

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	
<code>project_id</code>	A unique identifier for the proposed project. Example: 1234567890
<code>project_title</code>	Title of the project. Example: Art Will Make First Grade
<code>project_grade_category</code>	Grade level of students for which the project is targeted. One of the following enumerated categories: <ul style="list-style-type: none">• Elementary School• Middle School• High School• College
<code>project_subject_categories</code>	One or more (comma-separated) subject categories for the project. The following enumerated categories are available: <ul style="list-style-type: none">• Applied Science• Art• Health, Physical Education, and Safety• History• Literacy• Math• Music• Special Education
<code>project_subject_subcategories</code>	One or more (comma-separated) subject subcategories. The following enumerated categories are available: <ul style="list-style-type: none">• Music• Literacy & Language, Math & Science
<code>school_state</code>	State where school is located (Two-letter U.S. state abbreviations) (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#List_of_two-letter_U.S._state_abbreviations)
<code>project_resource_summary</code>	An explanation of the resources needed for the project. <ul style="list-style-type: none">• My students need hands on literacy material sense of community
<code>project_essay_1</code>	First applicant essay
<code>project_essay_2</code>	Second applicant essay
<code>project_essay_3</code>	Third applicant essay
<code>project_essay_4</code>	Fourth applicant essay
<code>project_submitted_datetime</code>	Datetime when project application was submitted. Example: 12/1/2014 12:00:00
<code>teacher_id</code>	A unique identifier for the teacher of the proposed project. Example: bdf8baa8fedef6bfeec7a6

Feature

Teacher's title. One of the following enum

teacher_prefix

-
-
-
-
-

teacher_number_of_previously_posted_projects

Number of project applications previously submitted by the

* 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 <code>project_id</code> value from the <code>train.csv</code> file. Example: p036502
description	Description 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

Notes on the Essay Data

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

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

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

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

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

In [1]:

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

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

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

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

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

from tqdm import tqdm
import os

from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
from scipy.sparse import hstack
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix
from prettytable import PrettyTable
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from sklearn.decomposition import TruncatedSVD
```

1.1 Reading Data

In [2]:

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

Adding price attribute to project_data dataframe from resources using merge function

In [3]:

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

In [4]:

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

Number of data points in train data (109248, 19)

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'
 'price' 'quantity']

In [5]:

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

Number of data points in train data (1541272, 4)

['id' 'description' 'quantity' 'price']

Out[5]:

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

1.2 preprocessing of project_subject_categories

In [6]:

```
categories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.c
om/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 categories:
    temp = ""
    # consider we have text like this "Mathidf & 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 s
pace "Math & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to r
eplace it with ''(i.e removing 'The')
            j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empt
y) ex:"Math & Science"=>"Math&Science"
            temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trail
ing spaces
            temp = temp.replace('&','_') # we are replacing the & value into
cat_list.append(temp.strip())

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

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

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

1.3 preprocessing of project_subject_subcategories

In [7]:

```

sub_categories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.c
om/a/47301924/4084039

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

sub_cat_list = []
for i in sub_categories:
    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 category based on s
pace "Math & Science"=> "Math", "&", "Science"
            j=j.replace('The', '') # if we have the words "The" we are going to r
eplace it with '' (i.e removing 'The')
            j = j.replace(' ', '') # we are placing all the ' ' (space) with '' (empt
y) ex: "Math & Science"=> "Math&Science"
            temp +=j.strip()+" #" "abc ".strip() will return "abc", remove the trail
ing spaces
            temp = temp.replace('&', '_')
            sub_cat_list.append(temp.strip())

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

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

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

```

1.3 Text preprocessing

In [8]:

```

# merge two column text dataframe:
project_data["essay"] = project_data["project_essay_1"].map(str) + \
    project_data["project_essay_2"].map(str) + \
    project_data["project_essay_3"].map(str) + \
    project_data["project_essay_4"].map(str)

```

In [9]:

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

Out[9]:

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



In [10]:

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


In [11]:

```
# 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)
```


How do you remember your days of school? Was it in a sterile environment with plain walls, rows of desks, and a teacher in front of the room? A typical day in our room is nothing like that. I work hard to create a warm inviting themed room for my students look forward to coming to each day.\r\n\r\nMy class is made up of 28 wonderfully unique boys and girls of mixed races in Arkansas.\r\n\r\nThey attend a Title I school, which means there is a high enough percentage of free and reduced-price lunch to qualify. Our school is an "open classroom" concept, which is very unique as there are no walls separating the classrooms. These 9 and 10 year-old students are very eager learners; they are like sponges, absorbing all the information and experiences and keep on wanting more. With these resources such as the comfy red throw pillows and the whimsical nautical hanging decor and the blue fish nets, I will be able to help create the mood in our classroom setting to be one of a themed nautical environment. Creating a classroom environment is very important in the success in each and every child's education. The nautical photo props will be used with each child as they step foot into our classroom for the first time on Meet the Teacher evening. I'll take pictures of each child with them, have them developed, and then hung in our classroom ready for their first day of 4th grade. This kind gesture will set the tone before even the first day of school! The nautical thank you cards will be used throughout the year by the students as they create thank you cards to their team groups.\r\n\r\n\r\nYour generous donations will help me to help make our classroom a fun, inviting, learning environment from day one.\r\n\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. They are eager beavers and always strive to work their hardest working past their limitations. \r\n\r\n\r\nThe materials we have are the ones I seek out for my students. I teach in a Title I school where most of the students receive free or reduced price lunch. Despite their disabilities and limitations, my students love coming to school and come eager to learn and 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. They want to be able to move as they learn or so they say. Wobble chairs are the answer and I love them because they develop their core, which enhances gross motor and in turn fine motor skills. \r\n\r\nThey also want to learn 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 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

=====
The mediocre teacher tells. The good teacher explains. The superior teacher demonstrates. The great teacher inspires. -William A. Ward\r\n\r\n\r\nMy school 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 made up of 23.2% African-American students. Most of the students are on free or reduced lunch. We aren't receiving doctors, lawyers, or engineers children from rich backgrounds or neighborhoods. As an educator I am inspiring minds of young 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 message.

Due to the volume of my speaker my students can't hear videos or books clearly and it isn't making the lessons as meaningful. But with the bluetooth speaker my students will be able to hear and I can stop, pause and replay it at any time.\r\nThe cart will allow me to have more room for storage of things that are needed for the day and has an extra part to it I can 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 [12]:

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

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

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

In [13]:

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

My kindergarten students have varied disabilities ranging from speech and language delays, cognitive delays, gross/fine motor delays, to autism. They are eager beavers and always strive to work their hardest working past their limitations. \r\n\r\nThe materials we have are the ones I seek out for my students. I teach in a Title I school where most of the students receive free or reduced price lunch. Despite their disabilities and limitations, my students love coming to school and come eager to learn and 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. They want to be able to move as they learn or so they say. Wobble chairs are the answer and I love them because they develop their core, which enhances gross motor and in turn fine motor skills. \r\n\r\nThey also want to learn through games, my kids do not want to sit and do worksheets. They want to learn to count by jumping and playing. Physical engagement is the 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 [14]:

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

My kindergarten students have varied disabilities ranging from speech and language delays, cognitive delays, gross/fine motor delays, to autism. They are 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 reduced price lunch. Despite their disabilities and limitations, my students love coming to school and come eager to learn and explore. Have you ever felt like you had 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. They want to be able to move as they learn or so they say. Wobble chairs are the answer and I love them because they develop their core, which enhances gross motor and in turn fine motor skills. They also want to learn through games, my kids do not want to sit and do 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. nanan

In [15]:

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

My kindergarten students have varied disabilities ranging from speech and language delays cognitive delays gross fine motor delays to autism They are 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 reduced price lunch Despite their disabilities and limitations my students love coming to school and come eager to learn and explore Have you ever felt like you had 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 They want to be able to move as they learn or so they say Wobble chairs are the answer and I love them because they develop their core which enhances gross motor and in turn fine motor skills They also want to learn through games my kids do not want to sit and do 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 nanan

In [16]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= {'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',
"you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he',
'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itse
lf', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'tha
t', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'ha
s', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'becaus
e', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 't
hrough', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'of
f', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'al
l', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than'
, 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should'v
e", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "d
idn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma'
, 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "should
n't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"}
```

In [17]:

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

100%|██████████| 109248/109248 [00:14<00:00, 7552.79it/s]

In [18]:

```
# after preprocessing
preprocessed_essays[20000]
```

Out[18]:

```
'kindergarten students varied disabilities ranging speech language d
elays cognitive delays gross fine motor delays autism eager beavers
always strive work hardest working past limitations the materials on
es seek students teach title school students receive free reduced pr
ice lunch despite disabilities limitations students love coming scho
ol come eager learn explore have ever felt like ants pants needed gr
oove move meeting kids feel time want able move learn say wobble cha
irs answer love develop core enhances gross motor turn fine motor sk
ills they also want learn games kids want sit worksheets want learn
count jumping playing physical engagement key success number toss co
lor shape mats make happen students forget work fun 6 year old deser
ves nannan'
```

In [19]:

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

In [20]:

```
project_data.drop(['project_essay_1', 'project_essay_2', 'project_essay_3', 'projec
t_essay_4'], axis=1, inplace=True)
```

1.4 Preprocessing of `project_title`

- Decontract project titles, remove line breaks and extra spaces, convert everything to lowercase and then remove all the stop words.

In [21]:

```
preprocessed_titles = []

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

```
100%|██████████| 109248/109248 [00:01<00:00, 68523.62it/s]
```

In [22]:

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

Pre-processing teacher_prefix

In [23]:

```
#remove nan from teacher prefix:
#https://stackoverflow.com/questions/21011777/how-can-i-remove-nan-from-list-pyth
hon-numpy
def remove_nan(prefix):
    if str(prefix)!='nan':
        pr = str(prefix)
        pr = re.sub("\\\\.", "", pr) #remove dot from the end of prefix
        return pr
    return "none"

cleaned_teacher_prefix = project_data['teacher_prefix'].map(remove_nan)
project_data['clean_teacher_prefix'] = cleaned_teacher_prefix
```

In [24]:

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

Pre-process project_grade_category

- Clean the project grade categories:
 - Convert Grades 3-5 ==> Grades_3_5

In [25]:

```
def clean_project_grades(grade):
    grade = re.sub("\\-", "_", grade)
    grade = re.sub(" ", "_", grade)
    return grade.strip()

clean_grades = project_data['project_grade_category'].map(clean_project_grades)
project_data['clean_grade_category'] = clean_grades
```

In [26]:

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

In [27]:

```
# Dropping all features we won't need going forward
project_data.drop(['project_resource_summary'],axis=1,inplace=True)
project_data.drop(['Unnamed: 0', 'teacher_id'],axis=1,inplace=True)
```


In [28]:

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

Out[28]:

	id	school_state	project_submitted_datetime	teacher_number_of_previously_posted_p
0	p253737	IN	2016-12-05 13:43:57	
1	p258326	FL	2016-10-25 09:22:10	

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 parameter 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 parameter 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-tp-r-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>)
(<https://seaborn.pydata.org/generated/seaborn.heatmap.html>)

4. [Task-2] Apply the Support Vector Machines on these features by finding the best hyper parameter as suggested in step 2 and step 3 (<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/) (<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). (<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 [29]:

```
#Separating features and label column
Y = project_data['project_is_approved']
X = project_data.drop(['project_is_approved', 'id'],axis=1)
print("Shape of X: ",X.shape)
print("Shape of Y: ",Y.shape)
```

Shape of X: (109248, 12)
Shape of Y: (109248,)

In [30]:

```
#separating data into train and test
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.30,stratify=Y)
print("Shape of X_train: ", X_train.shape)
print("Shape of Y_train: ",Y_train.shape)
print("Shape of X_test: ",X_test.shape)
print("Shape of Y_test: ",Y_test.shape)
```

```
Shape of X_train: (76473, 12)
Shape of Y_train: (76473,)
Shape of X_test: (32775, 12)
Shape of Y_test: (32775,)
```

In [31]:

```
X_train.columns
```

Out[31]:

```
Index(['school_state', 'project_submitted_datetime',
      'teacher_number_of_previously_posted_projects', 'price', 'quantity',
      'clean_categories', 'clean_subcategories', 'essay', 'clean_essay',
      'clean_title', 'clean_teacher_prefix', 'clean_grade_category'],
      dtype='object')
```

2.2 Make Data Model Ready: encoding numerical, categorical features

2.2.1 Encoding Categorical Features

One hot encoding: clean_categories

In [32]:

```
from collections import Counter
my_counter = Counter()
for word in X_train['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]))
```

In [33]:

```
# we use count vectorizer to convert the values into one
vectorizer_category = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True)
vectorizer_category.fit(X_train['clean_categories'].values)

X_train_category_ohe = vectorizer_category.transform(X_train['clean_categories'].values)
X_test_category_ohe = vectorizer_category.transform(X_test['clean_categories'].values)
```

In [34]:

```
print(vectorizer_category.get_feature_names())
print("Shape of X_train after one hot encoding ", X_train_category_ohe.shape)
print("Shape of X_test after one hot encoding ", X_test_category_ohe.shape)
print("Print some random encoded categories: ")
print(X_train_category_ohe[0].toarray())
print(X_test_category_ohe[15].toarray())
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
```

```
Shape of X_train after one hot encoding (76473, 9)
```

```
Shape of X_test after one hot encoding (32775, 9)
```

```
Print some random encoded categories:
```

```
[[0 0 0 0 0 0 0 0 1]]
```

```
[[0 0 0 0 1 0 0 1 0]]
```

One hot encoding: clean_subcategories

In [35]:

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

In [36]:

```
# we use count vectorizer to convert the values into one
vectorizer_subcategory = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=True)
vectorizer_subcategory.fit(X_train['clean_subcategories'].values)

X_train_subcategory_ohe = vectorizer_subcategory.transform(X_train['clean_subcategories'].values)
X_test_subcategory_ohe = vectorizer_subcategory.transform(X_test['clean_subcategories'].values)
```

```
print(vectorizer_subcategory.get_feature_names())
print("Shape of X_train subcategory after one hot encoding ",X_train_subcategory_oh
e.shape)
print("Shape of X_test subcategory after one hot encoding ",X_test_subcategory_oh
e.shape)
print("Print some random encoded categories: ")
print(X_train_subcategory_oh[0].toarray())
print(X_test_subcategory_oh[10].toarray())
```

One hot encoding: school_state

```
# create a vocabulary for states
unique_states = np.unique(X_train['school_state'].values)

vectorizer_state = CountVectorizer(vocabulary=unique_states, lowercase=False, binary=True)
vectorizer_state.fit(X_train['school_state'].values)

X_train_school_state_ohe = vectorizer_state.transform(X_train['school_state'].values)
X_test_school_state_ohe = vectorizer_state.transform(X_test['school_state'].values)
```

```
print(vectorizer_state.get_feature_names())
print("Shape of X_train school_state after one hot encoding ",X_train_school_state_
e_oh.shape)
print("Shape of X_test school_state after one hot encoding ",X_test_school_state_
oh.shape)
print("Print some random encoded school_state: ")
print(X_train_school_state_oh[0].toarray())
print(X_test_school_state_oh[15].toarray())
```

```
Shape of X_train school_state after one hot encoding (76473, 51)
Shape of X_test school_state after one hot encoding (32775, 51)
Print some random encoded school state:
```

In [40]:

```
unique_teacher_prefix = np.unique(X_train['clean_teacher_prefix'])

vectorizer_teacher_prefix = CountVectorizer(vocabulary=unique_teacher_prefix, lowercase=False, binary=True)
vectorizer_teacher_prefix.fit(X_train['clean_teacher_prefix'].values)

X_train_teacher_prefix_ohe = vectorizer_teacher_prefix.transform(X_train['clean_teacher_prefix'].values)
X_test_teacher_prefix_ohe = vectorizer_teacher_prefix.transform(X_test['clean_teacher_prefix'].values)
```

In [41]:

```
print(vectorizer_teacher_prefix.get_feature_names())
print("Shape of X_train clean_teacher_prefix after one hot encoding ",X_train_teacher_prefix_ohe.shape)
print("Shape of X_test clean_teacher_prefix after one hot encoding ",X_test_teacher_prefix_ohe.shape)
print("Print some random encoded clean_teacher_prefix: ")
print(X_train_teacher_prefix_ohe[0].toarray())
print(X_test_teacher_prefix_ohe[15].toarray())
```

```
['Dr', 'Mr', 'Mrs', 'Ms', 'Teacher', 'none']
Shape of X_train clean_teacher_prefix after one hot encoding (76473,
6)
Shape of X_test clean_teacher_prefix after one hot encoding (32775,
6)
Print some random encoded clean_teacher_prefix:
[[0 0 0 0 1 0]]
[[0 0 0 1 0 0]]
```

One hot encoding: project_grade_category

In [42]:

```
unique_grades = np.unique(X_train['clean_grade_category'])

vectorizer_grade = CountVectorizer(vocabulary=unique_grades,lowercase=False,binary=True)
vectorizer_grade.fit(X_train['clean_grade_category'].values)

X_train_grade_category_ohe = vectorizer_grade.transform(X_train['clean_grade_category'].values)
X_test_grade_category_ohe = vectorizer_grade.transform(X_test['clean_grade_category'].values)
```

In [43]:

```
print(vectorizer_grade.get_feature_names())
print("Shape of X_train clean_grade_category after one hot encoding ",X_train_grade_category_ohe.shape)
print("Shape of X_test clean_grade_category after one hot encoding ",X_test_grade_category_ohe.shape)
print("Print some random encoded clean_grade_category: ")
print(X_train_grade_category_ohe[0].toarray())
print(X_test_grade_category_ohe[15].toarray())
```

```
['Grades_3_5', 'Grades_6_8', 'Grades_9_12', 'Grades_PreK_2']
Shape of X_train clean_grade_category after one hot encoding (76473,
4)
Shape of X_test clean_grade_category after one hot encoding (32775,
4)
Print some random encoded clean_grade_category:
[[0 0 0 1]]
[[1 0 0 0]]
```


2.2.2 Encoding Numerical features

Normalizing Price

In [44]:

```
price_vectorizer = preprocessing.Normalizer().fit(X_train['price'].values.reshape(1,-1))
```

In [45]:

```
X_train_price_normalized = price_vectorizer.transform(X_train['price'].values.reshape(1,-1))
X_test_price_normalized = price_vectorizer.transform(X_test['price'].values.reshape(1, -1))
```

In [46]:

```
X_train_price_normalized
```

Out[46]:

```
array([[0.00300388, 0.00015313, 0.00066094, ..., 0.00230623, 0.00053
9      ,
        0.00012193]])
```

In [47]:

```
X_test_price_normalized
```

Out[47]:

```
array([[6.95566263e-04, 2.41088670e-03, 2.63640368e-03, ...,
        5.39826206e-03, 9.66963488e-03, 6.08721934e-05]])
```

Normalize teacher_number_of_previously_posted_projects

In [48]:

```
project_vectorizer = preprocessing.Normalizer().fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
```

In [49]:

```
X_train_normal_previous_project = project_vectorizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1, -1))
X_test_normal_previous_project = project_vectorizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(1, -1))
```

2.3 Make Data Model Ready: encoding eassay, and project_title

2.3.1 Bag of words : Essay

In [50]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer_essay_bow = CountVectorizer(min_df=10,ngram_range=(1,2), max_features=5000)
vectorizer_essay_bow.fit(X_train['clean_essay'])
```

Out[50]:

```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                lowercase=True, max_df=1.0, max_features=5000, min_df=10,
                ngram_range=(1, 2), preprocessor=None, stop_words=None,
                strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, vocabulary=None)
```

In [51]:

```
X_train_essay_bow = vectorizer_essay_bow.transform(X_train['clean_essay'])
X_test_essay_bow = vectorizer_essay_bow.transform(X_test['clean_essay'])

print("Shape of X_train_essay_bow ",X_train_essay_bow.shape)
print("Shape of X_test_essay_bow ",X_test_essay_bow.shape)
```

```
Shape of X_train_essay_bow (76473, 5000)
Shape of X_test_essay_bow (32775, 5000)
```

2.3.2 Bag of words : Project Title

In [52]:

```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
vectorizer_title_bow = CountVectorizer(min_df=10,ngram_range=(1,2), max_features=5000)
vectorizer_title_bow.fit(X_train['clean_title'])
```

Out[52]:

```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                lowercase=True, max_df=1.0, max_features=5000, min_df=10,
                ngram_range=(1, 2), preprocessor=None, stop_words=None,
                strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, vocabulary=None)
```

In [53]:

```
X_train_title_bow = vectorizer_title_bow.transform(X_train['clean_title'])
X_test_title_bow = vectorizer_title_bow.transform(X_test['clean_title'])

print("Shape of X_train_title_bow ",X_train_title_bow.shape)
print("Shape of X_test_title_bow ",X_test_title_bow.shape)
```

```
Shape of X_train_title_bow (76473, 4887)
Shape of X_test_title_bow (32775, 4887)
```

2.3.3 TFIDF vectorizer: Essay

In [54]:

```
vectorizer_essay_tfidf = TfidfVectorizer(min_df=10,ngram_range=(1,2), max_features=5000)
vectorizer_essay_tfidf.fit(X_train['clean_essay'])
```

Out[54]:

```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.float64'>, encoding='utf-8', input='content',
                lowercase=True, max_df=1.0, max_features=5000, min_df=10,
                ngram_range=(1, 2), norm='l2', preprocessor=None, smooth_idf=True,
                stop_words=None, strip_accents=None, sublinear_tf=False,
                token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
                vocabulary=None)
```

In [55]:

```
X_train_essay_tfidf = vectorizer_essay_tfidf.transform(X_train['clean_essay'])
X_test_essay_tfidf = vectorizer_essay_tfidf.transform(X_test['clean_essay'])

print("Shape of X_train_essay_tfidf ",X_train_essay_tfidf.shape)
print("Shape of X_test_essay_tfidf ",X_test_essay_tfidf.shape)
```

```
Shape of X_train_essay_tfidf (76473, 5000)
Shape of X_test_essay_tfidf (32775, 5000)
```

2.3.4 TFIDF vectorizer: Project title

In [56]:

```
vectorizer_title_tfidf = TfidfVectorizer(min_df=10,ngram_range=(1,2), max_features=5000)
vectorizer_title_tfidf.fit(X_train['clean_title'])
```

Out[56]:

```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.float64'>, encoding='utf-8', input='content',
                lowercase=True, max_df=1.0, max_features=5000, min_df=10,
                ngram_range=(1, 2), norm='l2', preprocessor=None, smooth_idf=True,
                stop_words=None, strip_accents=None, sublinear_tf=False,
                token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
                vocabulary=None)
```

In [57]:

```
X_train_title_tfidf = vectorizer_title_tfidf.transform(X_train['clean_title'])
X_test_title_tfidf = vectorizer_title_tfidf.transform(X_test['clean_title'])

print("Shape of X_train_title_tfidf ",X_train_title_tfidf.shape)
print("Shape of X_test_title_tfidf",X_test_title_tfidf.shape)
```

```
Shape of X_train_title_tfidf (76473, 4887)
Shape of X_test_title_tfidf (32775, 4887)
```

2.3.5 Using Pretrained Models: Avg W2V : Essay

In [58]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-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 [59]:

```
# average Word2Vec
def get_avg_w2v(corpus):
    avg_w2v_vectors=[]
    for sentence in tqdm(corpus): # for each review/sentence
        vector = np.zeros(300) # as word vectors are of zero length
        cnt_words =0; # num of words with a valid vector in the sentence/review
        for word in sentence.split(): # for each word in a review/sentence
            if word in glove_words:
                vector += model[word]
                cnt_words += 1
        if cnt_words != 0:
            vector /= cnt_words
        avg_w2v_vectors.append(vector)
    return avg_w2v_vectors
```

```
X_train_essay_avg_w2v_vectors = get_avg_w2v(X_train['clean_essay'])
X_test_essay_avg_w2v_vectors = get_avg_w2v(X_test['clean_essay'])
```

```
100%|██████████| 76473/76473 [00:24<00:00, 3103.09it/s]
100%|██████████| 32775/32775 [00:10<00:00, 3210.26it/s]
```

In [60]:

```
print("Shape of X_train_essay_avg_w2v_vectors",len(X_train_essay_avg_w2v_vectors
),len(X_train_essay_avg_w2v_vectors[0]))
print("Shape of X_test_essay_avg_w2v_vectors ",len(X_test_essay_avg_w2v_vectors
),len(X_test_essay_avg_w2v_vectors[0]))
```

```
Shape of X_train_essay_avg_w2v_vectors 76473 300
Shape of X_test_essay_avg_w2v_vectors 32775 300
```

2.3.6 Using Pretrained Models: Avg W2V : Project Title

In [61]:

```
X_train_title_avg_w2v_vectors = get_avg_w2v(X_train['clean_title'])
X_test_title_avg_w2v_vectors = get_avg_w2v(X_test['clean_title'])
```

```
100%|██████████| 76473/76473 [00:01<00:00, 56951.97it/s]
100%|██████████| 32775/32775 [00:00<00:00, 59328.26it/s]
```

2.3.7 Using Pretrained Models: TFIDF weighted W2V : Essay

In [62]:

```
def get_tfidf_weighted_w2v(corpus,dictionary,tfidf_words):
    tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in
    this list
    for sentence in tqdm(corpus): # 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/rev
        iew
        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.spl
                it())) # getting the tfidf value for each word
                vector += (vec * tf_idf) # calculating tfidf weighted w2v
                tf_idf_weight += tf_idf
            if tf_idf_weight != 0:
                vector /= tf_idf_weight
        tfidf_w2v_vectors.append(vector)
    return tfidf_w2v_vectors
```

In [63]:

```
dictionary = dict(zip(vectorizer_essay_tfidf.get_feature_names(), list(vectorize
r_essay_tfidf.idf_)))
tfidf_words = set(vectorizer_essay_tfidf.get_feature_names())

X_train_essay_tfidf_w2v_vectors = get_tfidf_weighted_w2v(X_train['clean_essay'].
values,dictionary,tfidf_words)
X_test_essay_tfidf_w2v_vectors = get_tfidf_weighted_w2v(X_test['clean_essay'].va
lues,dictionary,tfidf_words)
```

```
100%|██████████| 76473/76473 [02:05<00:00, 609.48it/s]
100%|██████████| 32775/32775 [00:53<00:00, 610.45it/s]
```

In [64]:

```
print("Shape of X_train_essay_tfidf_w2v_vectors",len(X_train_essay_tfidf_w2v_vec
tors),len(X_train_essay_tfidf_w2v_vectors[0]))
print("Shape of X_test_essay_tfidf_w2v_vectors ",len(X_test_essay_tfidf_w2v_vect
ors),len(X_test_essay_tfidf_w2v_vectors[0]))
```

```
Shape of X_train_essay_tfidf_w2v_vectors 76473 300
Shape of X_test_essay_tfidf_w2v_vectors 32775 300
```

2.3.7 Using Pretrained Models: TFIDF weighted W2V : Project Title

In [65]:

```
dictionary = dict(zip(vectorizer_title_tfidf.get_feature_names(), list(vectorizer_title_tfidf.idf_)))
tfidf_words = set(vectorizer_title_tfidf.get_feature_names())

X_train_title_tfidf_w2v_vectors = get_tfidf_weighted_w2v(X_train['clean_title'], dictionary, tfidf_words)
X_test_title_tfidf_w2v_vectors = get_tfidf_weighted_w2v(X_test['clean_title'], dictionary, tfidf_words)

print("Shape of X_train_title_tfidf_w2v_vectors", len(X_train_title_tfidf_w2v_vectors), len(X_train_title_tfidf_w2v_vectors[0]))
print("Shape of X_test_title_tfidf_w2v_vectors ", len(X_test_title_tfidf_w2v_vectors), len(X_test_title_tfidf_w2v_vectors[0]))
```

```
100%|██████████| 76473/76473 [00:02<00:00, 29875.26it/s]
100%|██████████| 32775/32775 [00:00<00:00, 33786.78it/s]
```

```
Shape of X_train_title_tfidf_w2v_vectors 76473 300
Shape of X_test_title_tfidf_w2v_vectors 32775 300
```

2.4 Applying SVM on different kind of featurization as mentioned in the instructions

Apply SVM 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 instructions

2.4.1 SET 1 : BOW

In [68]:

```
f1 = X_train_school_state_ohe
f2 = X_train_category_ohe
f3 = X_train_subcategory_ohe
f4 = X_train_grade_category_ohe
f5 = X_train_teacher_prefix_ohe
f6 = np.array(X_train_price_normalized).reshape(-1,1)
f7 = np.array(X_train_normal_previous_project).reshape(-1,1)

X_train_bow = hstack((f1,f2,f3,f4,f5,f6,f7,X_train_essay_bow,X_train_title_bow))
X_train_bow.shape
```

Out[68]:

```
(76473, 9989)
```

In [69]:

```
f1 = X_test_school_state_ohe
f2 = X_test_category_ohe
f3 = X_test_subcategory_ohe
f4 = X_test_grade_category_ohe
f5 = X_test_teacher_prefix_ohe
f6 = np.array(X_test_price_normalized).reshape(-1,1)
f7 = np.array(X_test_normal_previous_project).reshape(-1,1)

X_test_bow = hstack((f1,f2,f3,f4,f5,f6,f7,X_test_essay_bow,X_test_title_bow))
X_test_bow.shape
```

Out[69]:

(32775, 9989)

Hyperparameter Tuning: Lambda

In [70]:

```
tune_parameters = [{'alpha': [10**-4, 10**-2, 10**-1, 10**0, 10, 10**2, 10**3, 10**4], 'penalty': ['l1', 'l2']}]

#Using GridSearchCV
model = GridSearchCV(SGDClassifier(class_weight='balanced'), tune_parameters, scoring = 'roc_auc', cv=10, return_train_score=True, n_jobs=-1, verbose=True)
model.fit(X_train_bow, Y_train)
```

Fitting 10 folds for each of 16 candidates, totalling 160 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.

[Parallel(n_jobs=-1)]: Done 18 tasks | elapsed: 5.8s

[Parallel(n_jobs=-1)]: Done 160 out of 160 | elapsed: 15.1s finished

Out[70]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
                                     early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
                                     l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
                                     n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
                                     power_t=0.5, random_state=None, shuffle=True, tol=None,
                                     validation_fraction=0.1, verbose=0, warm_start=False),
             fit_params=None, iid='warn', n_jobs=-1,
             param_grid=[{'alpha': [0.0001, 0.01, 0.1, 1, 10, 100, 1000, 10000], 'penalty': ['l1', 'l2']}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=True)
```


In [71]:

```
results = pd.DataFrame.from_dict(model.cv_results_)
results = results.sort_values(['param_alpha'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
param_alpha = results['param_alpha']

log_alpha = [(np.log(x)/np.log(10)) for x in param_alpha]
#log_alphas = [np.log(x) for x in alphas]

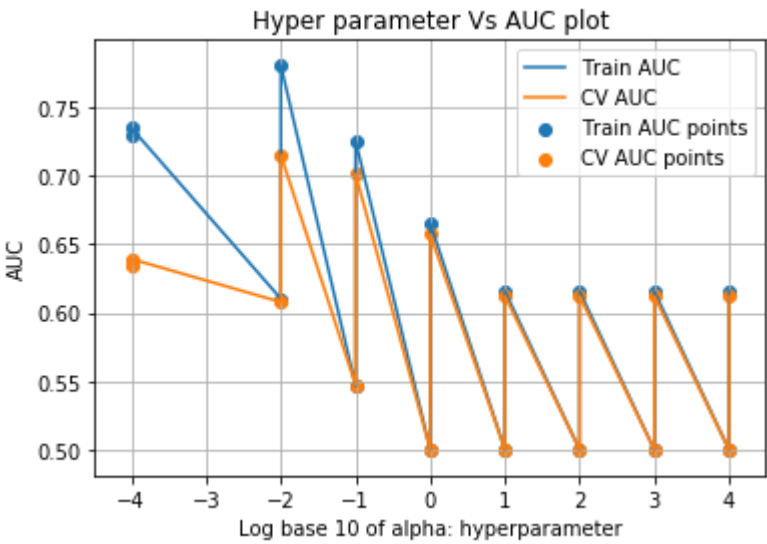
plt.plot(log_alpha, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039

plt.plot(log_alpha, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039

plt.scatter(log_alpha, train_auc, label='Train AUC points')
plt.scatter(log_alpha, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("Log base 10 of alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```



Out[71]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	param_penalty
0	2.843923	0.040551	0.010042	0.000609	0.0001	1
1	1.496724	0.661077	0.009824	0.000665	0.0001	1
2	1.100516	0.017109	0.009675	0.000683	0.01	1
3	0.756755	0.022184	0.009610	0.000640	0.01	1
4	1.126698	0.043440	0.009371	0.000533	0.1	1

5 rows × 32 columns



In [72]:

```
model.best_estimator_
```

Out[72]:

```
SGDClassifier(alpha=0.01, average=False, class_weight='balanced',  
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,  
              l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,  
              n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',  
              power_t=0.5, random_state=None, shuffle=True, tol=None,  
              validation_fraction=0.1, verbose=0, warm_start=False)
```

Observations:

1. We can see both from the graph as well as from the API response that the best value of alpha = 0.01
2. This can be seen as value of -2 on the x-axis of the error vs log alpha plot.

The best value for alpha is 0.01. Training model on optimal value.

In [73]:

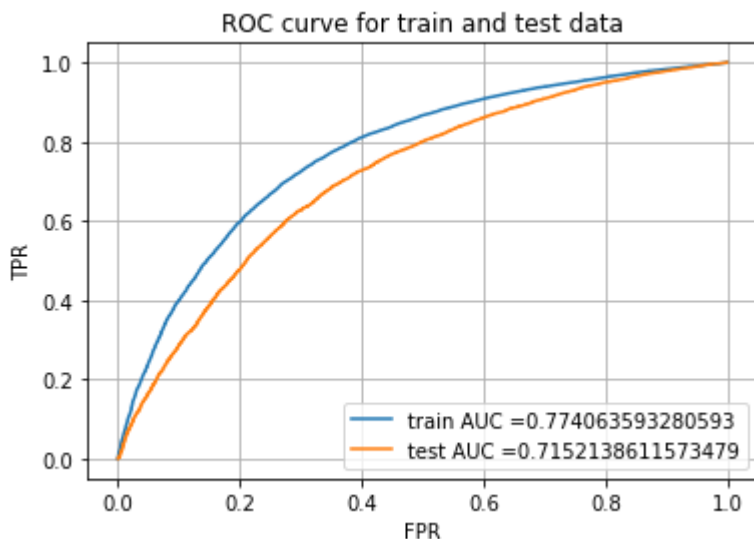
```
svm_bow = SGDClassifier(class_weight='balanced',alpha=0.01,penalty='l2')
svm_bow.fit(X_train_bow,Y_train)

y_train_score = svm_bow.decision_function(X_train_bow)
y_test_score = svm_bow.decision_function(X_test_bow)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs
train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train, y_train_score)
test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, y_test_score)

plt.plot(train_fpr, train_tpr, label="train AUC =" + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()

plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC curve for train and test data")
plt.grid()
plt.show()
```



In [74]:

```
def predict_result(proba, thresholds, tpr,fpr):
    best_threshold = thresholds[np.argmax(tpr-fpr)]
    print("threshold used: ", best_threshold)
    y_predicted = []

    for p in proba:
        if p >= best_threshold:
            y_predicted.append(1)
        else:
            y_predicted.append(0)
    return y_predicted
```

Confusion Matrix

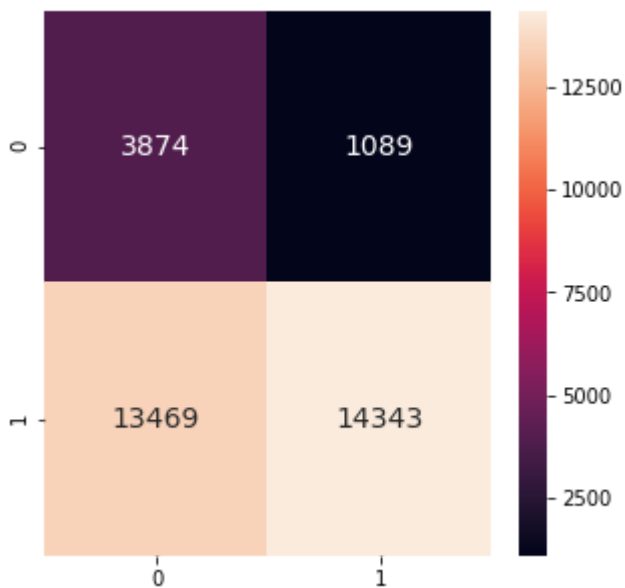
In [75]:

```
y_test_predict = svm_bow.predict(X_test_bow)

results = confusion_matrix(Y_test, y_test_predict)
plt.figure(figsize = (5,5))
sns.heatmap(results, annot=True,annot_kws={"size": 14}, fmt='g')
```

Out[75]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f7a6fa7aa90>



In [76]:

y_test_predict

Out[76]:

array([0, 0, 1, ..., 0, 0, 1])

Confusion Matrix (using custom threshold)

In [77]:

```

y_test_predict = predict_result(y_test_score, te_thresholds, test_tpr, test_fpr)

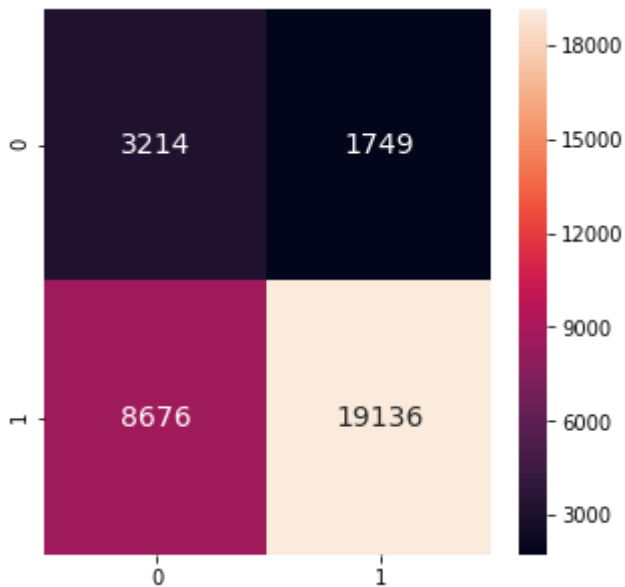
results = confusion_matrix(Y_test, y_test_predict)
plt.figure(figsize = (5,5))
sns.heatmap(results, annot=True, annot_kws={"size": 14}, fmt='g')

```

threshold used: -0.3958908879442399

Out[77]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f7a56c63160>



2.4.2 SET 2 : TFIDF

In [78]:

```

f1 = X_train_school_state_ohe
f2 = X_train_category_ohe
f3 = X_train_subcategory_ohe
f4 = X_train_grade_category_ohe
f5 = X_train_teacher_prefix_ohe
f6 = np.array(X_train_price_normalized).reshape(-1,1)
f7 = np.array(X_train_normal_previous_project).reshape(-1,1)

X_train_tfidf = hstack((f1,f2,f3,f4,f5,f6,f7,X_train_essay_tfidf,X_train_title_tfidf))
X_train_tfidf.shape

```

Out[78]:

(76473, 9989)

In [79]:

```
f1 = X_test_school_state_ohe
f2 = X_test_category_ohe
f3 = X_test_subcategory_ohe
f4 = X_test_grade_category_ohe
f5 = X_test_teacher_prefix_ohe
f6 = X_test_price_normalized.reshape(-1,1)
f7 = X_test_normal_previous_project.reshape(-1,1)

X_test_tfidf = hstack((f1,f2,f3,f4,f5,f6,f7,X_test_essay_tfidf,X_test_title_tfidf))
X_test_tfidf.shape
```

Out[79]:

(32775, 9989)

Hyperparameter Tuning: Lambda

In [80]:

```
tune_parameters = {'alpha': [10**-4, 10**-2, 10**-1, 10**0, 10, 10**2, 10**3, 10**4], 'penalty': ['l1', 'l2']}

#Using GridSearchCV
model = GridSearchCV(SGDClassifier(class_weight='balanced'), tune_parameters, scoring = 'roc_auc', cv=10, return_train_score=True, n_jobs=-1, verbose=True)
model.fit(X_train_tfidf, Y_train)
```

Fitting 10 folds for each of 16 candidates, totalling 160 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done 18 tasks      | elapsed:    1.9s
[Parallel(n_jobs=-1)]: Done 160 out of 160 | elapsed:   11.1s finished
```

Out[80]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
                                     early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
                                     l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
                                     n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
                                     power_t=0.5, random_state=None, shuffle=True, tol=None,
                                     validation_fraction=0.1, verbose=0, warm_start=False),
             fit_params=None, iid='warn', n_jobs=-1,
             param_grid={'alpha': [0.0001, 0.01, 0.1, 1, 10, 100, 1000, 10000], 'penalty': ['l1', 'l2']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=True)
```

In [82]:

```
results = pd.DataFrame.from_dict(model.cv_results_)
results = results.sort_values(['param_alpha'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
param_alpha = results['param_alpha']

log_alpha = [(np.log(x)/np.log(10)) for x in param_alpha]
#log_alphas = [np.log(x) for x in alphas]

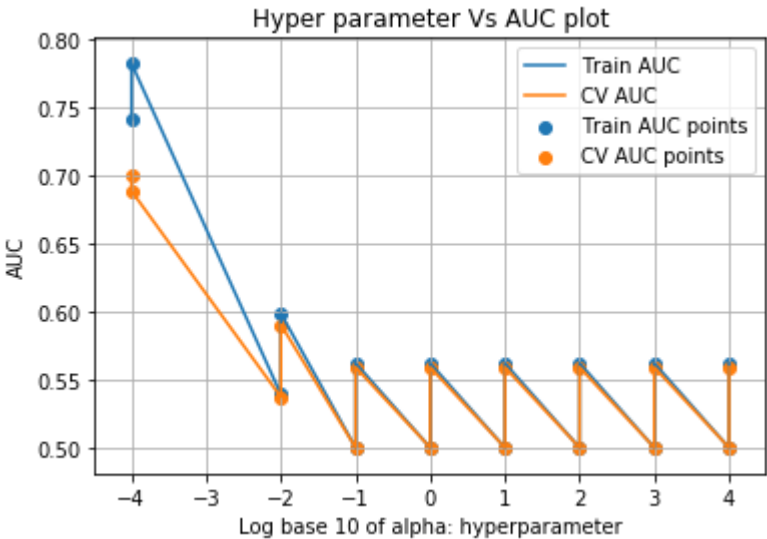
plt.plot(log_alpha, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039

plt.plot(log_alpha, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039

plt.scatter(log_alpha, train_auc, label='Train AUC points')
plt.scatter(log_alpha, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("Log base 10 of alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```

Out[82]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	param_penalty
0	1.120494	0.102787	0.009350	0.001561	0.0001	l1
1	0.768686	0.086943	0.009110	0.001250	0.0001	l1
2	0.969084	0.072445	0.007591	0.001360	0.01	l1
3	0.772738	0.077398	0.008968	0.001003	0.01	l1
4	1.134860	0.030906	0.008317	0.000495	0.1	l1

5 rows × 32 columns

In [83]:

```
model.best_estimator_
```

Out[83]:

```
SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=
              None,
              n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l1',
              power_t=0.5, random_state=None, shuffle=True, tol=None,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

Observations:

1. We can see both from the graph as well as from the API response that the best value of $\alpha = 0.0001$.
2. This can be seen as value of -4 on the x-axis of the error vs log alpha plot.

Training the model on the most optimal value of $\alpha=0.0001$

In [84]:

```

svm_tfidf = SGDClassifier(class_weight='balanced',alpha=0.0001,penalty='l2')
svm_tfidf.fit(X_train_tfidf,Y_train)

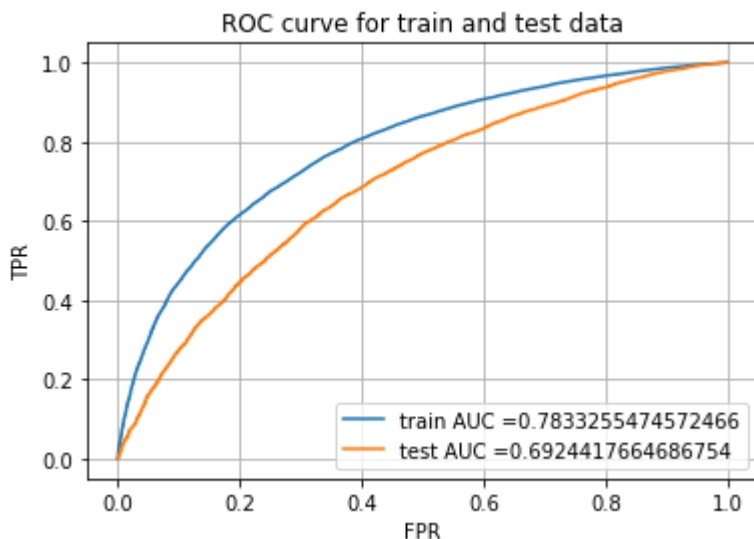
y_train_pred = svm_tfidf.decision_function(X_train_tfidf)
y_test_pred = svm_tfidf.decision_function(X_test_tfidf)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs
train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC =" + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()

plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC curve for train and test data")
plt.grid()
plt.show()

```



In [85]:

```
te_thresholds[np.argmax(test_tpr - test_fpr)]
```

Out[85]:

```
-0.31201136891747966
```

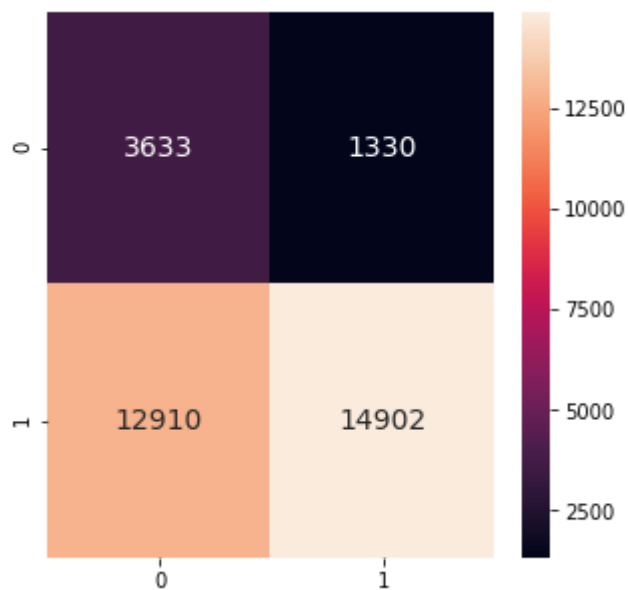
Confusion Matrix

In [86]:

```
y_test_predict = svm_tfidf.predict(X_test_tfidf)
results = confusion_matrix(Y_test, y_test_predict)
plt.figure(figsize = (5,5))
sns.heatmap(results, annot=True,annot_kws={"size": 14}, fmt='g')
```

Out[86]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f7a8cfbaba8>



Confusion Matrix using custom threshold from ROC curve

In [87]:

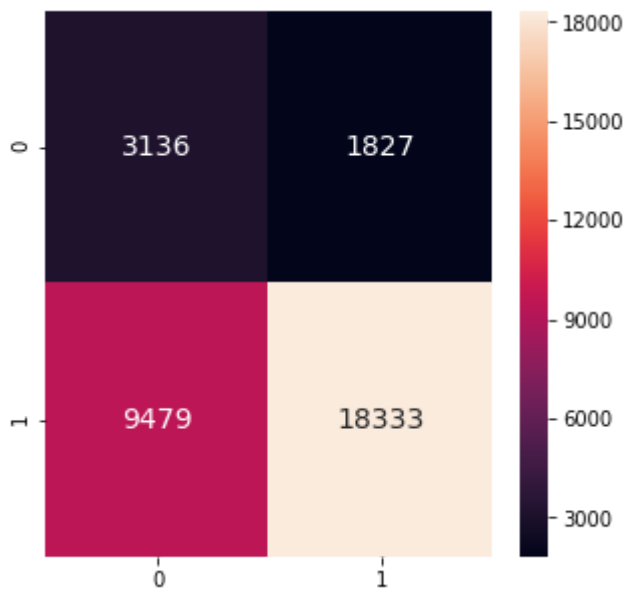
```
y_test_predict = predict_result(y_test_pred,te_thresholds,test_tpr,test_fpr)

results = confusion_matrix(Y_test, y_test_predict)
plt.figure(figsize = (5,5))
sns.heatmap(results, annot=True,annot_kws={"size": 14}, fmt='g')
```

threshold used: -0.31201136891747966

Out[87]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f7a7573b160>



2.4.3 SET 3 : W2Vec

In [88]:

```
f1 = X_train_school_state_ohe
f2 = X_train_category_ohe
f3 = X_train_subcategory_ohe
f4 = X_train_grade_category_ohe
f5 = X_train_teacher_prefix_ohe
f6 = np.array(X_train_price_normalized).reshape(-1,1)
f7 = np.array(X_train_normal_previous_project).reshape(-1,1)

X_train_w2v = hstack((f1,f2,f3,f4,f5,f6,f7,X_train_essay_avg_w2v_vectors,X_train_title_avg_w2v_vectors))
X_train_w2v.shape
```

Out[88]:

(76473, 702)

In [89]:

```
f1 = X_test_school_state_ohe
f2 = X_test_category_ohe
f3 = X_test_subcategory_ohe
f4 = X_test_grade_category_ohe
f5 = X_test_teacher_prefix_ohe
f6 = X_test_price_normalized.reshape(-1,1)
f7 = X_test_normal_previous_project.reshape(-1,1)

X_test_w2v = hstack((f1,f2,f3,f4,f5,f6,f7,X_test_essay_avg_w2v_vectors,X_test_title_avg_w2v_vectors))
X_test_w2v.shape
```

Out[89]:

(32775, 702)

Hyperparameter Tuning: Lambda

In [90]:

```
tune_parameters = {'alpha': [10**-4, 10**-2, 10**-1, 10**0, 10, 10**2, 10**3, 10**4], 'penalty': ['l1', 'l2']}
```

#Using GridSearchCV

```
model = GridSearchCV(SGDClassifier(class_weight='balanced'), tune_parameters, scoring = 'roc_auc', cv=10, return_train_score=True, n_jobs=-1, verbose=True)
model.fit(X_train_w2v, Y_train)
```

Fitting 10 folds for each of 16 candidates, totalling 160 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.

[Parallel(n_jobs=-1)]: Done 18 tasks | elapsed: 11.9s

[Parallel(n_jobs=-1)]: Done 160 out of 160 | elapsed: 45.8s finished

Out[90]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
                                     early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
                                     l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
                                     n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
                                     power_t=0.5, random_state=None, shuffle=True, tol=None,
                                     validation_fraction=0.1, verbose=0, warm_start=False),
             fit_params=None, iid='warn', n_jobs=-1,
             param_grid={'alpha': [0.0001, 0.01, 0.1, 1, 10, 100, 1000, 10000], 'penalty': ['l1', 'l2']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=True)
```

In [91]:

```
results = pd.DataFrame.from_dict(model.cv_results_)
results = results.sort_values(['param_alpha'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
param_alpha = results['param_alpha']

log_alpha = [(np.log(x)/np.log(10)) for x in param_alpha]
#log_alphas = [np.log(x) for x in alphas]

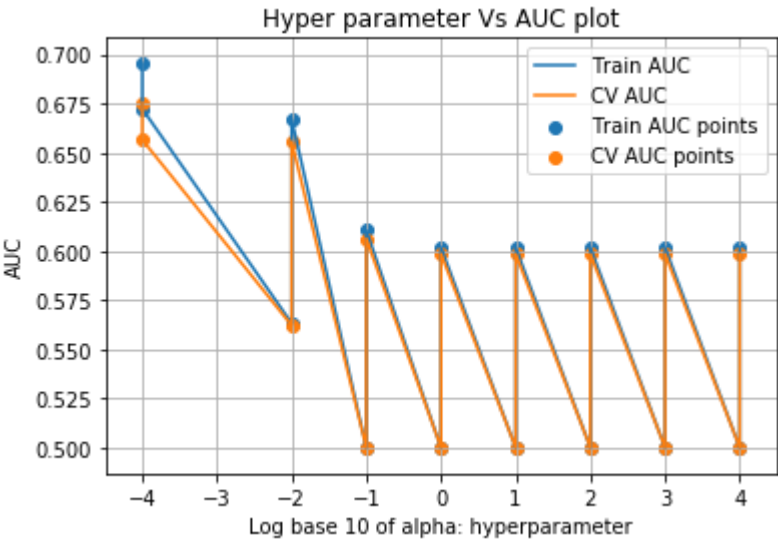
plt.plot(log_alpha, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039

plt.plot(log_alpha, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039

plt.scatter(log_alpha, train_auc, label='Train AUC points')
plt.scatter(log_alpha, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("Log base 10 of alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```



Out[91]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	param_penalty
0	10.682227	0.068548	0.022444	0.001712	0.0001	l1
1	6.105872	3.318567	0.022435	0.001591	0.0001	l1
2	4.837917	0.033814	0.022546	0.001560	0.01	l1
3	2.269946	0.042570	0.022101	0.001288	0.01	l1
4	4.973264	0.040999	0.021673	0.001826	0.1	l1

5 rows × 32 columns

In [93]:

```
model.best_estimator_
```

Out[93]:

```
SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
              n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l1',
              power_t=0.5, random_state=None, shuffle=True, tol=None,
              validation_fraction=0.1, verbose=0, warm_start=False)
```


Observations:

1. We can see both from the graph as well as from the API response that the best value of $\alpha = 0.0001$.
2. This can be seen as value of 4 on the x-axis of the error vs logC plot.

Training the model on the most optimal value of $\alpha=0.0001$

In [94]:

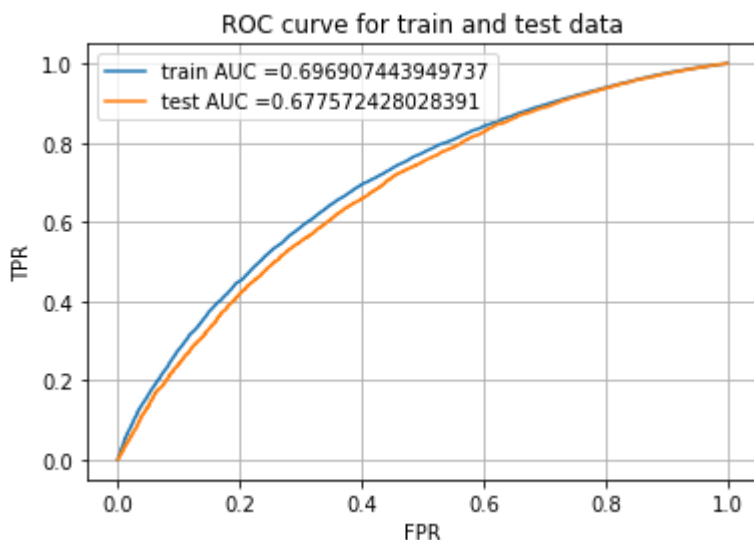
```
svm_w2v = SGDClassifier(class_weight='balanced',alpha=0.0001,penalty='l1')
svm_w2v.fit(X_train_w2v,Y_train)

y_train_pred = svm_w2v.decision_function(X_train_w2v)
y_test_pred = svm_w2v.decision_function(X_test_w2v)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs
train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC =" + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()

plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC curve for train and test data")
plt.grid()
plt.show()
```



In [95]:

```
te_thresholds[np.argmax(test_tpr-test_fpr)]
```

Out[95]:

```
-0.09012564491204611
```

Confusion Matrix

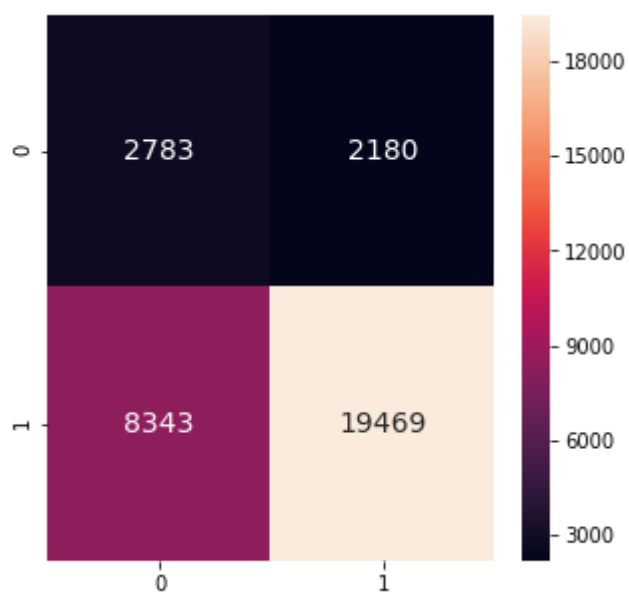
In [96]:

```
y_test_predict = svm_w2v.predict(X_test_w2v)

results = confusion_matrix(Y_test, y_test_predict)
plt.figure(figsize = (5,5))
sns.heatmap(results, annot=True,annot_kws={"size": 14}, fmt='g')
```

Out[96]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f7a47b47fd0>



Creating confusion matrix with custom threshold

In [97]:

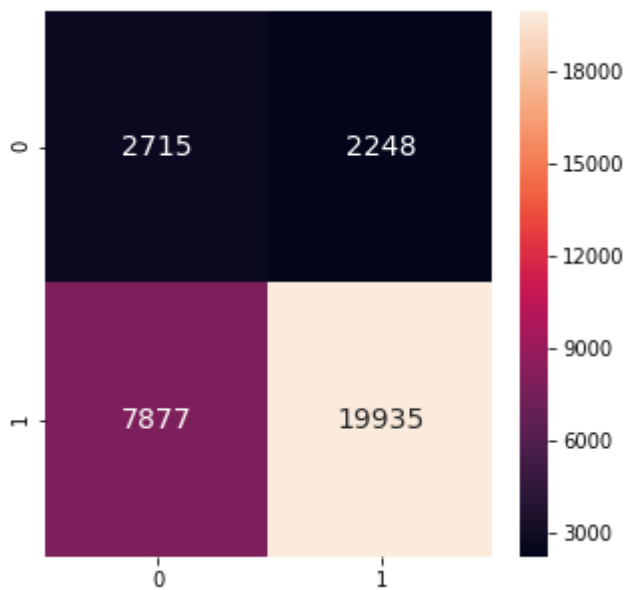
```
y_test_predict = predict_result(y_test_pred, te_thresholds, test_tpr, test_fpr)

results = confusion_matrix(Y_test, y_test_predict)
plt.figure(figsize = (5,5))
sns.heatmap(results, annot=True, annot_kws={"size": 14}, fmt='g')
```

threshold used: -0.09012564491204611

Out[97]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f7a7f92c9b0>



2.4.4 SET 4 : TFIDF-weighted W2Vec

In [98]:

```
f1 = X_train_school_state_ohe
f2 = X_train_category_ohe
f3 = X_train_subcategory_ohe
f4 = X_train_grade_category_ohe
f5 = X_train_teacher_prefix_ohe
f6 = np.array(X_train_price_normalized).reshape(-1,1)
f7 = np.array(X_train_normal_previous_project).reshape(-1,1)

X_train_tfidf_w2v = hstack((f1,f2,f3,f4,f5,f6,f7,X_train_essay_tfidf_w2v_vectors,
,X_train_title_tfidf_w2v_vectors))
X_train_tfidf_w2v.shape
```

Out[98]:

(76473, 702)

In [99]:

```
f1 = X_test_school_state_ohe
f2 = X_test_category_ohe
f3 = X_test_subcategory_ohe
f4 = X_test_grade_category_ohe
f5 = X_test_teacher_prefix_ohe
f6 = X_test_price_normalized.reshape(-1,1)
f7 = X_test_normal_previous_project.reshape(-1,1)

X_test_tfidf_w2v = hstack((f1,f2,f3,f4,f5,f6,f7,X_test_essay_tfidf_w2v_vectors,X
_test_title_tfidf_w2v_vectors))
X_test_tfidf_w2v.shape
```

Out[99]:

(32775, 702)

In [100]:

```
tune_parameters = {'alpha': [10**-4, 10**-2, 10**-1, 10**0, 10, 10**2, 10**3, 10**4], 'penalty': ['l1', 'l2']}
```

#Using GridSearchCV

```
model = GridSearchCV(SGDClassifier(class_weight='balanced'), tune_parameters, scoring = 'roc_auc', cv=10, return_train_score=True, n_jobs=-1, verbose=True)
model.fit(X_train_tfidf_w2v, Y_train)
```

Fitting 10 folds for each of 16 candidates, totalling 160 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.

[Parallel(n_jobs=-1)]: Done 18 tasks | elapsed: 5.8s

[Parallel(n_jobs=-1)]: Done 160 out of 160 | elapsed: 39.4s finished

Out[100]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
                                     early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
                                     l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
                                     n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
                                     power_t=0.5, random_state=None, shuffle=True, tol=None,
                                     validation_fraction=0.1, verbose=0, warm_start=False),
             fit_params=None, iid='warn', n_jobs=-1,
             param_grid={'alpha': [0.0001, 0.01, 0.1, 1, 10, 100, 1000, 10000], 'penalty': ['l1', 'l2']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=True)
```

In [101]:

```
results = pd.DataFrame.from_dict(model.cv_results_)
results = results.sort_values(['param_alpha'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
param_alpha = results['param_alpha']

log_alpha = [(np.log(x)/np.log(10)) for x in param_alpha]
#log_alphas = [np.log(x) for x in alphas]

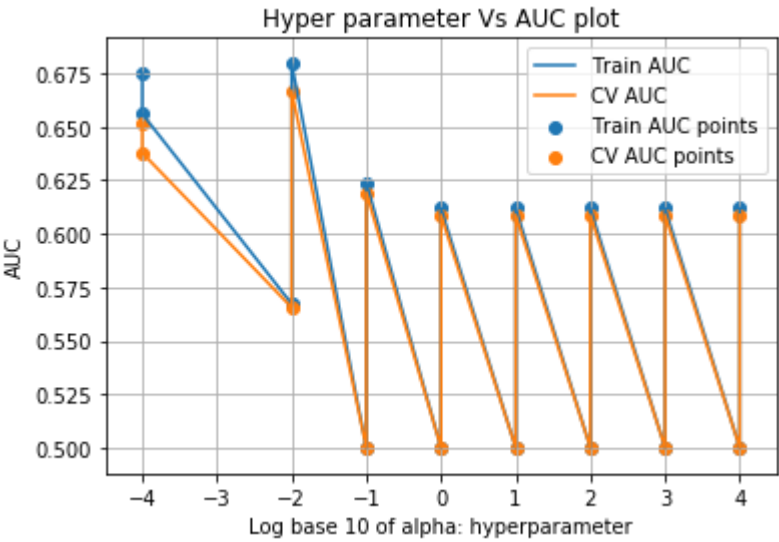
plt.plot(log_alpha, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039

plt.plot(log_alpha, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039

plt.scatter(log_alpha, train_auc, label='Train AUC points')
plt.scatter(log_alpha, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("Log base 10 of alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```



Out[101]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	param_penalty
0	4.180806	0.262781	0.022107	0.001295	0.0001	l
1	2.414715	0.340307	0.023009	0.001946	0.0001	l
2	4.814998	0.066456	0.021535	0.000837	0.01	l
3	2.243992	0.030056	0.021768	0.001518	0.01	l
4	4.954298	0.062850	0.021003	0.001562	0.1	l

5 rows × 32 columns

In [102]:

```
model.best_estimator_
```

Out[102]:

```
SGDClassifier(alpha=0.01, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
              n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=None,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

Observations:

1. We can see both from the graph as well as from the API response that the best value of $\alpha = 0.01$.
2. This can be seen as value of -2 on the x-axis of the error vs log alpha plot.

In [107]:

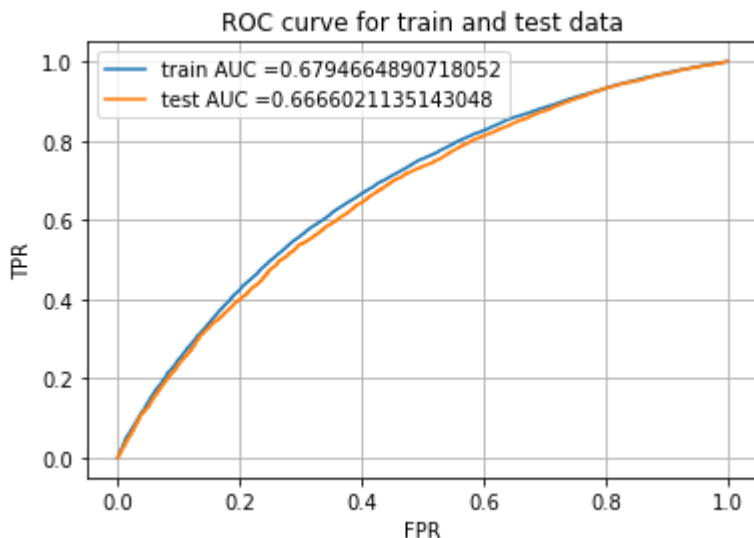
```
svm_tf_idf_w2v = SGDClassifier(class_weight='balanced',alpha=0.01, penalty='l2')
svm_tf_idf_w2v.fit(X_train_tfidf_w2v,Y_train)

y_train_pred = svm_tf_idf_w2v.decision_function(X_train_tfidf_w2v)
y_test_pred = svm_tf_idf_w2v.decision_function(X_test_tfidf_w2v)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs
train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC =" + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()

plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC curve for train and test data")
plt.grid()
plt.show()
```



In [122]:

```
te_thresholds[np.argmax(test_tpr-test_fpr)]
```

Out[122]:

```
-0.4169205726835887
```

Confusion Matrix

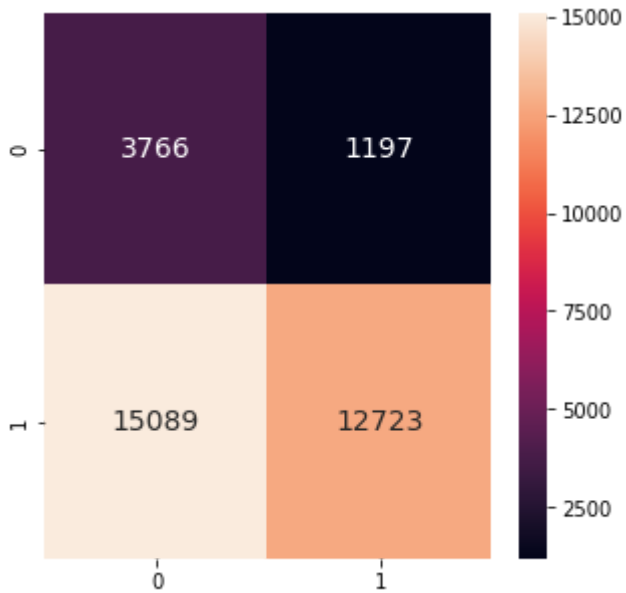
In [109]:

```
y_test_predict = svm_tf_idf_w2v.predict(X_test_tfidf_w2v)

results = confusion_matrix(Y_test, y_test_predict)
plt.figure(figsize = (5,5))
sns.heatmap(results, annot=True,annot_kws={"size": 14}, fmt='g')
```

Out[109]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f7a757e4fd0>



Confusion matrix from custom threshold

In [110]:

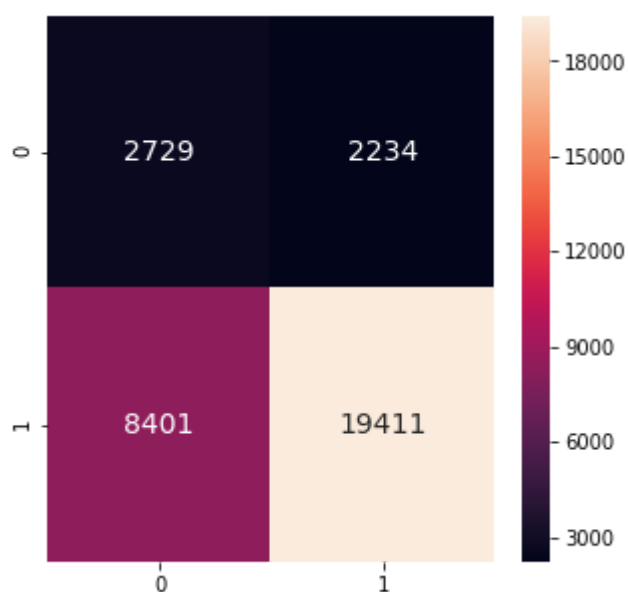
```
y_test_predict = predict_result(y_test_pred, te_thresholds, test_tpr, test_fpr)

results = confusion_matrix(Y_test, y_test_predict)
plt.figure(figsize = (5,5))
sns.heatmap(results, annot=True, annot_kws={"size": 14}, fmt='g')
```

threshold used: -0.4169205726835887

Out[110]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f7a6dac1eb8>



2.5 Logistic Regression with added Features `Set 5`

Adding new numerical features to the data

1. Quantity
2. sentiment score's of each of the essay
3. number of words in the title
4. number of words in the combine essays
5. Consider these set of features **Set 5** :

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

1. Normalizing Quantity

In [111]:

```
quantity_vectorizer = preprocessing.Normalizer().fit(X_train['quantity'].values.
reshape(1, -1))
X_train_qty = quantity_vectorizer.transform(X_train['quantity'].values.reshape(1,
-1))
X_test_qty = quantity_vectorizer.transform(X_test['quantity'].values.reshape(1, -
1))
```

2. Sentiment score of each essay

In [112]:

```
def get_sentiment_score(feature):

    analyser = SentimentIntensityAnalyzer()
    pos = []
    neg = []
    neu = []
    compound = []

    for text in tqdm(feature) :
        score = analyser.polarity_scores(text)
        nt = score['neg']
        pt = score['pos']
        ntr = score['neu']
        cmp = score['compound']
        neg.append(nt)
        pos.append(pt)
        neu.append(ntr)
        compound.append(cmp)

    return (pos,neg,neu,compound)
```

In [113]:

```
X_train_essay_pos, X_train_essay_neg, X_train_essay_neu, X_train_cmp = get_senti
ment_score(X_train['clean_essay'])
```

100%|██████████| 76473/76473 [02:20<00:00, 544.52it/s]

In [114]:

```
X_test_essay_pos, X_test_essay_neg, X_test_essay_neu, X_test_cmp = get_sentiment
_score(X_test['clean_essay'])
```

100%|██████████| 32775/32775 [01:00<00:00, 545.56it/s]

3. Number of words in Title

In [115]:

```
#Reference: https://stackoverflow.com/questions/27488446/how-do-i-get-word-frequ
ency-in-a-corpus-using-scikit-learn-countvectorizer
def get_word_count(feature):
    vectorizer = CountVectorizer()
    vectorizer.fit(feature)
    v0 = vectorizer.transform(feature)
    return np.sum(v0,axis=1)
```

In [116]:

```
X_train_title_wc = get_word_count(X_train['clean_title'])
X_test_title_wc = get_word_count(X_test['clean_title'])
print(X_train_title_wc.shape)
print(X_test_title_wc.shape)
```

```
(76473, 1)
(32775, 1)
```

4. Number of words in combined essays

In [117]:

```
X_train_essay_wc = get_word_count(X_train['clean_essay'])
X_test_essay_wc = get_word_count(X_test['clean_essay'])
print(X_train_essay_wc.shape)
print(X_test_essay_wc.shape)
```

```
(76473, 1)
(32775, 1)
```

Reducing dimensionality of tfidf vectorization of essays using Truncated SVD

In [118]:

```
X_train_essay_tfidf.shape
```

Out[118]:

```
(76473, 5000)
```

In [119]:

```
tsvd = TruncatedSVD(n_components= 4999, n_iter=5, random_state=42)
tsvd.fit(X_train_essay_tfidf)
```

Out[119]:

```
TruncatedSVD(algorithm='randomized', n_components=4999, n_iter=5,
              random_state=42, tol=0.0)
```

In [123]:

```
X_train_tsvd_tfidf_essay_transform = tsvd.transform(X_train_essay_tfidf)
```

In [124]:

```
#https://chrisalbon.com/machine_learning/feature_engineering/select_best_number_
of_components_in_tsvd/
# Create a function
def select_n_components(var_ratio, goal_var: float):
    # Set initial variance explained so far
    total_variance = 0.0

    # Set initial number of features
    n_components = 0

    # For the explained variance of each feature:
    for explained_variance in var_ratio:

        # Add the explained variance to the total
        total_variance += explained_variance

        # Add one to the number of components
        n_components += 1

        # If we reach our goal level of explained variance
        if total_variance >= goal_var:
            # End the loop
            break

    # Return the number of components
    return n_components
```

In [125]:

```
# List of explained variances
tsvd_var_ratios = tsvd.explained_variance_ratio_
select_n_components(tsvd_var_ratios, 0.90)
```

Out[125]:

3173

In [129]:

```
X_train_tsvd_tfidf_essay = X_train_tsvd_tfidf_essay_transform[:,0:3173]
```

In [132]:

```
X_train_tsvd_tfidf_essay.shape
```

Out[132]:

(76473, 3173)

In [131]:

```
X_test_tsvd_tfidf_essay_transform = tsvd.transform(X_test_essay_tfidf)
X_test_tsvd_tfidf_essay = X_test_tsvd_tfidf_essay_transform[:,0:3173]
```

In [135]:

```
tsvd.explained_variance_ratio_[0:1]
```

Out[135]:

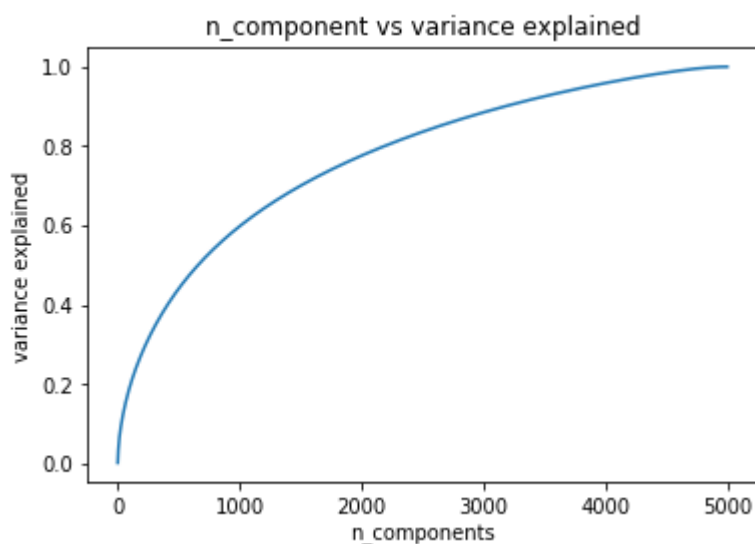
```
array([0.00203552])
```

In [156]:

```
variance_explained = np.cumsum(tsvd.explained_variance_ratio_)
```

In [158]:

```
n_features = np.arange(4999)
plt.plot(n_features, variance_explained)
plt.xlabel("n_components")
plt.ylabel("variance explained")
plt.title("n_component vs variance explained")
plt.show()
```



In [159]:

```
print(variance_explained)
```

```
[0.00203552 0.01172062 0.02007241 ... 0.99999919 0.99999961 0.999999
81]
```

Conclusion: We can see from the curve that 3500 components explain 92% variance. Therefore, we will use `n_components = 3500`.

In [160]:

```
#tsvd = TruncatedSVD(n_components= 3500, n_iter=7, random_state=42)
#tsvd.fit(X_train_essay_tfidf)
# X_train_tsvd_tfidf_essay = tsvd.transform(X_train_essay_tfidf)
# X_test_tsvd_tfidf_essay = tsvd.transform(X_test_essay_tfidf)
print(X_train_tsvd_tfidf_essay.shape)
print(X_test_tsvd_tfidf_essay.shape)
```

(76473, 3173)

(32775, 3173)

In [161]:

```
f1 = X_train_school_state_ohe
f2 = X_train_category_ohe
f3 = X_train_subcategory_ohe
f4 = X_train_grade_category_ohe
f5 = X_train_teacher_prefix_ohe
f6 = np.array(X_train_price_normalized).reshape(-1,1)
f7 = np.array(X_train_normal_previous_project).reshape(-1,1)
f8 = np.array(X_train_qty).reshape(-1,1)
f9 = np.array(X_train_essay_pos).reshape(-1,1)
f10 = np.array(X_train_essay_neg).reshape(-1,1)
f11 = np.array(X_train_essay_neu).reshape(-1,1)
f12 = np.array(X_train_cmp).reshape(-1,1)
f13 = X_train_title_wc
f14 = X_train_essay_wc
f15 = X_train_tsvd_tfidf_essay
X_tr_new = hstack((f1,f2,f3,f4,f5,f6,f7,f8,f9, f10, f11, f12, f13, f14,f15))
X_tr_new.shape
```

Out[161]:

(76473, 3282)

In [162]:

```
f1 = X_test_school_state_ohe
f2 = X_test_category_ohe
f3 = X_test_subcategory_ohe
f4 = X_test_grade_category_ohe
f5 = X_test_teacher_prefix_ohe
f6 = np.array(X_test_price_normalized).reshape(-1,1)
f7 = np.array(X_test_normal_previous_project).reshape(-1,1)
f8 = np.array(X_test_qty).reshape(-1,1)
f9 = np.array(X_test_essay_pos).reshape(-1,1)
f10 = np.array(X_test_essay_neg).reshape(-1,1)
f11 = np.array(X_test_essay_neu).reshape(-1,1)
f12 = np.array(X_test_cmp).reshape(-1,1)
f13 = X_test_title_wc
f14 = X_test_essay_wc
f15 = X_test_tsvd_tfidf_essay
X_test_new = hstack((f1,f2,f3,f4,f5,f6,f7,f8,f9, f10, f11, f12, f13, f14,f15))
X_test_new.shape
```

Out[162]:

(32775, 3282)

In [163]:

```
tune_parameters = {'alpha': [10**-4, 10**-2, 10**-1, 10**0, 10, 10**2, 10**3, 10**4], 'penalty': ['l1', 'l2']}
```

#Using GridSearchCV

```
model = GridSearchCV(SGDClassifier(class_weight='balanced'), tune_parameters, scoring = 'roc_auc', cv=10, return_train_score=True, n_jobs=8, verbose=True)
model.fit(X_train, Y_train)
```

Fitting 10 folds for each of 16 candidates, totalling 160 fits

[Parallel(n_jobs=8)]: Using backend LokyBackend with 8 concurrent workers.

[Parallel(n_jobs=8)]: Done 34 tasks | elapsed: 1.4min

[Parallel(n_jobs=8)]: Done 160 out of 160 | elapsed: 5.2min finished

Out[163]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
                                     early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
                                     l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
                                     n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
                                     power_t=0.5, random_state=None, shuffle=True, tol=None,
                                     validation_fraction=0.1, verbose=0, warm_start=False),
             fit_params=None, iid='warn', n_jobs=8,
             param_grid={'alpha': [0.0001, 0.01, 0.1, 1, 10, 100, 1000, 10000], 'penalty': ['l1', 'l2']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=True)
```

In [166]:

```
results = pd.DataFrame.from_dict(model.cv_results_)
results = results.sort_values(['param_alpha'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
param_alpha = results['param_alpha']

log_alpha = [(np.log(x)/np.log(10)) for x in param_alpha]
#log_alphas = [np.log(x) for x in alphas]

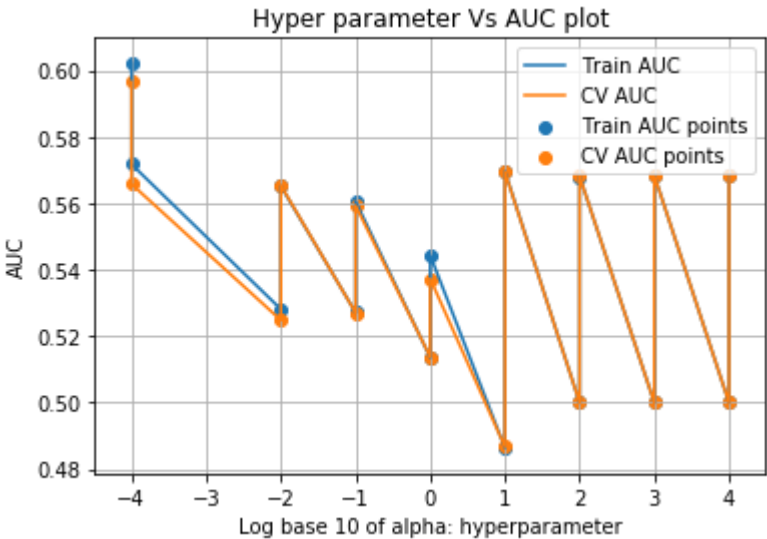
plt.plot(log_alpha, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039

plt.plot(log_alpha, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039

plt.scatter(log_alpha, train_auc, label='Train AUC points')
plt.scatter(log_alpha, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("Log base 10 of alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```



Out[166]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	param_penalty
0	33.910553	9.520917	0.081658	0.006352	0.0001	l1
1	7.279975	0.168345	0.080951	0.002128	0.0001	l1
2	14.870802	0.953758	0.081250	0.004548	0.01	l1
3	7.396511	0.139038	0.079226	0.001032	0.01	l1
4	18.313752	0.296072	0.080758	0.004264	0.1	l1

5 rows × 32 columns

In [167]:

```
model.best_estimator_
```

Out[167]:

```
SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
              n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l1',
              power_t=0.5, random_state=None, shuffle=True, tol=None,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

Observations:

1. As seen from the graph value for $\alpha = 0.0001$ is most optimal.
2. $\alpha = 0.0001$ has been suggested by the best estimator.

Training Model on the optimal value of $\alpha = 0.0001$

In [168]:

```

svm_new = SGDClassifier(class_weight='balanced',alpha=0.0001,penalty='l1')
svm_new.fit(X_tr_new,Y_train)

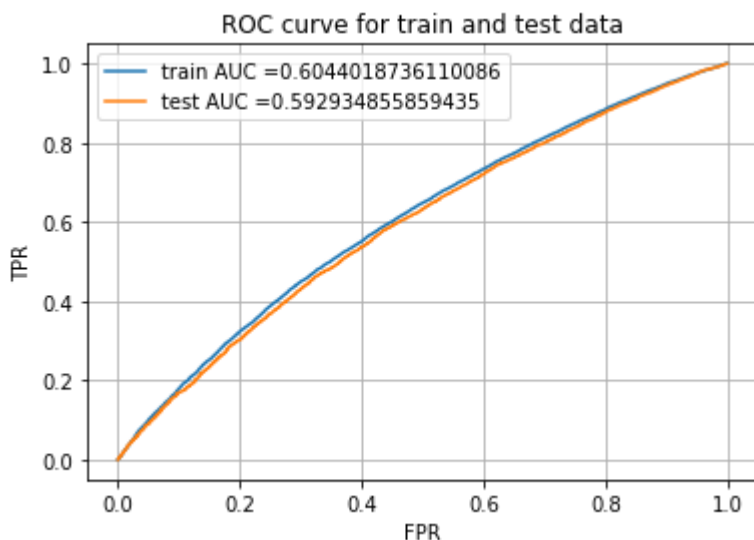
y_train_pred = svm_new.decision_function(X_tr_new)
y_test_pred = svm_new.decision_function(X_test_new)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs
train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC =" + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()

plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC curve for train and test data")
plt.grid()
plt.show()

```



In [169]:

```
te_thresholds[np.argmax(test_tpr-test_fpr)]
```

Out[169]:

```
-208.04425500023112
```

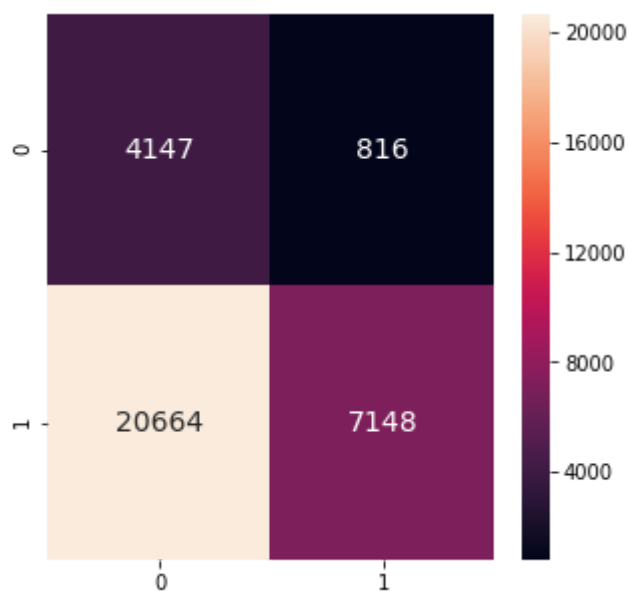
In [170]:

```
y_test_predict = svm_new.predict(X_test_new)

results = confusion_matrix(Y_test, y_test_predict)
plt.figure(figsize = (5,5))
sns.heatmap(results, annot=True,annot_kws={"size": 14}, fmt='g')
```

Out[170]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f7a8e626400>



Creating confusion matrix from custom threshold from ROC curve

In [171]:

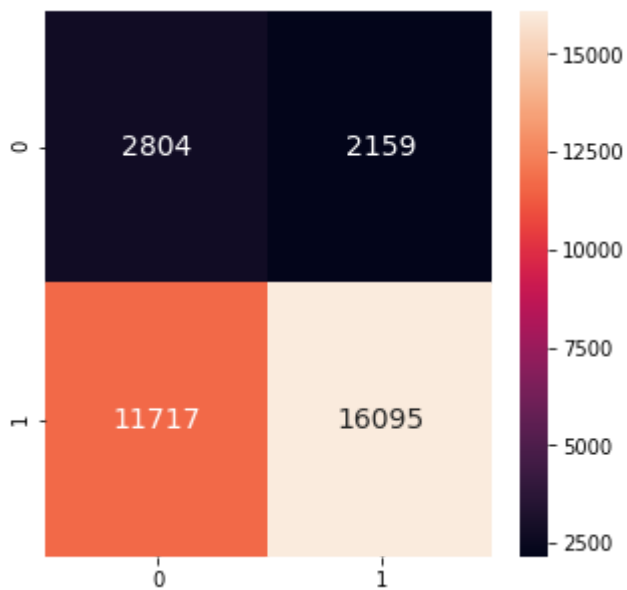
```
y_test_predict = predict_result(y_test_pred, te_thresholds, test_tpr, test_fpr)

results = confusion_matrix(Y_test, y_test_predict)
plt.figure(figsize = (5,5))
sns.heatmap(results, annot=True, annot_kws={"size": 14}, fmt='g')
```

threshold used: -208.04425500023112

Out[171]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f7a5675c748>



Summary

- The best AUC is seen for BOW vectorization of essays.
- The TFIDF-W2Vec classifies negative points(minority class) really well.
- Almost always the prediction made using the confidence level scores returned by the SVM i.e. distance of point from the hyperplane and using a custom threshold from ROC curve gives best prediction.
- The model with truncatedSVD of tf-idf essays underperforms tf-idf-essays, this is because the truncated svd preserves only 90% of the variance.
- Reducing the dimensions of essays reduces the AUC significantly, this shows that essays are crucial at performing classification task.

3. Conclusion

In [172]:

```
x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "alpha:Hyper Parameter", "Penalty", "AUC"]

x.add_row(["BOW", "SVM-SGDClassifier", 0.01, "L2", 0.67])
x.add_row(["TFIDF", "SVM-SGDClassifier", 0.0001, "L2", 0.67])
x.add_row(["W2Vec", "SVM-SGDClassifier", 0.0001, "L1", 0.66])
x.add_row(["TFIDF-W2Vec", "SVM-SGDClassifier", 0.0001, "L2", 0.63])
x.add_row(["Truncated SVD-Essays", "SVM-SGDClassifier", 0.0001, "L1", 0.59])
print(x)
```

```
+-----+-----+-----+
+-----+-----+
|      Vectorizer      |      Model      | alpha:Hyper Parameter |
Penalty | AUC |
+-----+-----+-----+
+-----+-----+
|      BOW      | SVM-SGDClassifier |      0.01      |
L2  | 0.67 |
|      TFIDF      | SVM-SGDClassifier |      0.0001     |
L2  | 0.67 |
|      W2Vec      | SVM-SGDClassifier |      0.0001     |
L1  | 0.66 |
|      TFIDF-W2Vec      | SVM-SGDClassifier |      0.0001     |
L2  | 0.63 |
| Truncated SVD-Essays | SVM-SGDClassifier |      0.0001     |
L1  | 0.59 |
+-----+-----+-----+
+-----+-----+
```