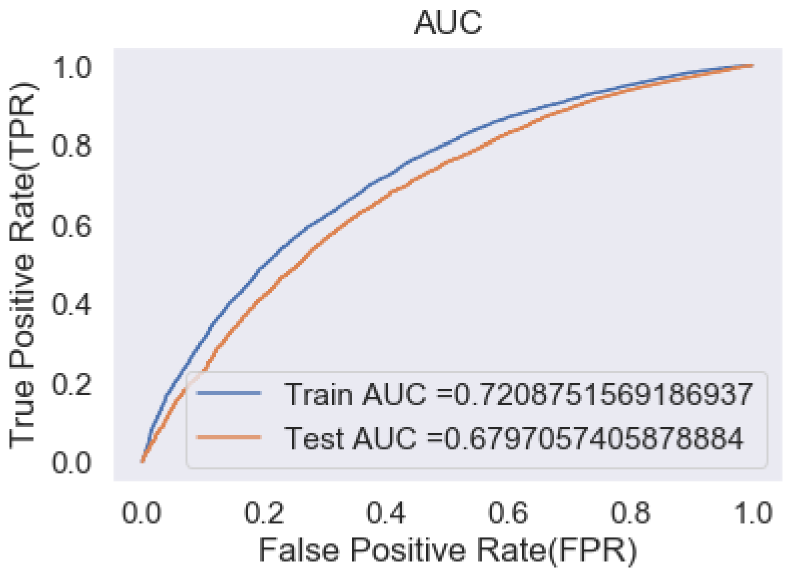
Observation:

l2 regularization works better than l1 and best alpha is 0.0001.

Fitting Model to Hyper parameter curve

In [101]:

```
# https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve

from sklearn.metrics import roc_curve, auc
Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l2', alpha = 0.0001, class_weight = 'balanced')
Classifier_bow.fit(X_tr_AVG_W2V ,y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positiveclass
#https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGDClassifier.decision_function
y_train_pred = Classifier_bow.decision_function(X_tr_AVG_W2V)
y_test_pred = Classifier_bow.decision_function(X_te_AVG_W2V)

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



Confusion Matrix

In [102]:

```
#our objective here is to make auc the maximum
#so we find  the best threshold that will give the least fpr
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 0.4391138940138796 for threshold -0.381
Train confusion matrix
[[ 3241  1927]
 [ 8494 19838]]
```

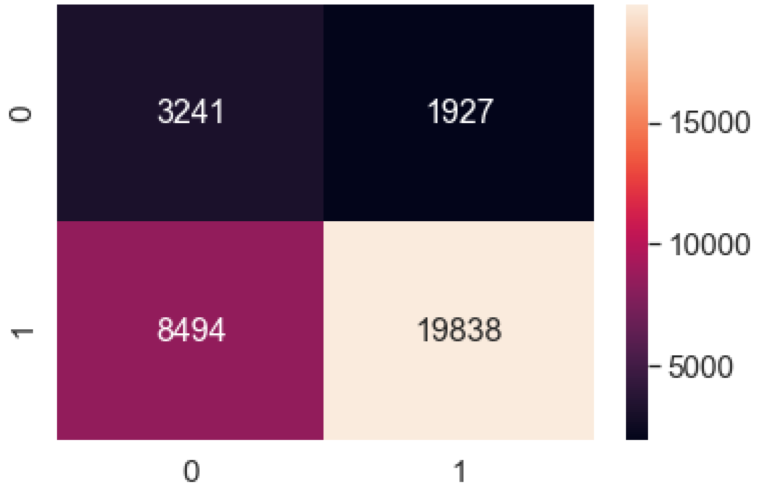
In [103]:

```
#plotting confusion matrix using seaborn's heatmap
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

print("Train data confusion matrix")

confusion_matrix_df_train = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(
y_train_pred, best_t)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(confusion_matrix_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Train data confusion matrix
```

Out[103]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ed29586608>
```



In [104]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
Test confusion matrix
[[1485 1061]
 [4367 9587]]
```

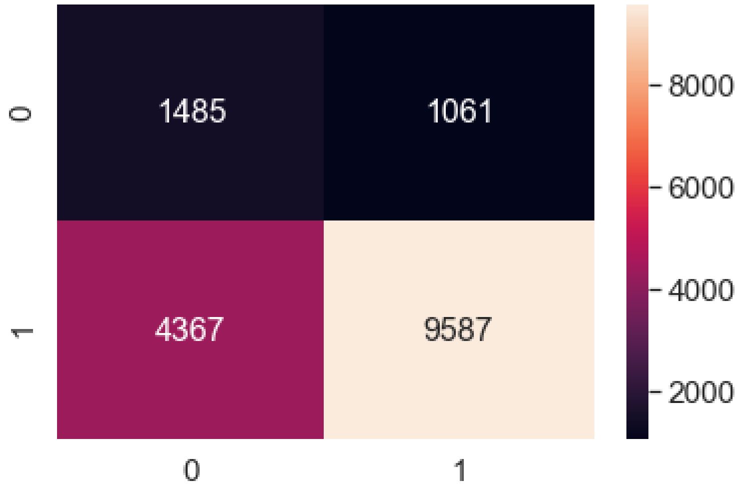
In [105]:

```
print("Test data confusion matrix")

confusion_matrix_df_test = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(confusion_matrix_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Test data confusion matrix
```

Out[105]:

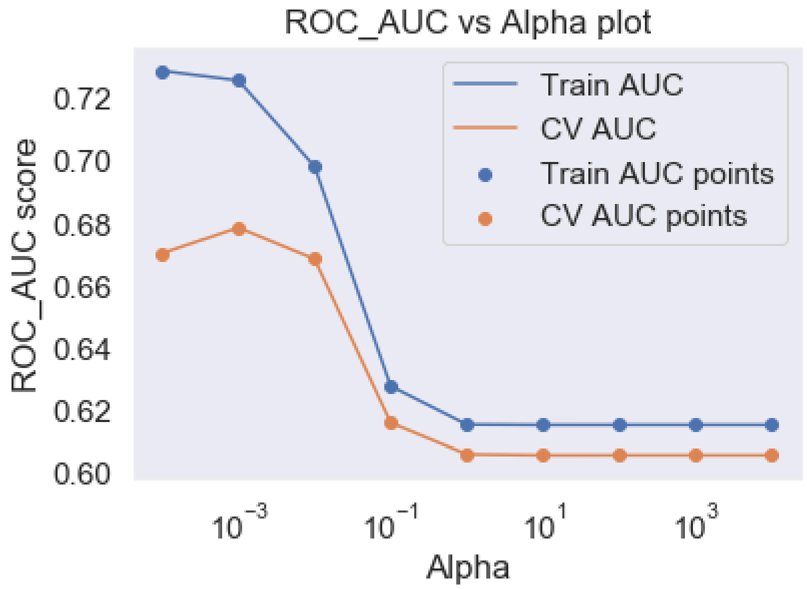
```
<matplotlib.axes._subplots.AxesSubplot at 0x1ed20bfca88>
```



2.4.4 Applying SVM on TFIDF W2V, SET 4

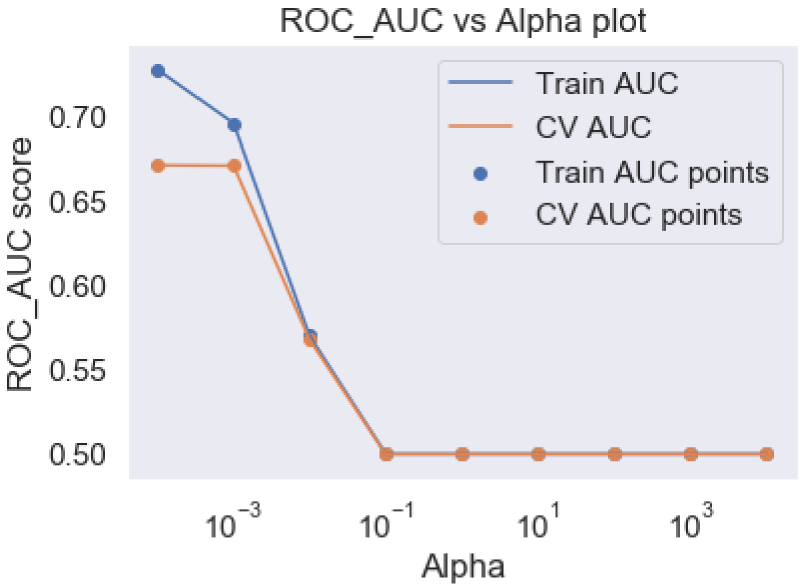
In [106]:

```
#BY USING "L2" REGULARISER

parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**
4]}
SV = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced',)

classifier = GridSearchCV(SV, parameters, cv= 5, scoring='roc_auc',return_train_score=T
rue)
classifier.fit(X_tr_TFIDF_W2V, y_train)

train_auc= classifier.cv_results_['mean_train_score']
cv_auc = classifier.cv_results_['mean_test_score']

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
plt.legend()
plt.xscale('log')
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs Alpha plot")
plt.grid()
plt.show()
```



In [107]:

```
#BY USING "L1" REGULARIZER

parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**
4]}
SV = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced')

classifier = GridSearchCV(SV, parameters, cv= 5, scoring='roc_auc',return_train_score=T
rue)
classifier.fit(X_tr_TFIDF_W2V, y_train)

train_auc= classifier.cv_results_['mean_train_score']
cv_auc = classifier.cv_results_['mean_test_score']

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
plt.legend()
plt.xscale('log')
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs Alpha plot")
plt.grid()
plt.show()
```



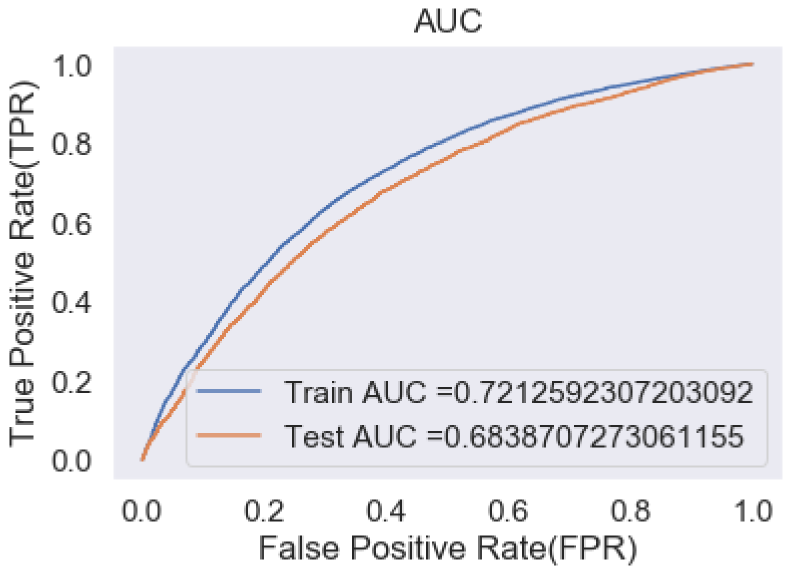
Observation:

l2 regularization works better than l1 and best alpha is 0.001.

Fitting Model to Hyper parameter curve

In [108]:

```
# https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l2', alpha = 0.001, class_weight = 'balanced')
Classifier_bow.fit(X_tr_TFIDF_W2V ,y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positiveclass
#https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGDClassifier.decision_function
y_train_pred = Classifier_bow.decision_function(X_tr_TFIDF_W2V)
y_test_pred = Classifier_bow.decision_function(X_te_TFIDF_W2V)

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



Confusion Matrix

In [109]:

```
#our objective here is to make auc the maximum
#so we find  the best threshold that will give the least fpr
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 0.4489254101850286 for threshold 0.086
Train confusion matrix
[[ 3418  1750]
 [ 9101 19231]]
```

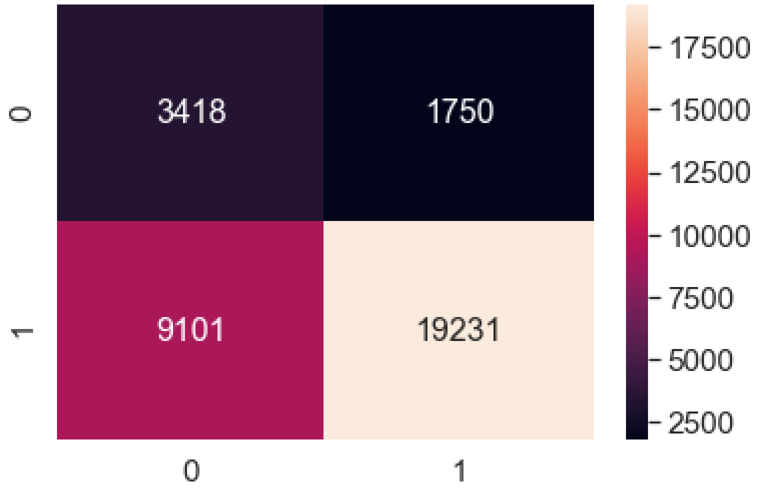
In [110]:

```
#plotting confusion matrix using seaborn's heatmap
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

print("Train data confusion matrix")

confusion_matrix_df_train = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(
y_train_pred, best_t)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(confusion_matrix_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Train data confusion matrix
```

Out[110]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ed21e87508>
```



In [111]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
Test confusion matrix
[[1563  983]
 [4622 9332]]
```

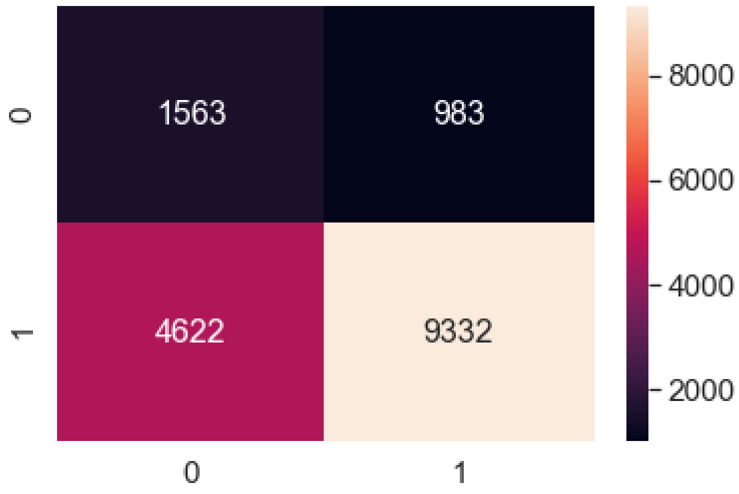
In [112]:

```
print("Test data confusion matrix")

confusion_matrix_df_test = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(confusion_matrix_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Test data confusion matrix
```

Out[112]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ed21702788>
```



Adding New Features

We add 3 new features:

```
1. Sentiment scores of each essay
2. Number of words in title
3. Number of words in combined esssays
```

No.of Words in title

In [113]:

```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 33500 entries, 49271 to 41202
Data columns (total 11 columns):
 #   Column                                        Non-Null Count  Dtype  
---  ------                                        --------------  -----  
 0   id                                            33500 non-null  object 
 1   school_state                                  33500 non-null  object 
 2   teacher_number_of_previously_posted_projects  33500 non-null  int64  
 3   clean_categories                              33500 non-null  object 
 4   clean_subcategories                           33500 non-null  object 
 5   clean_teacher_prefix                          33500 non-null  object 
 6   clean_project_grade_category                  33500 non-null  object 
 7   preprocessed_essays                           33500 non-null  object 
 8   preprocessed_titles                           33500 non-null  object 
 9   price                                         33500 non-null  float64
 10  quantity                                      33500 non-null  int64  
dtypes: float64(1), int64(2), object(8)
memory usage: 3.1+ MB
```

In [114]:

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

# import nltk
nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()

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


for a in tqdm(project_data["preprocessed_essays"]) :
    b = sid.polarity_scores(a)['neg']
    c = sid.polarity_scores(a)['pos']
    d = sid.polarity_scores(a)['neu']
    e = sid.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\KALYAN
[nltk_data]     SRINIVAS\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
100%|██████████| 50000/50000 [13:58<00:00, 59.60it/s]
```

In [115]:

```
project_data["pos"] = pos
project_data["neg"] = neg
project_data["neu"] = neu
project_data["compound"] = compound
```

Essays and title word count

In [116]:

```
project_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50000 entries, 0 to 49999
Data columns (total 16 columns):
 #   Column                                        Non-Null Count  Dtype  
---  ------                                        --------------  -----  
 0   id                                            50000 non-null  object 
 1   school_state                                  50000 non-null  object 
 2   teacher_number_of_previously_posted_projects  50000 non-null  int64  
 3   project_is_approved                           50000 non-null  int64  
 4   clean_categories                              50000 non-null  object 
 5   clean_subcategories                           50000 non-null  object 
 6   clean_teacher_prefix                          50000 non-null  object 
 7   clean_project_grade_category                  50000 non-null  object 
 8   preprocessed_essays                           50000 non-null  object 
 9   preprocessed_titles                           50000 non-null  object 
 10  price                                         50000 non-null  float64
 11  quantity                                      50000 non-null  int64  
 12  pos                                           50000 non-null  float64
 13  neg                                           50000 non-null  float64
 14  neu                                           50000 non-null  float64
 15  compound                                      50000 non-null  float64
dtypes: float64(5), int64(3), object(8)
memory usage: 6.5+ MB
```

In [117]:

```
essay_word_count = []
for ess in project_data["preprocessed_essays"] :
    c = len(ess.split())
    essay_word_count.append(c)
```

In [118]:

```
project_data['essay_word_count'] = essay_word_count
```

In [119]:

```
title_word_count = []
for ess in project_data["preprocessed_titles"] :
    c = len(ess.split())
    title_word_count.append(c)
```

In [120]:

```
project_data['title_word_count'] = title_word_count
```

In [121]:

```
y = project_data['project_is_approved'].values
X = project_data.drop(['project_is_approved'], axis=1)
X.head(1)
```

Out[121]:

	id	school_state	teacher_number_of_previously_posted_projects	clean_categories	c
0	p253737	IN	0	Literacy_Language	

In [122]:

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

In [ ]:

```

```

Pos Vectorization

In [123]:

```
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['pos'].values.reshape(1,-1))

X_train_pos_norm = normalizer.transform(X_train['pos'].values.reshape(1,-1))
X_test_pos_norm = normalizer.transform(X_test['pos'].values.reshape(1,-1))

print("After vectorizations")
print(X_train_pos_norm.shape, y_train.shape)
print(X_test_pos_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(1, 33500) (33500,)
(1, 16500) (16500,)
==========================================================================
==========================
```

In [124]:

```
print("Transpose of pos")

X_train_pos_norm = X_train_pos_norm.transpose()
X_test_pos_norm = X_test_pos_norm.transpose()


print("After vectorizations")
print(X_train_pos_norm.shape, y_train.shape)
print(X_test_pos_norm.shape, y_test.shape)
print("="*100)
```

```
Transpose of pos
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
==========================================================================
==========================
```

Neg Vectorization

In [125]:

```
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['neg'].values.reshape(1,-1))

X_train_neg_norm = normalizer.transform(X_train['neg'].values.reshape(1,-1))
X_test_neg_norm = normalizer.transform(X_test['neg'].values.reshape(1,-1))

print("After vectorizations")
print(X_train_neg_norm.shape, y_train.shape)
print(X_test_neg_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(1, 33500) (33500,)
(1, 16500) (16500,)
==========================================================================
==========================
```

In [126]:

```
print("Transpose of neg")

X_train_neg_norm = X_train_neg_norm.transpose()
X_test_neg_norm = X_test_neg_norm.transpose()


print("After vectorizations")
print(X_train_neg_norm.shape, y_train.shape)
print(X_test_neg_norm.shape, y_test.shape)
print("="*100)
```

```
Transpose of neg
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
==========================================================================
==========================
```

Neutral vectorization

In [127]:

```
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['neu'].values.reshape(1,-1))

X_train_neu_norm = normalizer.transform(X_train['neu'].values.reshape(1,-1))
X_test_neu_norm = normalizer.transform(X_test['neu'].values.reshape(1,-1))

print("After vectorizations")
print(X_train_neu_norm.shape, y_train.shape)
print(X_test_neu_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(1, 33500) (33500,)
(1, 16500) (16500,)
==========================================================================
==========================
```

In [128]:

```
print("Transpose of neutral")

X_train_neu_norm = X_train_neu_norm.transpose()
X_test_neu_norm = X_test_neu_norm.transpose()


print("After vectorizations")
print(X_train_neu_norm.shape, y_train.shape)
print(X_test_neu_norm.shape, y_test.shape)
print("="*100)
```

```
Transpose of neutral
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
==========================================================================
==========================
```

compound vectorization

In [129]:

```
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['compound'].values.reshape(1,-1))

X_train_compound_norm = normalizer.transform(X_train['compound'].values.reshape(1,-1))
X_test_compound_norm = normalizer.transform(X_test['compound'].values.reshape(1,-1))

print("After vectorizations")
print(X_train_compound_norm.shape, y_train.shape)
print(X_test_compound_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(1, 33500) (33500,)
(1, 16500) (16500,)
==========================================================================
==========================
```

In [130]:

```
print("Transpose of compound")

X_train_compound_norm = X_train_compound_norm.transpose()
X_test_compound_norm = X_test_compound_norm.transpose()


print("After vectorizations")
print(X_train_compound_norm.shape, y_train.shape)
print(X_test_compound_norm.shape, y_test.shape)
print("="*100)
```

```
Transpose of compound
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
==========================================================================
==========================
```

essay word count vectorization

In [131]:

```
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['essay_word_count'].values.reshape(1,-1))

X_train_essay_word_count_norm = normalizer.transform(X_train['essay_word_count'].values
.reshape(1,-1))
X_test_essay_word_count_norm = normalizer.transform(X_test['essay_word_count'].values.r
eshape(1,-1))

print("After vectorizations")
print(X_train_essay_word_count_norm.shape, y_train.shape)
print(X_test_essay_word_count_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(1, 33500) (33500,)
(1, 16500) (16500,)
==========================================================================
==========================
```

In [132]:

```
print("Transpose of essay word count norm")

X_train_essay_word_count_norm = X_train_essay_word_count_norm.transpose()
X_test_essay_word_count_norm = X_test_essay_word_count_norm.transpose()

print("After vectorizations")
print(X_train_essay_word_count_norm.shape, y_train.shape)
print(X_test_essay_word_count_norm.shape, y_test.shape)
print("="*100)
```

```
Transpose of essay word count norm
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
==========================================================================
==========================
```

Title word count vectorization

In [133]:

```
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['title_word_count'].values.reshape(1,-1))

X_train_title_word_count_norm = normalizer.transform(X_train['title_word_count'].values
.reshape(1,-1))
X_test_title_word_count_norm = normalizer.transform(X_test['title_word_count'].values.r
eshape(1,-1))

print("After vectorizations")
print(X_train_title_word_count_norm.shape, y_train.shape)
print(X_test_title_word_count_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(1, 33500) (33500,)
(1, 16500) (16500,)
==========================================================================
==========================
```

In [134]:

```
print("Transpose of essay word count norm")

X_train_title_word_count_norm = X_train_title_word_count_norm.transpose()
X_test_title_word_count_norm = X_test_title_word_count_norm.transpose()

print("After vectorizations")
print(X_train_title_word_count_norm.shape, y_train.shape)
print(X_test_title_word_count_norm.shape, y_test.shape)
print("="*100)
```

```
Transpose of essay word count norm
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
==========================================================================
==========================
```

TFIDF Vectorizer of essay text

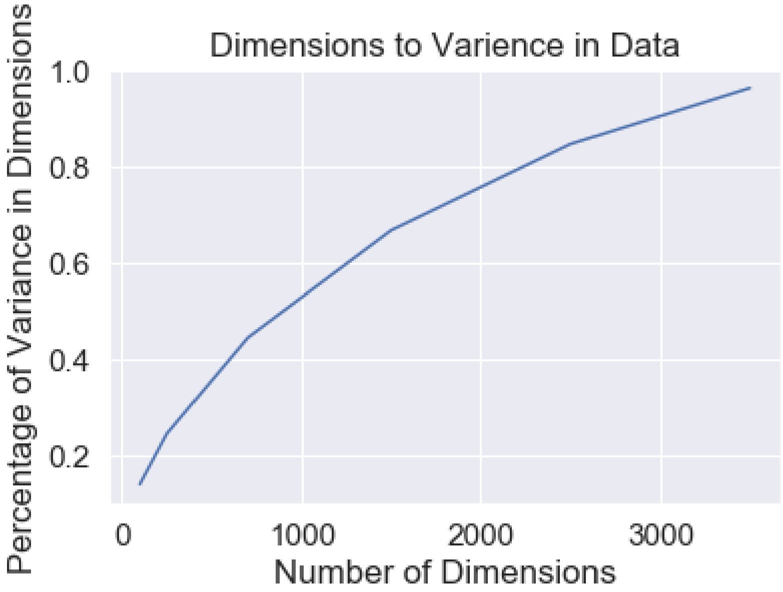
In [135]:

```
#Dimensions are very large so we take few.
X_train_tf_essay=X_train_essay_tfidf[:,0:4000]
X_test_tf_essay=X_test_essay_tfidf[:,0:4000]
from sklearn.decomposition import TruncatedSVD
#https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html
#declaring index as Dimensions in train_text_tfidf
Di = [100,250,700,1500,2500,3500]
Varience_sum = []
for i in tqdm(Di):
    svd = TruncatedSVD(n_components = i, random_state = 42)
    svd.fit(X_train_tf_essay)
    Varience_sum.append(svd.explained_variance_ratio_.sum())
```

```
100%|██████████| 6/6 [10:48<00:00, 108.06s/it]
```

In [136]:

```
plt.xlabel("Number of Dimensions")
plt.ylabel("Percentage of Variance in Dimensions")
plt.title("Dimensions to Varience in Data")
plt.plot(Di,Varience_sum)
plt.show()
```



Observation

At 2500 dimensions we have Accuracy of greater than 90% so considering 2500 dimensions

In [137]:

```
svd = TruncatedSVD(n_components= 2500)
svd.fit(X_train_tf_essay)
#Transforms:
#Train SVD
X_train_tf_essay= svd.transform(X_train_tf_essay )
#Test SVD
X_test_tf_essay = svd.transform(X_test_tf_essay )
```

In [ ]:

```
```

Merging all features into Set5

In [138]:

```
#for train
# combine all
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
:)
X_set5_train = hstack((X_train_tf_essay,X_train_pos_norm, X_train_neg_norm, X_train_neu
_norm, X_train_compound_norm, X_train_essay_word_count_norm, X_train_title_word_count_n
orm, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_CSC_ohe, X_trai
n_CC_ohe, X_train_price_norm, X_train_quantity_norm, X_train_TPPP_norm)).tocsr()


X_set5_test = hstack((X_test_tf_essay,X_test_pos_norm, X_test_neg_norm, X_test_neu_norm
, X_test_compound_norm, X_test_essay_word_count_norm, X_test_title_word_count_norm, X_t
est_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_CSC_ohe, X_test_CC_ohe, X_t
est_price_norm, X_test_quantity_norm, X_test_TPPP_norm)).tocsr()
```

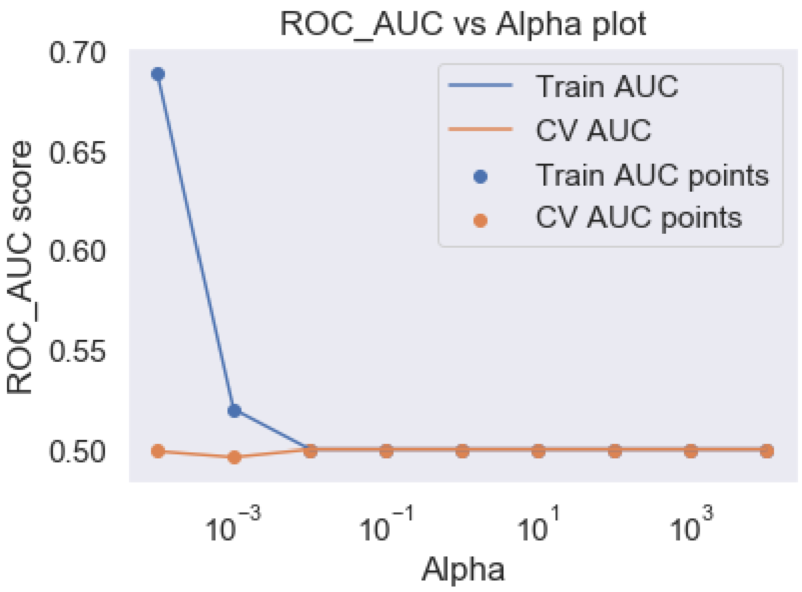
2.5 Support Vector Machines with added Features `Set 5`

In [139]:

```
# please write all the code with proper documentation, and proper titles for each subse
ction
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your
code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

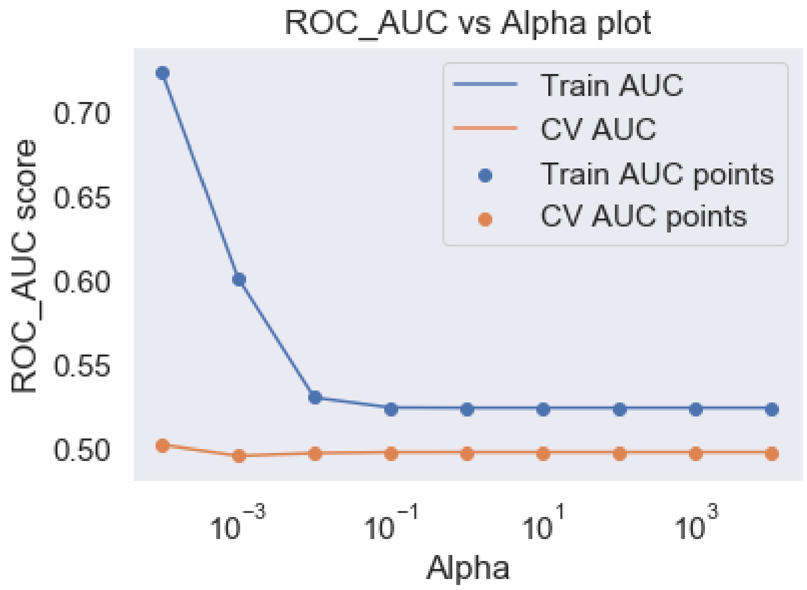
In [140]:

```
#BY USING L1 RGULARISER
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
#from sklearn.datasets import *
from sklearn import linear_model
from sklearn.linear_model import SGDClassifier
from sklearn import svm
# hyperparameter tuning with l2 reg
""#we are using L1 Regularizer
parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**
4]}
SV = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced',)
classifier = GridSearchCV(SV, parameters, cv= 3, scoring='roc_auc',return_train_score=T
rue)
classifier.fit(X_set5_train, y_train)

train_auc= classifier.cv_results_['mean_train_score']
cv_auc = classifier.cv_results_['mean_test_score']

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
plt.plot(parameters['alpha'], cv_auc, label='CV AUC')

plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
plt.legend()
plt.xscale("log")
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs Alpha plot")
plt.grid()
plt.show()
```



In [141]:

```
#BY USING L2 REGULARISER

parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**
4]}
SV = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced',)
classifier = GridSearchCV(SV, parameters, cv= 3, scoring='roc_auc',return_train_score=T
rue)
classifier.fit(X_set5_train, y_train)

train_auc= classifier.cv_results_['mean_train_score']
cv_auc = classifier.cv_results_['mean_test_score']

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
plt.legend()
plt.xscale("log")
plt.xlabel("Alpha")
plt.ylabel("ROC_AUC score")
plt.title("ROC_AUC vs Alpha plot")
plt.grid()
plt.show()
```



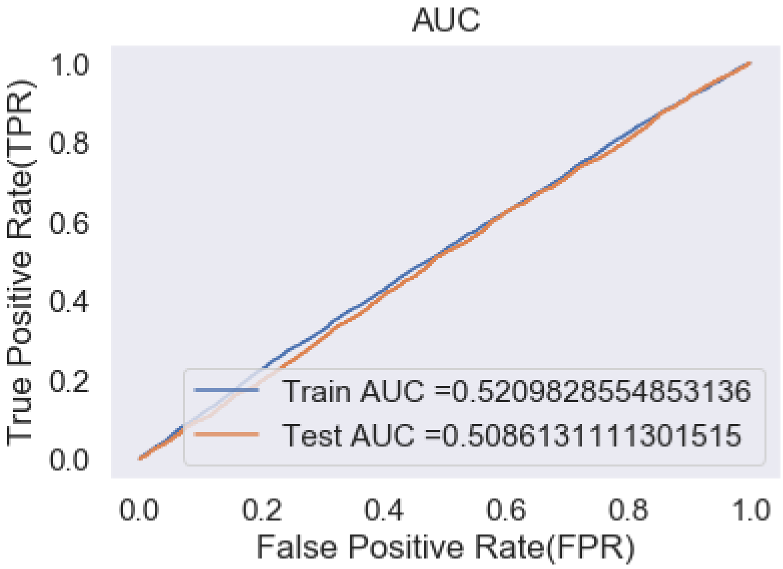
Observation

l2 regularization works better than l1 and best alpha is 0.01

Fitting Model to Hyper parameter curve

In [142]:

```
# https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
Classifier_bow = SGDClassifier(loss = 'hinge', penalty = 'l2', alpha = 0.01, class_weight = 'balanced')
Classifier_bow.fit(X_set5_train ,y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positiveclass
#https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGDClassifier.decision_function
y_train_pred = Classifier_bow.decision_function(X_set5_train)
y_test_pred = Classifier_bow.decision_function(X_set5_test)

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



Confusion Matrix

In [143]:

```
#our objective here is to make auc the maximum
#so we find  the best threshold that will give the least fpr
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 0.265610432295703 for threshold 0.14
Train confusion matrix
[[ 2820  2348]
 [14541 13791]]
```

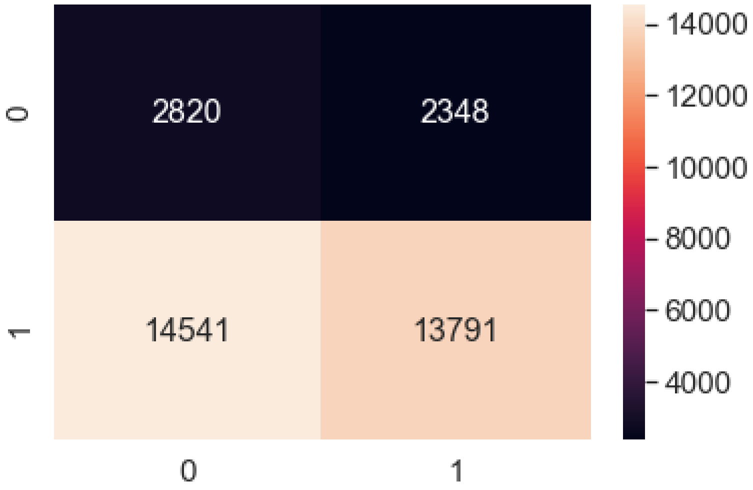
In [144]:

```
#plotting confusion matrix using seaborn's heatmap
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

print("Train data confusion matrix")

confusion_matrix_df_train = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(
y_train_pred, best_t)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(confusion_matrix_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Train data confusion matrix
```

Out[144]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ed21770988>
```



In [145]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
Test confusion matrix
[[1336 1210]
 [7094 6860]]
```

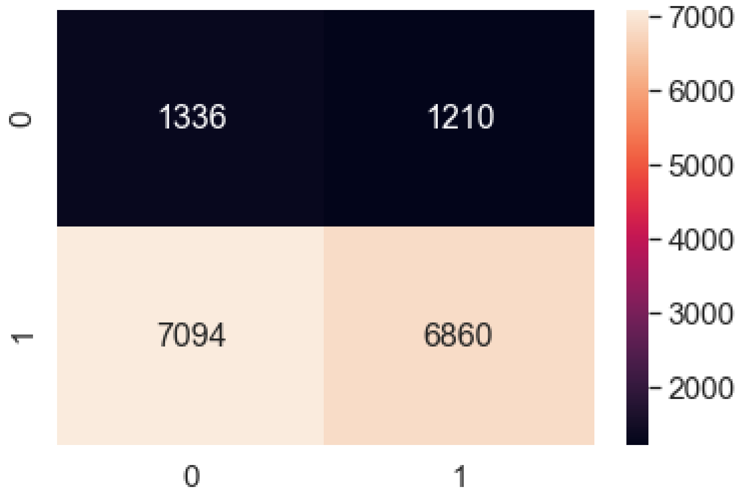
In [146]:

```
print("Test data confusion matrix")

confusion_matrix_df_test = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_
test_pred, best_t)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(confusion_matrix_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Test data confusion matrix
```

Out[146]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ed22aeaa88>
```



In [ ]:

```

```

3. Conclusion

In [147]:

```
# Please compare all your models using Prettytable library
#how to use pretty table http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
tb = PrettyTable()
tb.field_names= ("Vectorizer", " Alpha ", " AUC ")
tb.add_row(["BOW ", 0.01, 0.66])
tb.add_row(["Tf - Idf ", 0.0001, 0.64])
tb.add_row(["AVG - W2V", 0.0001, 0.67])
tb.add_row(["AVG - Tf - Idf", 0.001, 0.68])
tb.add_row(["SVD TFIDF", 0.01, 0.50])
print(tb.get_string(titles = "SVM- Observations"))
```

```
+----------------+---------+-------+
|   Vectorizer   |  Alpha  |  AUC  |
+----------------+---------+-------+
|      BOW       |   0.01  |  0.66 |
|   Tf - Idf     |  0.0001 |  0.64 |
|   AVG - W2V    |  0.0001 |  0.67 |
| AVG - Tf - Idf |  0.001  |  0.68 |
|   SVD TFIDF    |   0.01  |  0.5  |
+----------------+---------+-------+
```

In [ ]: