

DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

Feature	Description
<code>project_id</code>	A unique identifier for the proposed project. Example: p036502
<code>project_title</code>	Title of the project. Examples: <ul style="list-style-type: none">• Art Will Make You Happy!• First Grade Fun
<code>project_grade_category</code>	Grade level of students for which the project is targeted. One of the following enumerated values: <ul style="list-style-type: none">• Grades PreK-2• Grades 3-5• Grades 6-8• Grades 9-12
<code>project_subject_categories</code>	One or more (comma-separated) subject categories for the project from the following enumerated list of values: <ul style="list-style-type: none">• Applied Learning• Care & Hunger• Health & Sports• History & Civics• Literacy & Language• Math & Science• Music & The Arts• Special Needs• Warmth Examples: <ul style="list-style-type: none">• Music & The Arts• Literacy & Language, Math & Science
<code>school_state</code>	State where school is located (Two-letter U.S. postal code). Example: WY
<code>project_subject_subcategories</code>	One or more (comma-separated) subject subcategories for the project. Examples: <ul style="list-style-type: none">• Literacy

Feature	Description
<code>project_resource_summary</code>	An explanation of the resources needed for the project. Example: <ul style="list-style-type: none"> • My students need hands on literacy materials to manage sensory needs!
<code>project_essay_1</code>	First application essay*
<code>project_essay_2</code>	Second application essay*
<code>project_essay_3</code>	Third application essay*
<code>project_essay_4</code>	Fourth application essay*
<code>project_submitted_datetime</code>	Datetime when project application was submitted. Example: 2016-04-28 12:43:56.245
<code>teacher_id</code>	A unique identifier for the teacher of the proposed project. Example: bdf8baa8fedef6bfeec7ae4ff1c15c56
<code>teacher_prefix</code>	Teacher's title. One of the following enumerated values: <ul style="list-style-type: none"> • nan • Dr. • Mr. • Mrs. • Ms. • Teacher.
<code>teacher_number_of_previously_posted_projects</code>	Number of project applications previously submitted by the same teacher. Example: 2

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

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

Feature	Description
<code>id</code>	A <code>project_id</code> value from the <code>train.csv</code> file. Example: p036502
<code>description</code>	Description of the resource. Example: Tenor Saxophone Reeds, Box of 25
<code>quantity</code>	Quantity of the resource required. Example: 3
<code>price</code>	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
<code>project_is_approved</code>	A binary flag indicating whether DonorsChoose approved the project. A value of 0 indicates the project was not approved, and a value of 1 indicates the project was approved.

Notes on the Essay Data

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

- `__project_essay_1__` "Introduce us to your classroom"
- `__project_essay_2__` "Tell us more about your students"
- `__project_essay_3__` "Describe how your students will use the materials you're requesting"
- `__project_essay_3__` "Close by sharing why your project will make a difference"

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

- `__project_essay_1__` "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."

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
```

```
C:\Users\Shashank\Anaconda3\lib\site-packages\gensim\utils.py:1209: UserWarning: detected Windows;
aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

1.1 Reading Data

In [2]:

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

In [3]:

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

```
Number of data points in train data (109248, 17)
-----
```

```
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories']
```

```
'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
'project_essay_4' 'project_resource_summary'
'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [4]:

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

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

Out[4]:

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

1.2 preprocessing of project_subject_categories

In [5]:

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

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

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

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

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

1.3 preprocessing of project_subject_subcategories

In [6]:

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

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

```
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python

sub_cat_list = []
for i in sub_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 space "Math & Science" => "Math", "&", "Science"
            j = j.replace('The', '') # if we have the words "The" we are going to replace it with '' (i.e removing 'The')
            j = j.replace(' ', '') # we are placing all the ' ' (space) with '' (empty) ex: "Math & Science" => "Math&Science"
            temp += j.strip() + " #" + abc ".strip() will return "abc", remove the trailing spaces
            temp = temp.replace('&', '_')
        sub_cat_list.append(temp.strip())

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

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

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

1.3 Text preprocessing

In [7]:

```
# 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 [8]:

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

Out[8]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Gra
1	140945	p258326	897464ce9ddc600bcd1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Gra

In [9]:

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

In [10]:

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

My students are English learners that are working on English as their second or third languages. We are a melting pot of refugees, immigrants, and native-born Americans bringing the gift of language to our school. We have over 24 languages represented in our English Learner program with students at every level of mastery. We also have over 40 countries represented with the families within our school. Each student brings a wealth of knowledge and experiences to us that open our eyes to new cultures, beliefs, and respect. "The limits of your language are the limits of your world." -Ludwig Wittgenstein Our English learner's have a strong support system at home that begs for more resources. Many times our parents are learning to read and speak English alongside of their children. Sometimes this creates barriers for parents to be able to help their child learn phonetics, letter recognition, and other reading skills. By providing these dvd's and players, students are able to continue their mastery of the English language even if no one at home is able to assist. All families with students within the Level 1 proficiency status, will be offered to be a part of this program. These educational videos will be specially chosen by the English Learner Teacher and will be sent home regularly to watch. The videos are to help the child develop early reading skills. Parents that do not have access to a dvd player will have the opportunity to check out a dvd player to use for the year. The plan is to use these videos and educational dvd's for the years to come for other EL students.

=====

The 51 fifth grade students that will cycle through my classroom this year all love learning, at least most of the time. At our school, 97.3% of the students receive free or reduced price lunch. Of the 560 students, 97.3% are minority students. The school has a vibrant community that loves to get together and celebrate. Around Halloween there is a whole school parade to show off the beautiful costumes that students wear. On Cinco de Mayo we put on a big festival with crafts made by the students, dances, and games. At the end of the year the school hosts a carnival to celebrate the hard work put in during the school year, with a dunk tank being the most popular activity. My students will use these five brightly colored Hokki stools in place of regular, stationary, 4-legged chairs. As I will only have a total of ten in the classroom and not enough for each student to have an individual one, they will be used in a variety of ways. During independent reading time they will be used as special chairs students will each use on occasion. I will utilize them in place of chairs at my small group tables during math and reading times. The rest of the day they will be used by the students who need the highest amount of movement in their life in order to stay focused on school. Whenever asked what the classroom is missing, my students always say more Hokki Stools. They can't get their fill of the 5 stools we already have. When the students are sitting in group with me on the Hokki Stools, they are always moving, but at the same time doing their work. Anytime the students get to pick where they can sit, the Hokki Stools are the first to be taken. There are always students who head over to the kidney table to get one of the stools who are disappointed as there are not enough of them. We ask a lot of students to sit for 7 hours a day. The Hokki stools will be a compromise that allow my students to do desk work and move at the same time. These stools will help students to meet their 60 minutes a day of movement by allowing them to activate their core muscles for balance while they sit. For many of my students, these chairs will take away the barrier that exists in place of a child who can't sit still.

=====

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. My class is made up of 28 wonderfully unique boys and girls of mixed races in Arkansas. They 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. Your generous donations will help me to help make our classroom a fun, inviting, learning environment from day one. It costs a lot 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 more successful one. Thank you!

o 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\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\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\n\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 letters, words and pictures for students to learn about different letters and it is more accessible.nannan

In [11]:

```
# 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 [12]:

```
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 [13]:

```
# \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. nannan

In [14]:

```
#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 nannan

In [15]:

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


In [16]:

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

100%|

109248/109248 [02:23<00:00, 763.18it/s]

In [17]:

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

Out[17]:

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

In [18]:

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

In [19]:

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

1.4 Preprocessing of `project_title`

In [20]:

```
# similarly you can preprocess the titles also
```

In [21]:

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



In [22]:

```
project_data['project_title_list'] = project_title_list
```

In [23]:

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

1.5 Preparing data for models

In [24]:

```
project_data.columns
```

Out[24]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',  
      'project_submitted_datetime', 'project_grade_category',  
      'project_essay_1', 'project_essay_2', 'project_essay_3',  
      'project_essay_4', 'project_resource_summary',  
      'teacher_number_of_previously_posted_projects', 'project_is_approved',  
      'clean_categories', 'clean_subcategories', 'preprocessed_essays',  
      'project_title_list'],  
      dtype='object')
```

we are going to consider

- school_state : categorical data
- clean_categories : categorical data
- clean_subcategories : categorical data
- project_grade_category : categorical data
- teacher_prefix : categorical data
- project_title : text data
- text : text data
- project_resource_summary: text data (optinal)
- quantity : numerical (optinal)
- teacher_number_of_previously_posted_projects : numerical
- price : numerical

1.5.1 Vectorizing Categorical data

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

In [25]:

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

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

In [26]:

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

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

In [27]:

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

In [28]:

```
#onehotencoding for school_state
one_hot_encoding_school_state=pd.get_dummies(project_data.school_state)
print("Shape of dataframe for school_state", one_hot_encoding_school_state.shape)
```

Shape of dataframe for school_state (109248, 51)

In [29]:

```
#onehotencoding for project_grade_category
one_hot_encoding_project_grade_category=pd.get_dummies(project_data.project_grade_category)

print("Shape of dataframe for project_grade_category", one_hot_encoding_project_grade_category.shape)
```

Shape of dataframe for project_grade_category (109248, 4)

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [30]:

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

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

In [31]:

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

In [32]:

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

Shape of matrix after one hot encodig (109248, 3222)

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

1.5.2.2 TFIDF vectorizer

In [33]:

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

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

1.5.2.3 Using Pretrained Models: Avg W2V

In [34]:

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

# =====
Output:

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

# =====

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

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

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

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

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

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

'''
```

Out[34]:

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

In [35]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-sa
ve-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 [36]:

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

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

```
100%|
109248
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [37]:

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

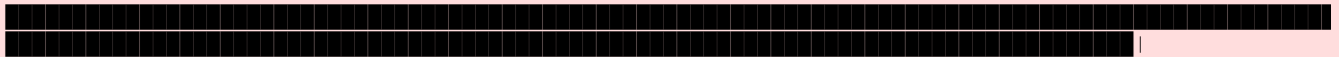
In [38]:

```
# average Word2Vec
```

```
# compute average word2vec for each review.
tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
            idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors.append(vector)

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

100%|



109248/109248 [07:29<00:00, 242.94it/s]

109248
300

In [39]:

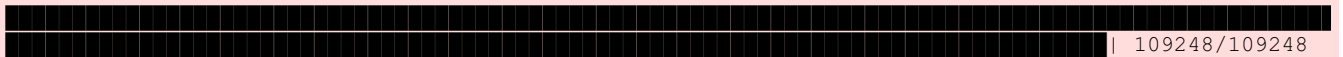
```
# Similarly you can vectorize for title also
```

In [40]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_project_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(project_title_list): # 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_project_title.append(vector)

print(len(avg_w2v_vectors_project_title))
print(len(avg_w2v_vectors_project_title[0]))
```

100%|



[00:03<00:00, 32416.99it/s]

109248
300

In [41]:

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

In [42]:

```

# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_project_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(project_title_list): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
            idf 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_project_title.append(vector)

print(len(tfidf_w2v_vectors_project_title))
print(len(tfidf_w2v_vectors_project_title[0]))

```

100%|

[00:08<00:00, 13171.25it/s] | 109248/109248

109248

300

1.5.3 Vectorizing Numerical features

In [43]:

```

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 [44]:

```

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

# price_standardized = standardScaler.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329. ... 399. 287.
73 5.5 ].
# Reshape your data either using array.reshape(-1, 1)

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

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

```

Mean : 298.1193425966608, Standard deviation : 367.49634838483496

In [45]:

```
price_standardized
```

Out[45]:

```

array([[ -0.3905327 ],
       [  0.00239637],
       [  0.59519138],
       ...,

```

```
[-0.15825829],  
[-0.61243967],  
[-0.51216657]])
```

1.5.4 Merging all the above features

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

In [46]:

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

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

In [47]:

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

Out[47]:

```
(109248, 16663)
```

In [48]:

```
# please write all the code with proper documentation, and proper titles for each subsection  
# when you plot any graph make sure you use  
# a. Title, that describes your plot, this will be very helpful to the reader  
# b. Legends if needed  
# c. X-axis label  
# d. Y-axis label
```

Computing Sentiment Scores

In [49]:

```
import nltk  
from nltk.sentiment.vader import SentimentIntensityAnalyzer  
  
# import nltk  
# nltk.download('vader_lexicon')  
  
sid = SentimentIntensityAnalyzer()  
  
for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest students w  
ith the biggest enthusiasm \  
for learning my students learn in many different ways using all of our senses and multiple intelli  
gences i use a wide range\  
of techniques to help all my students succeed students in my class come from a variety of differen  
t backgrounds which makes\  
for wonderful sharing of experiences and cultures including native americans our school is a carin  
g community of successful \  
learners which can be seen through collaborative student project based learning in and out of the  
classroom kindergarteners \  
in my class love to work with hands on materials and have many different opportunities to practice  
a skill before it is\  
mastered having the social skills to work cooperatively with friends is a crucial aspect of the ki  
ndergarten curriculum\  
montana is the perfect place to learn about agriculture and nutrition my students love to role pla  
y in our pretend kitchen\  
in the early childhood classroom i have had several kids ask me can we try cooking with real food'
```



```

in the early childhood classroom I have had several kids ask me can we try cooking with real food
i will take their idea \
and create common core cooking lessons where we learn important math and writing concepts while co
oking delicious healthy \
food for snack time my students will have a grounded appreciation for the work that went into maki
ng the food and knowledge \
of where the ingredients came from as well as how it is healthy for their bodies this project woul
d expand our learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade apple
sauce make our own bread \
and mix up healthy plants from our classroom garden in the spring we will also create our own cook
books to be printed and \
shared with families students will gain math and literature skills as well as a life long enjoymen
t for healthy cooking \
nannan'
ss = sid.polarity_scores(for_sentiment)

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

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

```

C:\Users\Shashank\Anaconda3\lib\site-packages\nltk\twitter__init__.py:20: UserWarning:

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

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

In [50]:

```
y=project_data['project_is_approved']
```

Assignment 7: SVM

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](#).

2. Support Vector Machines

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

In [51]:

```

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

```

In [52]:

```
from sklearn.model_selection import train_test_split
```

```
X1_train, X_test_bow, y1_train, y_test_bow = train_test_split(
    project_data, y, test_size=0.20, stratify=y, random_state=42)
X_cv_bow, X_train_bow, y_cv_bow, y_train_bow = train_test_split(X1_train, y1_train, test_size=0.70, stratify=y1_train, random_state=42)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [53]:

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

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

In [54]:

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

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329. ... 399. 287.
73 5.5 ].
# Reshape your data either using array.reshape(-1, 1)

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

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

Mean : 298.3190839994116, Standard deviation : 370.8546387731735

In [55]:

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

In [56]:

```
price_standardized_test_bow = price_scalar.transform(X_test_bow['price'].values.reshape(-1, 1))
```

In [57]:

```
#onehotencoding for teacher_prefix
one_hot_encoding_school_state_train_bow=pd.get_dummies(X_train_bow.school_state)

print("Shape of dataframe for school_state", one_hot_encoding_school_state_train_bow.shape)
```

Shape of dataframe for school_state (61179, 51)

In [58]:

```
#onehotencoding for school_state
one_hot_encoding_school_state_cv_bow=pd.get_dummies(X_cv_bow.school_state)
print("Shape of dataframe for school_state", one_hot_encoding_school_state_cv_bow.shape)
```

Shape of dataframe for school_state (26219, 51)

In [59]:

```
#onehotencoding for school_state
one_hot_encoding_school_state_test_bow=pd.get_dummies(X_test_bow.school_state)
print("Shape of dataframe for school_state", one_hot_encoding_school_state_test_bow.shape)
```

Shape of dataframe for school_state (21850, 51)

In [60]:

```
#onehotencoding for teacher_prefix
one_hot_encoding_teacher_prefix_train_bow=pd.get_dummies(X_train_bow.teacher_prefix)

print("Shape of dataframe for teacher_prefix", one_hot_encoding_teacher_prefix_train_bow.shape)
```

Shape of dataframe for teacher_prefix (61179, 5)

In [61]:

```
#onehotencoding for teacher_prefix
one_hot_encoding_teacher_prefix_cv_bow=pd.get_dummies(X_cv_bow.teacher_prefix)

print("Shape of dataframe for teacher_prefix", one_hot_encoding_teacher_prefix_cv_bow.shape)
```

Shape of dataframe for teacher_prefix (26219, 5)

In [62]:

```
#onehotencoding for teacher_prefix
one_hot_encoding_teacher_prefix_test_bow=pd.get_dummies(X_test_bow.teacher_prefix)

print("Shape of dataframe for teacher_prefix", one_hot_encoding_teacher_prefix_test_bow.shape)
```

Shape of dataframe for teacher_prefix (21850, 5)

In [63]:

```
#onehotencoding for project_grade_category
one_hot_encoding_project_grade_category_train_bow=pd.get_dummies(X_train_bow.project_grade_category
)

print("Shape of dataframe for project_grade_category",
one_hot_encoding_project_grade_category_train_bow.shape)
```

Shape of dataframe for project_grade_category (61179, 4)

In [64]:

```
#onehotencoding for project_grade_category
one_hot_encoding_project_grade_category_cv_bow=pd.get_dummies(X_cv_bow.project_grade_category)

print("Shape of dataframe for project_grade_category",
one_hot_encoding_project_grade_category_cv_bow.shape)
```

Shape of dataframe for project_grade_category (26219, 4)

In [65]:

```
#onehotencoding for project_grade_category
one_hot_encoding_project_grade_category_test_bow=pd.get_dummies(X_test_bow.project_grade_category)

print("Shape of dataframe for project_grade_category",
one_hot_encoding_project_grade_category_test_bow.shape)
```

Shape of dataframe for project_grade_category (21850, 4)

In [66]:

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

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding (61179, 9)
```

In [67]:

```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True)
categories_one_hot_cv_bow = vectorizer.transform(X_cv_bow['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encoding ",categories_one_hot_cv_bow.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding (26219, 9)
```

In [68]:

```
categories_one_hot_test_bow = vectorizer.transform(X_test_bow['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encoding ",categories_one_hot_test_bow.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding (21850, 9)
```

In [69]:

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

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL',
'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encoding (61179, 30)
```

In [70]:

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

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other']
```

```

SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL',
'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encoding (26219, 30)

```

In [71]:

```

# we use count vectorizer to convert the values into one

vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=True)
sub_categories_one_hot_cv_bow = vectorizer.transform(X_cv_bow['clean_subcategories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encoding ", sub_categories_one_hot_cv_bow.shape)

```

```

['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL',
'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encoding (26219, 30)

```

In [72]:

```

# we use count vectorizer to convert the values into one

sub_categories_one_hot_test_bow = vectorizer.transform(X_test_bow['clean_subcategories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encoding ", sub_categories_one_hot_test_bow.shape)

```

```

['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL',
'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encoding (21850, 30)

```

2.3 Make Data Model Ready: encoding essay, and project_title

In [73]:

```

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

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

```

In [74]:

```

# We are considering only the words which appeared in at least 10 documents (rows or projects).
vectorizer = CountVectorizer(ngram_range=(1, 2), min_df=10, max_features=5000)
text_bow_essay_train = vectorizer.fit_transform(X_train_bow['preprocessed_essays'])
print("Shape of matrix after one hot encoding ", text_bow_essay_train.shape)

```

Shape of matrix after one hot encoding (61179, 5000)

In [75]:

```

# We are considering only the words which appeared in at least 10 documents (rows or projects).

```

```
text_bow_essay_cv = vectorizer.transiorm(x_cv_bow['preprocessed_essays'])
print("Shape of matrix after one hot encodig ",text_bow_essay_cv.shape)
```

Shape of matrix after one hot encodig (26219, 5000)

In [76]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).

text_bow_essay_test = vectorizer.transform(X_test_bow['preprocessed_essays'])
print("Shape of matrix after one hot encodig ",text_bow_essay_test.shape)
```

Shape of matrix after one hot encodig (21850, 5000)

In [77]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(ngram_range=(1, 2),min_df=10,max_features=5000)
text_bow_project_title_train = vectorizer.fit_transform(X_train_bow['project_title_list'])
print("Shape of matrix after one hot encodig ",text_bow_project_title_train.shape)
```

Shape of matrix after one hot encodig (61179, 3933)

In [78]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).

text_bow_project_title_cv= vectorizer.transform(X_cv_bow['project_title_list'])
print("Shape of matrix after one hot encodig ",text_bow_project_title_cv.shape)
```

Shape of matrix after one hot encodig (26219, 3933)

In [79]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).

text_bow_project_title_test = vectorizer.transform(X_test_bow['project_title_list'])
print("Shape of matrix after one hot encodig ",text_bow_project_title_test.shape)
```

Shape of matrix after one hot encodig (21850, 3933)

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

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

For Every model that you work on make sure you do the step 2 and step 3 of instructions

In [80]:

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

2.4.1 Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (BOW with bi-grams with min_df=10 and max_features=5000)

In [81]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatenating a sparse matrix and a dense matrix :)
bow_data_matrix_train=
hstack((one_hot_encoding_school_state_train_bow,one_hot_encoding_teacher_prefix_train_bow,one_hot_encoding_project_grade_category_train_bow,categories_one_hot_train_bow,sub_categories_one_hot_train_bow,price_standardized_train_bow,text_bow_essay_train,text_bow_project_title_train))
bow_data_matrix_train.shape
```

Out[81]:

```
(61179, 9033)
```

In [82]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatenating a sparse matrix and a dense matrix :)
bow_data_matrix_cv=
hstack((one_hot_encoding_school_state_cv_bow,one_hot_encoding_teacher_prefix_cv_bow,one_hot_encoding_project_grade_category_cv_bow,categories_one_hot_cv_bow,sub_categories_one_hot_cv_bow,price_standardized_cv_bow,text_bow_essay_cv,text_bow_project_title_cv))
bow_data_matrix_cv.shape
```

Out[82]:

```
(26219, 9033)
```

In [83]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatenating a sparse matrix and a dense matrix :)
bow_data_matrix_test=
hstack((one_hot_encoding_school_state_test_bow,one_hot_encoding_teacher_prefix_test_bow,one_hot_encoding_project_grade_category_test_bow,categories_one_hot_test_bow,sub_categories_one_hot_test_bow,price_standardized_test_bow,text_bow_essay_test,text_bow_project_title_test))
bow_data_matrix_test.shape
```

Out[83]:

```
(21850, 9033)
```

In [84]:

```
y_train_bow.shape
```

Out[84]:

```
(61179,)
```

In [85]:

```
from scipy.sparse import coo_matrix
m = coo_matrix(bow_data_matrix_train)
m1 = m.tocsr()
```

In [86]:

```
new_bow_data_matrix_train=m1[:60001]
```

In [87]:

```
#Normalize Data
from sklearn import preprocessing
new_bow_data_matrix_train= preprocessing.normalize(new_bow_data_matrix_train)
```

In [88]:

```
new_y_train_bow=y_train_bow[:60001]
```

In [89]:

```
from scipy.sparse import coo_matrix
m2 = coo_matrix(bow_data_matrix_test)
m3 = m2.tocsr()
```

In [90]:

```
new_bow_data_matrix_test=m3[:20001]
```

In [91]:

```
#Normalize Data
new_bow_data_matrix_test= preprocessing.normalize(new_bow_data_matrix_test)
```

In [92]:

```
new_y_test_bow=y_test_bow[:20001]
```

In [93]:

```
from scipy.sparse import coo_matrix
m4 = coo_matrix(bow_data_matrix_cv)
m5 = m4.tocsr()
```

In [94]:

```
new_bow_data_matrix_cv=m5[:20001]
```

In [95]:

```
new_y_cv_bow=y_cv_bow[:20001]
```

In [101]:

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

    y_data_pred_bow = []
    tr_loop_bow = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate until the last 1000 multiplier
    for i in range(0, tr_loop_bow, 1000):
        y_data_pred_bow.extend(clf.predict_proba(data[i:i+1000])[:,1])
        # we will be predicting for the last data points
        y_data_pred_bow.extend(clf.predict_proba(data[tr_loop_bow:])[:,1])

    return y_data_pred_bow
```

In [97]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
from sklearn.linear_model import SGDClassifier
svm = linear_model.SGDClassifier(loss='hinge')
param_grid = {'alpha':[10000,1000,100,10,1,0.1,0.01,0.001,0.0001,0.00001,.000001],'penalty':['l1','l2']}

clf = GridSearchCV(svm, param_grid, cv=10, scoring='roc_auc')
clf.fit(new_bow_data_matrix_train,new_y_train_bow)
```



```

train_auc_bow= clf.cv_results_['mean_train_score']
train_auc_std_bow= clf.cv_results_['std_train_score']
cv_auc_bow = clf.cv_results_['mean_test_score']
cv_auc_std_bow= clf.cv_results_['std_test_score']
train_auc_bow_new=train_auc_bow[:11]
cv_auc_bow_new=cv_auc_bow[:11]
train_auc_std_bow_new=train_auc_std_bow[:11]

cv_auc_std_bow_new =cv_auc_std_bow[:11]

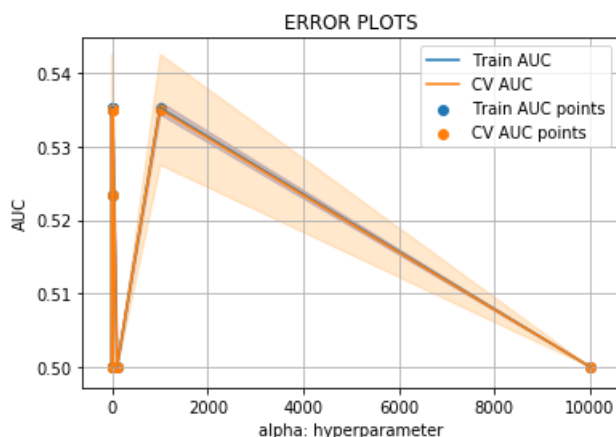
plt.plot(param_grid['alpha'], train_auc_bow_new, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(param_grid['alpha'],train_auc_bow_new -
train_auc_std_bow_new,train_auc_bow_new + train_auc_std_bow_new,alpha=0.2,color='darkblue')

plt.plot(param_grid['alpha'], cv_auc_bow_new, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(param_grid['alpha'],cv_auc_bow_new - cv_auc_std_bow_new,cv_auc_bow_new + cv_
auc_std_bow_new,alpha=0.2,color='darkorange')

plt.scatter(param_grid['alpha'], train_auc_bow_new, label='Train AUC points')
plt.scatter(param_grid['alpha'], cv_auc_bow_new, label='CV AUC points')

plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()

```



In [235]:

```

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

from sklearn.calibration import CalibratedClassifierCV
svm = linear_model.SGDClassifier(alpha=0.0001)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs
#https://www.kaggle.com/mpearmain/calibrated-sgdclassifier
model_calib = CalibratedClassifierCV(base_estimator=svm,method='isotonic')
model_calib.fit(new_bow_data_matrix_train,new_y_train_bow)

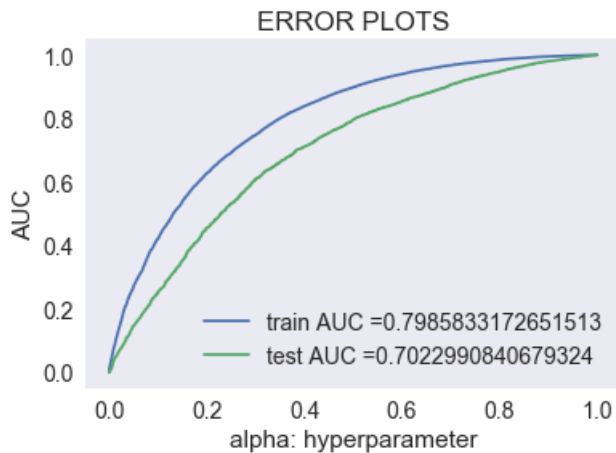
y_train_pred_bow = batch_predict(model_calib,new_bow_data_matrix_train )
y_test_pred_bow = batch_predict(model_calib, new_bow_data_matrix_test)

train_fpr_bow, train_tpr_bow, tr_thresholds_bow = roc_curve(new_y_train_bow, y_train_pred_bow)
test_fpr_bow, test_tpr_bow, te_thresholds_bow = roc_curve(new_y_test_bow, y_test_pred_bow)

plt.plot(train_fpr_bow, train_tpr_bow, label="train AUC =" +str(auc(train_fpr_bow, train_tpr_bow)))

```

```
plt.plot(test_fpr_bow, test_tpr_bow, label="test AUC =" + str(auc(test_fpr_bow, test_tpr_bow)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [104]:

```
# we are writing our own function for predict, with defined threshold
# we will pick a threshold that will give the least fpr
def predict(proba, threshold, fpr, tpr):

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

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

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [105]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
df_cm=confusion_matrix(new_y_train_bow, predict(y_train_pred_bow, tr_thresholds_bow, train_fpr_bow
, train_fpr_bow))

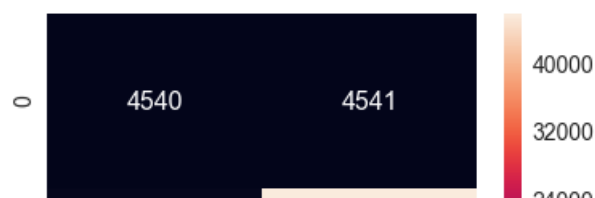
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

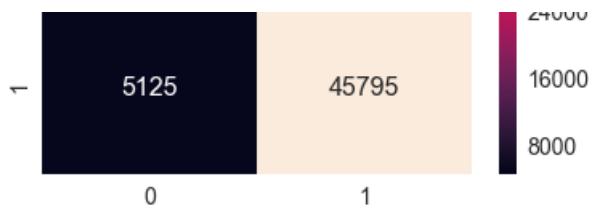
```
=====

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2499999969683947 for threshold 0.78
```

Out[105]:

<matplotlib.axes._subplots.AxesSubplot at 0x1eb26c1d278>





In [107]:

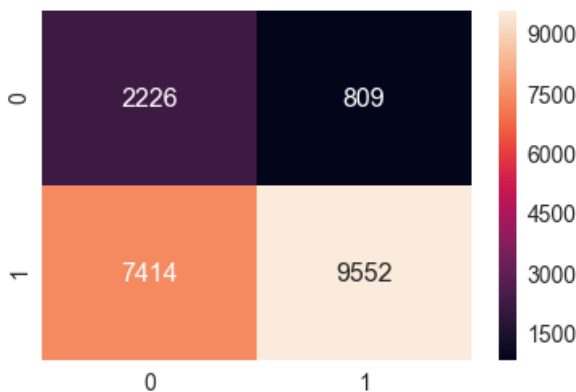
```
print("Test confusion matrix")

df_cm_test=confusion_matrix(new_y_test_bow, predict(y_test_pred_bow, tr_thresholds_bow,
test_fpr_bow, test_fpr_bow))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

Test confusion matrix
the maximum value of $tpr \cdot (1 - fpr)$ 0.24999997285920167 for threshold 0.866

Out[107]:

<matplotlib.axes._subplots.AxesSubplot at 0x1eb23d41a90>



2.4.2 Set 2: categorical, numerical features + project_title(TFIDF) + preprocessed_eassay (TFIDF with bi-grams with min_df=10 and max_features=5000)

In [51]:

```
from sklearn.model_selection import train_test_split
X1_train, X_test_tfidf, y1_train, y_test_tfidf = train_test_split(
    project_data, y, test_size=0.20, stratify=y, random_state=42)
X_cv_tfidf, X_train_tfidf, y_cv_tfidf, y_train_tfidf = train_test_split(X1_train, y1_train, test_size=0.70,
    stratify=y1_train, random_state=42)
```

In [52]:

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

# price_standardized = standardScaler.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329. ... 399. 287. 73 5.5 ].
# Reshape your data either using array.reshape(-1, 1)

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

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

Mean : 298.3190839994116, Standard deviation : 370.8546387731735

In [53]:

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

In [54]:

```
#onehotencoding for school_state
one_hot_encoding_school_state_train_tfidf=pd.get_dummies(X_train_tfidf.school_state)
print("Shape of dataframe for school_state", one_hot_encoding_school_state_train_tfidf.shape)
```

Shape of dataframe for school_state (61179, 51)

In [55]:

```
#onehotencoding for school_state
one_hot_encoding_school_state_cv_tfidf=pd.get_dummies(X_cv_tfidf.school_state)
print("Shape of dataframe for school_state", one_hot_encoding_school_state_cv_tfidf.shape)
```

Shape of dataframe for school_state (26219, 51)

In [56]:

```
#onehotencoding for school_state
one_hot_encoding_school_state_test_tfidf=pd.get_dummies(X_test_tfidf.school_state)
print("Shape of dataframe for school_state", one_hot_encoding_school_state_test_tfidf.shape)
```

Shape of dataframe for school_state (21850, 51)

In [57]:

```
#onehotencoding for teacher_prefix
one_hot_encoding_teacher_prefix_train_tfidf=pd.get_dummies(X_train_tfidf.teacher_prefix)

print("Shape of dataframe for teacher_prefix", one_hot_encoding_teacher_prefix_train_tfidf.shape)
```

Shape of dataframe for teacher_prefix (61179, 5)

In [58]:

```
#onehotencoding for teacher_prefix
one_hot_encoding_teacher_prefix_cv_tfidf=pd.get_dummies(X_cv_tfidf.teacher_prefix)

print("Shape of dataframe for teacher_prefix", one_hot_encoding_teacher_prefix_cv_tfidf.shape)
```

Shape of dataframe for teacher_prefix (26219, 5)

In [59]:

```
#onehotencoding for project_grade_category
one_hot_encoding_project_grade_category_train_tfidf=pd.get_dummies(X_train_tfidf.project_grade_category)

print("Shape of dataframe for project_grade_category",
one_hot_encoding_project_grade_category_train_tfidf.shape)
```

Shape of dataframe for project_grade_category (61179, 4)

In [60]:

In [60]:

```
#onehotencoding for project_grade_category
one_hot_encoding_project_grade_category_cv_tfidf=pd.get_dummies(X_cv_tfidf.project_grade_category)

print("Shape of dataframe for project_grade_category",
one_hot_encoding_project_grade_category_cv_tfidf.shape)
```

Shape of dataframe for project_grade_category (26219, 4)

In [61]:

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

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

Shape of matrix after one hot encodig (61179, 9)

In [62]:

```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True
)
categories_one_hot_cv_tfidf = vectorizer.transform(X_cv_tfidf['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",categories_one_hot_cv_tfidf.shape)
```

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

Shape of matrix after one hot encodig (26219, 9)

In [63]:

```
categories_one_hot_test_tfidf = vectorizer.transform(X_test_tfidf['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",categories_one_hot_test_tfidf.shape)
```

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

Shape of matrix after one hot encodig (21850, 9)

In [64]:

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

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
```

Shape of matrix after one hot encodig (61179, 30)

In [65]:

```
# we use count vectorizer to convert the values into one
```

```
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=True)
sub_categories_one_hot_cv_tfidf = vectorizer.transform(X_cv_tfidf['clean_subcategories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encoding ", sub_categories_one_hot_cv_tfidf.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL',
'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encoding (26219, 30)
```

In [66]:

```
# we use count vectorizer to convert the values into one

sub_categories_one_hot_test_tfidf =
vectorizer.transform(X_test_tfidf['clean_subcategories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encoding ", sub_categories_one_hot_test_tfidf.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL',
'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encoding (21850, 30)
```

In [67]:

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

# price_standardized = standardScaler.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329. ... 399. 287.
73 5.5 ].
# Reshape your data either using array.reshape(-1, 1)

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

In [68]:

```
#onehotencoding for school_state
one_hot_encoding_school_state_test_tfidf=pd.get_dummies(X_test_tfidf.school_state)
print("Shape of dataframe for school_state", one_hot_encoding_school_state_test_tfidf.shape)
```

Shape of dataframe for school_state (21850, 51)

In [69]:

```
#onehotencoding for teacher_prefix
one_hot_encoding_teacher_prefix_test_tfidf=pd.get_dummies(X_test_tfidf.teacher_prefix)

print("Shape of dataframe for teacher_prefix", one_hot_encoding_teacher_prefix_test_tfidf.shape)
```

Shape of dataframe for teacher_prefix (21850, 5)

In [70]:

```
#onehotencoding for teacher_prefix
one_hot_encoding_teacher_prefix_train_tfidf=pd.get_dummies(X_train_tfidf.teacher_prefix)
```

```
print("Shape of dataframe for teacher_prefix", one_hot_encoding_teacher_prefix_train_tfidf.shape)
```

Shape of dataframe for teacher_prefix (61179, 5)

In [71]:

```
#onehotencoding for teacher_prefix
one_hot_encoding_teacher_prefix_cv_tfidf=pd.get_dummies(X_cv_tfidf.teacher_prefix)

print("Shape of dataframe for teacher_prefix", one_hot_encoding_teacher_prefix_cv_tfidf.shape)
```

Shape of dataframe for teacher_prefix (26219, 5)

In [72]:

```
#onehotencoding for project_grade_category
one_hot_encoding_project_grade_category_test_tfidf=pd.get_dummies(X_test_tfidf.project_grade_category)

print("Shape of dataframe for project_grade_category",
one_hot_encoding_project_grade_category_test_tfidf.shape)
```

Shape of dataframe for project_grade_category (21850, 4)

In [73]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = TfidfVectorizer(ngram_range=(1, 2),min_df=10,max_features=5000)
text_tfidf_essay_train = vectorizer.fit_transform(X_train_tfidf['preprocessed_essays'])
print("Shape of matrix after one hot encodig ",text_tfidf_essay_train.shape)
```

Shape of matrix after one hot encodig (61179, 5000)

In [74]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
text_tfidf_essay_cv = vectorizer.transform(X_cv_tfidf['preprocessed_essays'])
print("Shape of matrix after one hot encodig ",text_tfidf_essay_cv.shape)
```

Shape of matrix after one hot encodig (26219, 5000)

In [75]:

```
text_tfidf_essay_test = vectorizer.transform(X_test_tfidf['preprocessed_essays'])
print("Shape of matrix after one hot encodig ",text_tfidf_essay_test.shape)
```

Shape of matrix after one hot encodig (21850, 5000)

In [76]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = TfidfVectorizer(ngram_range=(1, 2),min_df=10,max_features=5000)
tfidf_project_title_train = vectorizer.fit_transform(X_train_tfidf['project_title_list'])
print("Shape of matrix after one hot encodig ",tfidf_project_title_train.shape)
```

Shape of matrix after one hot encodig (61179, 3933)

In [77]:

```
tfidf_project_title_cv = vectorizer.transform(X_cv_tfidf['project_title_list'])
print("Shape of matrix after one hot encodig ",tfidf_project_title_cv.shape)
```

Shape of matrix after one hot encodig (26219, 3933)

In [78]:

```
tfidf_project_title_test = vectorizer.transform(X_test_tfidf['project_title_list'])  
print("Shape of matrix after one hot encodig ",tfidf_project_title_test.shape)
```

Shape of matrix after one hot encodig (21850, 3933)

In [79]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039  
from scipy.sparse import hstack  
# with the same hstack function we are concatenating a sparse matrix and a dense matirx :)  
tfidf_data_matrix_train=  
hstack((tfidf_project_title_train,text_tfidf_essay_train,one_hot_encoding_teacher_prefix_train_tfidf,  
,sub_categories_one_hot_train_tfidf,categories_one_hot_train_tfidf,one_hot_encoding_project_grade_category_train_tfidf,one_hot_encoding_school_state_train_tfidf,price_standardized_train_tfidf))  
tfidf_data_matrix_train.shape
```

Out[79]:

(61179, 9033)

In [81]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039  
from scipy.sparse import hstack  
# with the same hstack function we are concatenating a sparse matrix and a dense matirx :)  
tfidf_data_matrix_test=  
hstack((one_hot_encoding_school_state_test_tfidf,one_hot_encoding_teacher_prefix_test_tfidf,one_hot_encoding_project_grade_category_test_tfidf,categories_one_hot_test_tfidf,sub_categories_one_hot_test_tfidf,price_standardized_test_tfidf,text_tfidf_essay_test,tfidf_project_title_test))  
tfidf_data_matrix_test.shape
```

Out[81]:

(21850, 9033)

In [83]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039  
from scipy.sparse import hstack  
# with the same hstack function we are concatenating a sparse matrix and a dense matirx :)  
tfidf_data_matrix_cv=  
hstack((one_hot_encoding_school_state_cv_tfidf,one_hot_encoding_teacher_prefix_cv_tfidf,one_hot_encoding_project_grade_category_cv_tfidf,categories_one_hot_cv_tfidf,sub_categories_one_hot_cv_tfidf,price_standardized_cv_tfidf,text_tfidf_essay_cv,tfidf_project_title_cv))  
tfidf_data_matrix_cv.shape
```

Out[83]:

(26219, 9033)

In [84]:

```
from scipy.sparse import coo_matrix  
m = coo_matrix(tfidf_data_matrix_train)  
m1 = m.tocsr()
```

In [85]:

```
new_tfidf_data_matrix_train=m1[:60001]
```

In [86]:

```
new_y_train_tfidf=y_train_tfidf[:60001]
```

In [87]:

```
from scipy.sparse import coo_matrix
```



```
from scipy.sparse import coo_matrix
m2 = coo_matrix(tfidf_data_matrix_test)
m3 = m2.tocsr()
```

In [88]:

```
new_tfidf_data_matrix_test=m3[:20001]
```

In [89]:

```
new_y_test_tfidf=y_test_tfidf[:20001]
```

In [90]:

```
from scipy.sparse import coo_matrix
m4 = coo_matrix(tfidf_data_matrix_cv)
m5 = m4.tocsr()
```

In [91]:

```
new_tfidf_data_matrix_cv=m5[:20001]
```

In [92]:

```
new_y_cv_tfidf=y_cv_tfidf[:20001]
```

In [94]:

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

    y_data_pred_tfidf = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate until the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred_tfidf.extend(clf.predict_proba(data[i:i+1000])[:,1])
        # we will be predicting for the last data points
    y_data_pred_tfidf.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred_tfidf
```

In [116]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
from sklearn.linear_model import SGDClassifier
svm = linear_model.SGDClassifier(loss='hinge')
param_grid = {'alpha':[10000,1000,100,10,1,0.1,0.01,0.001,0.0001,0.00001,.000001], 'penalty':['l1','l2']}

clf = GridSearchCV(svm, param_grid, cv=10, scoring='roc_auc')
clf.fit(new_tfidf_data_matrix_train,new_y_train_tfidf)

train_auc_tfidf= clf.cv_results_['mean_train_score']
train_auc_std_tfidf= clf.cv_results_['std_train_score']
cv_auc_tfidf = clf.cv_results_['mean_test_score']
cv_auc_std_tfidf_new= clf.cv_results_['std_test_score']
train_auc_tfidf_new=train_auc_tfidf[:11]
cv_auc_tfidf_new=cv_auc_tfidf[:11]
train_auc_std_tfidf_new=train_auc_std_tfidf[:11]

cv_auc_std_tfidf_new =cv_auc_std_tfidf[:11]

plt.plot(param_grid['alpha'], train_auc_tfidf_new, label='Train AUC')
```

```

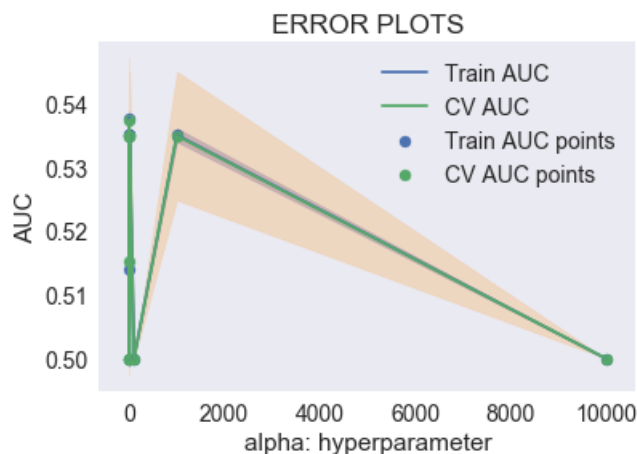
plt.gca().fill_between(param_grid['alpha'], train_auc_tfidf_new -
train_auc_std_tfidf_new, train_auc_tfidf_new + train_auc_std_tfidf_new, alpha=0.2, color='darkblue')

plt.plot(param_grid['alpha'], cv_auc_tfidf_new, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(param_grid['alpha'], cv_auc_tfidf_new -
cv_auc_std_tfidf_new, cv_auc_tfidf_new + cv_auc_std_tfidf_new, alpha=0.2, color='darkorange')

plt.scatter(param_grid['alpha'], train_auc_tfidf_new, label='Train AUC points')
plt.scatter(param_grid['alpha'], cv_auc_tfidf_new, label='CV AUC points')

plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()

```



In [130]:

```

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

from sklearn.calibration import CalibratedClassifierCV
svm = linear_model.SGDClassifier(alpha=0.000001, penalty='l1')

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs
# https://www.kaggle.com/mpearmain/calibrated-sgdclassifier
model_calib_tfidf = CalibratedClassifierCV(base_estimator=svm, method='isotonic')
model_calib_tfidf.fit(new_tfidf_data_matrix_train, new_y_train_tfidf)

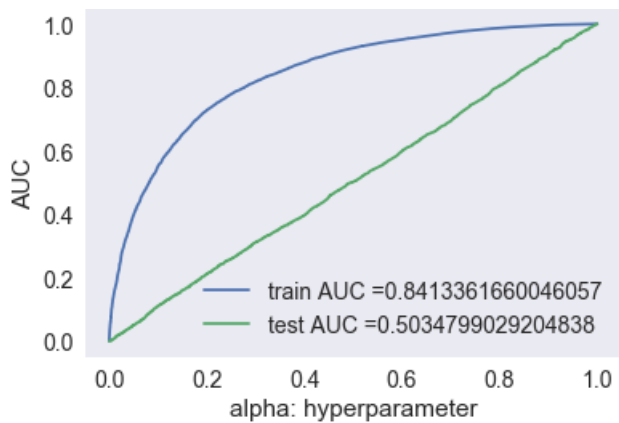
y_train_pred_tfidf = batch_predict(model_calib_tfidf, new_tfidf_data_matrix_train)
y_test_pred_tfidf = batch_predict(model_calib_tfidf, new_tfidf_data_matrix_test)

train_fpr_tfidf, train_tpr_tfidf, tr_thresholds_tfidf = roc_curve(new_y_train_tfidf,
y_train_pred_tfidf)
test_fpr_tfidf, test_tpr_tfidf, te_thresholds_tfidf = roc_curve(new_y_test_tfidf, y_test_pred_tfidf)

plt.plot(train_fpr_tfidf, train_tpr_tfidf, label="train AUC =" + str(auc(train_fpr_tfidf, train_tpr_tfidf)))
plt.plot(test_fpr_tfidf, test_tpr_tfidf, label="test AUC =" + str(auc(test_fpr_tfidf, test_tpr_tfidf)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()

```

ERROR PLOTS



In [131]:

```
# we are writing our own function for predict, with defined threshold
# we will pick a threshold that will give the least fpr
def predict(proba, threshold, fpr, tpr):

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

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

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [132]:

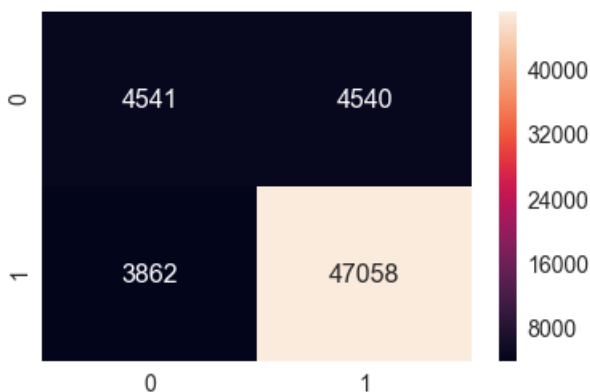
```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
df_cm=confusion_matrix(new_y_train_tfidf, predict(y_train_pred_tfidf, tr_thresholds_tfidf,
train_fpr_tfidf, train_fpr_tfidf))

sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999696839467 for threshold 0.782

Out[132]:

<matplotlib.axes._subplots.AxesSubplot at 0x17310062dd8>



In [133]:

```
print("Test confusion matrix")

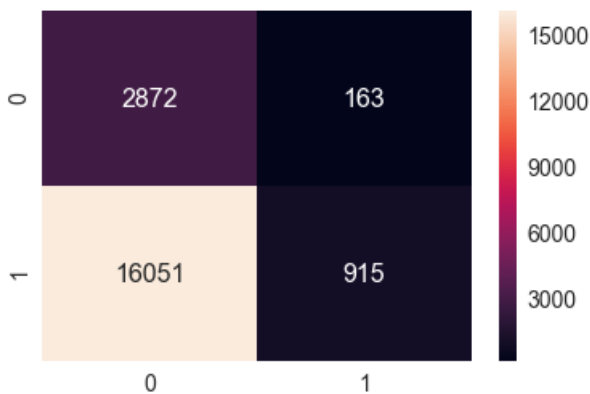
df_cm_test=confusion_matrix(new_y_test_tfidf, predict(y_test_pred_tfidf, tr_thresholds_tfidf,
test_fpr_tfidf, test_fpr_tfidf))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

Test confusion matrix

the maximum value of $tpr \cdot (1 - fpr)$ 0.249999321480042 for threshold 0.873

Out[133]:

<matplotlib.axes._subplots.AxesSubplot at 0x17310069b00>



2.4.3 Set 3: categorical, numerical features + project_title(Avg W2V) + preprocessed_eassay (AVG W2V)

In [134]:

```
from sklearn.model_selection import train_test_split
X1_train, X_test_avg, y1_train, y_test_avg = train_test_split(
    project_data, y, test_size=0.20, stratify=y, random_state=42)

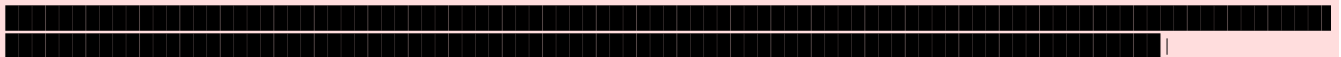
X_train_avg, X_cv_avg, y_train_avg, y_cv_avg = train_test_split(
    X1_train, y1_train, test_size=0.70, stratify=y1_train, random_state=42)
```

In [135]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_essay_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train_avg['preprocessed_essays'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_essay_train.append(vector)

print(len(avg_w2v_vectors_essay_train))
print(len(avg_w2v_vectors_essay_train[0]))
```

100%|



26219/26219 [00:30<00:00, 862.43it/s]

26219
300

In [136]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_essay_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv_avg['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_essay_cv.append(vector)

print(len(avg_w2v_vectors_essay_cv))
print(len(avg_w2v_vectors_essay_cv[0]))
```

100%|

[00:40<00:00, 1496.66it/s] | 61179/61179

61179
300

In [137]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_essay_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test_avg['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_essay_test.append(vector)

print(len(avg_w2v_vectors_essay_test))
print(len(avg_w2v_vectors_essay_test[0]))
```

100%|

[00:14<00:00, 1560.65it/s] | 21850/21850

21850
300

In [138]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_project_title_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train_avg['project_title_list']): # 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_project_title_train.append(vector)

print(len(avg_w2v_vectors_project_title_train))
```



```

from sklearn.preprocessing import StandardScaler

# price_standardized = standardScaler.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.    ... 399.    287.
73    5.5 ].
# Reshape your data either using array.reshape(-1, 1)

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

# Now standardize the data with above maen and variance.
price_standardized_train_avg = price_scaler.transform(X_train_avg['price'].values.reshape(-1, 1))

```

Mean : 294.989221938289, Standard deviation : 344.44986393419094

In [142]:

```
price_standardized_cv_avg = price_scaler.transform(X_cv_avg['price'].values.reshape(-1, 1))
```

In [143]:

```
price_standardized_test_avg = price_scaler.transform(X_test_avg['price'].values.reshape(-1, 1))
```

In [144]:

```

#onehotencoding for school_state
one_hot_encoding_school_state_train_avg=pd.get_dummies(X_train_avg.school_state)
print("Shape of dataframe for school_state", one_hot_encoding_school_state_train_avg.shape)

```

Shape of dataframe for school_state (26219, 51)

In [145]:

```

#onehotencoding for school_state
one_hot_encoding_school_state_cv_avg=pd.get_dummies(X_cv_avg.school_state)
print("Shape of dataframe for school_state", one_hot_encoding_school_state_cv_avg.shape)

```

Shape of dataframe for school_state (61179, 51)

In [146]:

```

#onehotencoding for school_state
one_hot_encoding_school_state_test_avg=pd.get_dummies(X_test_avg.school_state)
print("Shape of dataframe for school_state", one_hot_encoding_school_state_test_avg.shape)

```

Shape of dataframe for school_state (21850, 51)

In [147]:

```

#onehotencoding for teacher_prefix
one_hot_encoding_teacher_prefix_train_avg=pd.get_dummies(X_train_avg.teacher_prefix)

print("Shape of dataframe for teacher_prefix", one_hot_encoding_teacher_prefix_train_avg.shape)

```

Shape of dataframe for teacher_prefix (26219, 5)

In [148]:

```

#onehotencoding for teacher_prefix
one_hot_encoding_teacher_prefix_cv_avg=pd.get_dummies(X_cv_avg.teacher_prefix)

print("Shape of dataframe for teacher_prefix", one_hot_encoding_teacher_prefix_cv_avg.shape)

```

Shape of dataframe for teacher_prefix (61179, 5)

Shape of dataframe for teacher_prefix (61179, 5)

In [149]:

```
#onehotencoding for teacher_prefix
one_hot_encoding_teacher_prefix_test_avg=pd.get_dummies(X_test_avg.teacher_prefix)

print("Shape of dataframe for teacher_prefix", one_hot_encoding_teacher_prefix_test_avg.shape)
```

Shape of dataframe for teacher_prefix (21850, 5)

In [150]:

```
#onehotencoding for project_grade_category
one_hot_encoding_project_grade_category_train_avg=pd.get_dummies(X_train_avg.project_grade_category)

print("Shape of dataframe for project_grade_category",
one_hot_encoding_project_grade_category_train_avg.shape)
```

Shape of dataframe for project_grade_category (26219, 4)

In [151]:

```
#onehotencoding for project_grade_category
one_hot_encoding_project_grade_category_cv_avg=pd.get_dummies(X_cv_avg.project_grade_category)

print("Shape of dataframe for project_grade_category",
one_hot_encoding_project_grade_category_cv_avg.shape)
```

Shape of dataframe for project_grade_category (61179, 4)

In [152]:

```
#onehotencoding for project_grade_category
one_hot_encoding_project_grade_category_test_avg=pd.get_dummies(X_test_avg.project_grade_category)

print("Shape of dataframe for project_grade_category",
one_hot_encoding_project_grade_category_test_avg.shape)
```

Shape of dataframe for project_grade_category (21850, 4)

In [153]:

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

['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig (26219, 9)

In [154]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True)
categories_one_hot_cv_avg = vectorizer.transform(X_cv_avg['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",categories_one_hot_cv_avg.shape)
```

['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig (61179, 9)

In [155]:

```
categories_one_hot_test_avg = vectorizer.transform(X_test_avg['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encoding ",categories_one_hot_test_avg.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding (21850, 9)
```

In [156]:

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

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL',
'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encoding (26219, 30)
```

In [157]:

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

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL',
'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encoding (61179, 30)
```

In [158]:

```
# we use count vectorizer to convert the values into one

sub_categories_one_hot_test_avg =
vectorizer.fit_transform(X_test_avg['clean_subcategories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encoding ",sub_categories_one_hot_test_avg.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL',
'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encoding (21850, 30)
```

In [159]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatenating a sparse matrix and a dense matrix :)
avgw2v_data_matrix_train=
hstack((one_hot_encoding_school_state_train_avg,one_hot_encoding_teacher_prefix_train_avg,one_hot_e
ncoding_project_grade_category_train_avg,categories_one_hot_train_avg,sub_categories_one_hot_train_
```

```
avg,price_standardized_train_avg,avg_w2v_vectors_essay_train,avg_w2v_vectors_project_title_train))
avgw2v_data_matrix_train.shape
```

Out[159]:

(26219, 700)

In [160]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatenating a sparse matrix and a dense matrix :)
avgw2v_data_matrix_cv=
hstack((one_hot_encoding_school_state_cv_avg,one_hot_encoding_teacher_prefix_cv_avg,one_hot_encoding_project_grade_category_cv_avg,
categories_one_hot_cv_avg,sub_categories_one_hot_cv_avg,price_standardized_cv_avg,avg_w2v_vectors_essay_cv,avg_w2v_vectors_project_title_cv))
avgw2v_data_matrix_cv.shape
```

Out[160]:

(61179, 700)

In [161]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatenating a sparse matrix and a dense matrix :)
avgw2v_data_matrix_test=
hstack((one_hot_encoding_school_state_test_avg,one_hot_encoding_teacher_prefix_test_avg,one_hot_encoding_project_grade_category_test_avg,
categories_one_hot_test_avg,sub_categories_one_hot_test_avg,price_standardized_test_avg,avg_w2v_vectors_essay_test,avg_w2v_vectors_project_title_test))
avgw2v_data_matrix_test.shape
```

Out[161]:

(21850, 700)

In [162]:

```
from scipy.sparse import coo_matrix
n = coo_matrix(avgw2v_data_matrix_train)
n1 = n.tocsr()
```

In [163]:

```
new_avgw2v_data_matrix_train=n1[:60001]
```

In [164]:

```
new_y_train_avgw2v=y_train_avg[:60001]
```

In [165]:

```
from scipy.sparse import coo_matrix
n4 = coo_matrix(avgw2v_data_matrix_cv)
n5 = n4.tocsr()
```

In [166]:

```
new_avgw2v_data_matrix_cv=n5[:20001]
```

In [167]:

```
new_y_cv_avgw2v=y_cv_avg[:20001]
```

In [168]:

```
from scipy.sparse import coo_matrix
```

```

from scipy.sparse import coo_matrix
n2 = coo_matrix(avgw2v_data_matrix_test)
n3 = n2.tocsr()

```

In [169]:

```
new_avgw2v_data_matrix_test=n3[:20001]
```

In [170]:

```
new_y_test_avgw2v=y_test_avg[:20001]
```

In [171]:

```

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

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate until the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
        # we will be predicting for the last data points
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred

```

In [172]:

```

# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
from sklearn.linear_model import SGDClassifier
svm = linear_model.SGDClassifier(loss='hinge')
param_grid = {'alpha': [10000, 1000, 100, 10, 1, 0.1, 0.01, 0.001, 0.0001, 0.00001, .000001], 'penalty': ['l1', 'l2']}

clf = GridSearchCV(svm, param_grid, cv=10, scoring='roc_auc')
clf.fit(new_avgw2v_data_matrix_train, new_y_train_avgw2v)

train_auc_avgw2v= clf.cv_results_['mean_train_score']
train_auc_std_avgw2v= clf.cv_results_['std_train_score']
cv_auc_avgw2v = clf.cv_results_['mean_test_score']
cv_auc_std_avgw2v= clf.cv_results_['std_test_score']
train_auc_avgw2v_new=train_auc_avgw2v[:11]
cv_auc_avgw2v_new=cv_auc_avgw2v[:11]
train_auc_std_avgw2v_new=train_auc_std_avgw2v[:11]

cv_auc_std_avgw2v_new =cv_auc_std_avgw2v[:11]

plt.gca().fill_between(param_grid['alpha'], train_auc_avgw2v_new -
train_auc_std_avgw2v_new, train_auc_avgw2v_new +
train_auc_std_avgw2v_new, alpha=0.2, color='darkblue')

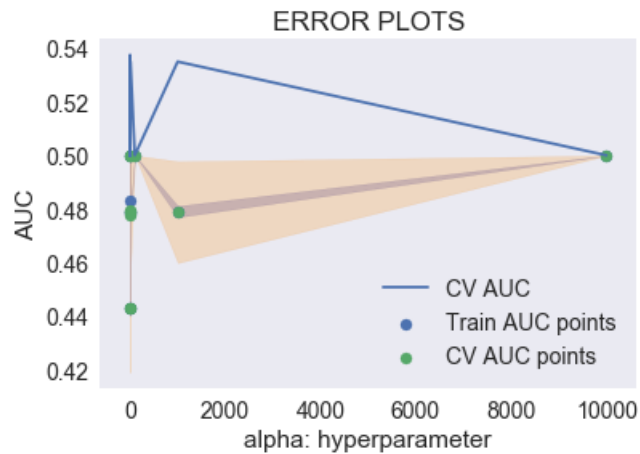
plt.plot(param_grid['alpha'], cv_auc_avgw2v_new, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(param_grid['alpha'], cv_auc_avgw2v_new -
cv_auc_std_avgw2v_new, cv_auc_avgw2v_new + cv_auc_std_avgw2v_new, alpha=0.2, color='darkorange')

plt.scatter(param_grid['alpha'], train_auc_avgw2v_new, label='Train AUC points')
plt.scatter(param_grid['alpha'], cv_auc_avgw2v_new, label='CV AUC points')

plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()

```

plt.grid()



In [180]:

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

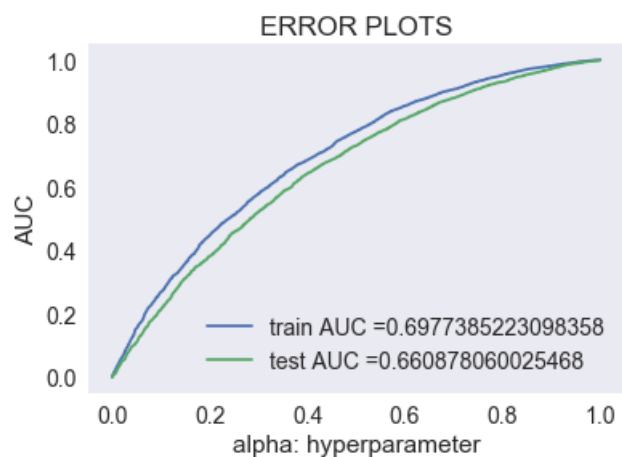
from sklearn.calibration import CalibratedClassifierCV
svm = linear_model.SGDClassifier(alpha=0.0001)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted output
# https://www.kaggle.com/mpearmain/calibrated-sgdclassifier
model_calib_avg2v = CalibratedClassifierCV(base_estimator=svm, method='isotonic')
model_calib_avg2v.fit(new_avg2v_data_matrix_train, new_y_train_avg2v)

y_train_pred_avg2v = batch_predict(model_calib_avg2v, new_avg2v_data_matrix_train)
y_test_pred_avg2v = batch_predict(model_calib_avg2v, new_avg2v_data_matrix_test)

train_fpr_avg2v, train_tpr_avg2v, tr_thresholds_avg2v = roc_curve(new_y_train_avg2v,
y_train_pred_avg2v)
test_fpr_avg2v, test_tpr_avg2v, te_thresholds_avg2v = roc_curve(new_y_test_avg2v,
y_test_pred_avg2v)

plt.plot(train_fpr_avg2v, train_tpr_avg2v, label="train AUC =" + str(auc(train_fpr_avg2v, train_tpr_avg2v)))
plt.plot(test_fpr_avg2v, test_tpr_avg2v, label="test AUC =" + str(auc(test_fpr_avg2v, test_tpr_avg2v)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [181]:

```
# we are writing our own function for predict, with defined threshold
# we will pick a threshold that will give the least fpr
def predict(proba, threshold, fpr, tpr):

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

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

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [182]:

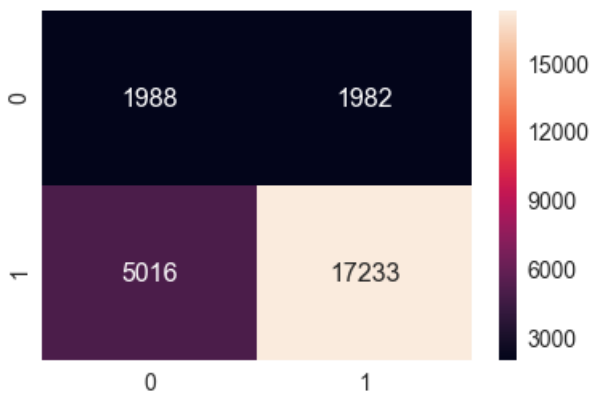
```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
df_cm_train=confusion_matrix(new_y_train_avgw2v, predict(y_train_pred_avgw2v, tr_thresholds_avgw2v,
, train_fpr_avgw2v, train_fpr_avgw2v))

sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2499994289666199 for threshold 0.827

Out[182]:

<matplotlib.axes._subplots.AxesSubplot at 0x173056ec8d0>



In [183]:

```
print("Test confusion matrix")

df_cm_test=confusion_matrix(new_y_test_avgw2v, predict(y_test_pred_avgw2v, tr_thresholds_avgw2v, t
est_fpr_avgw2v, test_fpr_avgw2v))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999975573281513 for threshold 0.844

Out[183]:

<matplotlib.axes._subplots.AxesSubplot at 0x17310024d30>

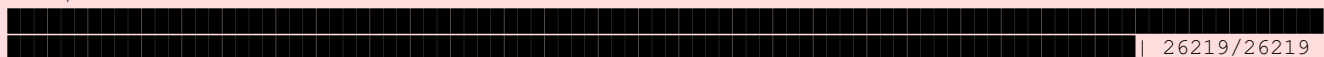

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_cv_tfidf_w2v['project_title_list'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [261]:

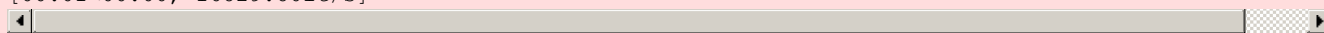
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_project_title_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv_tfidf_w2v['project_title_list'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
            idf 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_project_title_cv.append(vector)

print(len(tfidf_w2v_vectors_project_title_cv))
print(len(tfidf_w2v_vectors_project_title_cv[0]))
```

100% |



```
[00:01<00:00, 16829.80it/s]
```



26219

300

In [262]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_test_tfidf_w2v['project_title_list'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [263]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_project_title_test = []; # the avg-w2v for each sentence/review is stored in this list

for sentence in tqdm(X_test_tfidf_w2v['project_title_list'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
            idf 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_project_title_test.append(vector)

print(len(tfidf_w2v_vectors_project_title_test))
print(len(tfidf_w2v_vectors_project_title_test[0]))
```

100%|

21850/21850

[00:01<00:00, 16265.61it/s]

21850

300

In [264]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train_tfidf_w2v['preprocessed_essays'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [265]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_preprocessed_essays_train = []; # the avg-w2v for each sentence/review is stored
in this list
for sentence in tqdm(X_train_tfidf_w2v['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
            idf 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_preprocessed_essays_train.append(vector)
print(len(tfidf_w2v_vectors_preprocessed_essays_train))
print(len(tfidf_w2v_vectors_project_title_train[0]))
```

100%|

61179/61179 [03:32<00:00, 287.37it/s]

61179

300

In [266]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_cv_tfidf_w2v['preprocessed_essays'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [267]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_preprocessed_essays_cv = []; # the avg-w2v for each sentence/review is stored in
this list
for sentence in tqdm(X_cv_tfidf_w2v['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
```



```

        # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split()))))
        tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf 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_preprocessed_essays_cv.append(vector)
print(len(tfidf_w2v_vectors_preprocessed_essays_cv))
print(len(tfidf_w2v_vectors_preprocessed_essays_cv[0]))

```

100%|

26219/26219 [01:26<00:00, 304.55it/s]

26219

300

In [268]:

```

# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_test_tfidf_w2v['preprocessed_essays'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())

```

In [269]:

```

# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_test_tfidf_w2v['preprocessed_essays'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())

```

In [270]:

```

# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_preprocessed_essays_test = []; # the avg-w2v for each sentence/review is stored
in this list
for sentence in tqdm(X_test_tfidf_w2v['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split()))))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf 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_preprocessed_essays_test.append(vector)
print(len(tfidf_w2v_vectors_preprocessed_essays_test))
print(len(tfidf_w2v_vectors_preprocessed_essays_test[0]))

```

100%|

21850/21850 [01:07<00:00, 324.10it/s]

21850

300

In [271]:

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

# price_standardized = standardScaler.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329. ... 399. 287.
73 5.5 ].
# Reshape your data either using array.reshape(-1, 1)

price_scaler = StandardScaler()
price_scaler.fit(X_train_tfidf_w2v['price'].values.reshape(-1,1)) # finding the mean and standard
deviation of this data
print(f"Mean : {price_scaler.mean_[0]}, Standard deviation : {np.sqrt(price_scaler.var_[0])}")

# Now standardize the data with above mean and variance.
price_standardized_train_tfidf_w2v =
price_scaler.transform(X_train_tfidf_w2v['price'].values.reshape(-1, 1))
```

Mean : 298.3190839994116, Standard deviation : 370.8546387731735

In [272]:

```
price_standardized_cv_tfidf_w2v = price_scaler.transform(X_cv_tfidf_w2v['price'].values.reshape(-1,
1))
```

In [273]:

```
price_standardized_test_tfidf_w2v =
price_scaler.transform(X_test_tfidf_w2v['price'].values.reshape(-1, 1))
```

In [274]:

```
#onehotencoding for school_state
one_hot_encoding_school_state_train_tfidf_w2v=pd.get_dummies(X_train_tfidf_w2v.school_state)
print("Shape of dataframe for school_state", one_hot_encoding_school_state_train_tfidf_w2v.shape)
```

Shape of dataframe for school_state (61179, 51)

In [275]:

```
#onehotencoding for school_state
one_hot_encoding_school_state_cv_tfidf_w2v=pd.get_dummies(X_cv_tfidf_w2v.school_state)
print("Shape of dataframe for school_state", one_hot_encoding_school_state_cv_tfidf_w2v.shape)
```

Shape of dataframe for school_state (26219, 51)

In [276]:

```
#onehotencoding for school_state
one_hot_encoding_school_state_test_tfidf_w2v=pd.get_dummies(X_test_tfidf_w2v.school_state)
print("Shape of dataframe for school_state", one_hot_encoding_school_state_test_tfidf_w2v.shape)
```

Shape of dataframe for school_state (21850, 51)

In [277]:

```
#onehotencoding for teacher_prefix
one_hot_encoding_teacher_prefix_cv_tfidf_w2v=pd.get_dummies(X_cv_tfidf_w2v.teacher_prefix)

print("Shape of dataframe for teacher_prefix", one_hot_encoding_teacher_prefix_cv_tfidf_w2v.shape)
```

Shape of dataframe for teacher_prefix (26219, 5)

In [291]:

In [271]:

```
#onehotencoding for teacher_prefix
one_hot_encoding_teacher_prefix_train_tfidf_w2v=pd.get_dummies(X_train_tfidf_w2v.teacher_prefix)

print("Shape of dataframe for teacher_prefix", one_hot_encoding_teacher_prefix_train_tfidf_w2v.shape)
```

Shape of dataframe for teacher_prefix (61179, 5)

In [278]:

```
#onehotencoding for teacher_prefix
one_hot_encoding_teacher_prefix_test_tfidf_w2v=pd.get_dummies(X_test_tfidf_w2v.teacher_prefix)

print("Shape of dataframe for teacher_prefix", one_hot_encoding_teacher_prefix_test_tfidf_w2v.shape)
```

Shape of dataframe for teacher_prefix (21850, 5)

In [279]:

```
#onehotencoding for project_grade_category
one_hot_encoding_project_grade_category_train_tfidf_w2v=pd.get_dummies(X_train_tfidf_w2v.project_grade_category)

print("Shape of dataframe for project_grade_category",
one_hot_encoding_project_grade_category_train_tfidf_w2v.shape)
```

Shape of dataframe for project_grade_category (61179, 4)

In [280]:

```
#onehotencoding for project_grade_category
one_hot_encoding_project_grade_category_cv_tfidf_w2v=pd.get_dummies(X_cv_tfidf_w2v.project_grade_category)

print("Shape of dataframe for project_grade_category",
one_hot_encoding_project_grade_category_cv_tfidf_w2v.shape)
```

Shape of dataframe for project_grade_category (26219, 4)

In [281]:

```
#onehotencoding for project_grade_category
one_hot_encoding_project_grade_category_test_tfidf_w2v=pd.get_dummies(X_test_tfidf_w2v.project_grade_category)

print("Shape of dataframe for project_grade_category",
one_hot_encoding_project_grade_category_test_tfidf_w2v.shape)
```

Shape of dataframe for project_grade_category (21850, 4)

In [282]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True)
categories_one_hot_train_tfidf_w2v = vectorizer.fit_transform(X_train_tfidf_w2v['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encoding ",categories_one_hot_train_tfidf_w2v.shape)
```

['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding (61179, 9)

In [283]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True)
categories_one_hot_cv_tfidf_w2v = vectorizer.transform(X_cv_tfidf_w2v['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",categories_one_hot_cv_tfidf_w2v.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig (26219, 9)
```

In [284]:

```
categories_one_hot_test_tfidf_w2v=
vectorizer.transform(X_test_tfidf_w2v['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",categories_one_hot_test_tfidf_w2v.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig (21850, 9)
```

In [285]:

```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=True)
sub_categories_one_hot_train_tfidf_w2v =
vectorizer.fit_transform(X_train_tfidf_w2v['clean_subcategories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",sub_categories_one_hot_train_tfidf_w2v.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL',
'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig (61179, 30)
```

In [286]:

```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=True)
sub_categories_one_hot_cv_tfidf_w2v= vectorizer.transform(X_cv_tfidf_w2v['clean_subcategories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",sub_categories_one_hot_cv_tfidf_w2v.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL',
'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig (26219, 30)
```

In [287]:

```
# we use count vectorizer to convert the values into one

sub_categories_one_hot_test_tfidf_w2v =
vectorizer.fit_transform(X_test_tfidf_w2v['clean_subcategories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",sub_categories_one_hot_test_tfidf_w2v.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
```

```
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL',
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig (21850, 30)
```

In [292]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatenating a sparse matrix and a dense matirx :)
tfidf_w2v_data_matrix_train=
hstack((one_hot_encoding_school_state_train_tfidf_w2v,one_hot_encoding_teacher_prefix_train_tfidf_w
v,one_hot_encoding_project_grade_category_train_tfidf_w2v,categories_one_hot_train_tfidf_w2v,sub_ca
tegories_one_hot_train_tfidf_w2v,price_standardized_train_tfidf_w2v,tfidf_w2v_vectors_preprocessed_
ssays_train,tfidf_w2v_vectors_project_title_train))
tfidf_w2v_data_matrix_train.shape
```

Out[292]:

```
(61179, 700)
```

In [289]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatenating a sparse matrix and a dense matirx :)
tfidf_w2v_data_matrix_cv=
hstack((one_hot_encoding_school_state_cv_tfidf_w2v,one_hot_encoding_teacher_prefix_cv_tfidf_w2v,on
e_hot_encoding_project_grade_category_cv_tfidf_w2v,categories_one_hot_cv_tfidf_w2v,sub_categories_c
one_hot_cv_tfidf_w2v,price_standardized_cv_tfidf_w2v,tfidf_w2v_vectors_preprocessed_essays_cv,tfidf
_w2v_vectors_project_title_cv))
tfidf_w2v_data_matrix_cv.shape
```

Out[289]:

```
(26219, 700)
```

In [293]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatenating a sparse matrix and a dense matirx :)
tfidf_w2v_data_matrix_test=
hstack((one_hot_encoding_school_state_test_tfidf_w2v,one_hot_encoding_teacher_prefix_test_tfidf_w2v
,one_hot_encoding_project_grade_category_test_tfidf_w2v,categories_one_hot_test_tfidf_w2v,sub_cateco
ries_one_hot_test_tfidf_w2v,price_standardized_test_tfidf_w2v,tfidf_w2v_vectors_preprocessed_essay
_test,tfidf_w2v_vectors_project_title_test))
tfidf_w2v_data_matrix_test.shape
```

Out[293]:

```
(21850, 700)
```

In [306]:

```
from scipy.sparse import coo_matrix
k= coo_matrix(tfidf_w2v_data_matrix_train)
k1 = k.tocsr()
```

In [307]:

```
new_tfidf_w2v_data_matrix_train=k1[:60001]
```

In [308]:

```
new_y_train_tfidf_w2v=y_train_tfidf_w2v[:60001]
```

In [309]:

```
from scipy.sparse import coo_matrix
```

```
from scipy.sparse import coo_matrix
k4 = coo_matrix(tfidf_w2v_data_matrix_cv)
k5 = k4.tocsr()
```

In [310]:

```
new_tfidf_w2v_data_matrix_cv=k5[:20001]
```

In [311]:

```
new_y_cv_tfidf_w2v=y_cv_tfidf_w2v[:20001]
```

In [312]:

```
from scipy.sparse import coo_matrix
k2 = coo_matrix(tfidf_w2v_data_matrix_test)
k3 = k2.tocsr()
```

In [313]:

```
new_tfidf_w2v_data_matrix_test=k3[:20001]
```

In [314]:

```
new_y_test_tfidf_w2v=y_test_tfidf_w2v[:20001]
```

In [315]:

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

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate until the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [316]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
from sklearn.linear_model import SGDClassifier
svm = linear_model.SGDClassifier(loss='hinge')
param_grid = {'alpha':[10000,1000,100,10,1,0.1,0.01,0.001,0.0001,0.00001,.000001], 'penalty':['l1','l2']}

clf = GridSearchCV(svm, param_grid, cv=10, scoring='roc_auc')
clf.fit(new_tfidf_w2v_data_matrix_train,new_y_train_tfidf_w2v)

train_auc_tfidf_w2v= clf.cv_results_['mean_train_score']
train_auc_std_tfidf_w2v= clf.cv_results_['std_train_score']
cv_auc_tfidf_w2v = clf.cv_results_['mean_test_score']
cv_auc_std_tfidf_w2v= clf.cv_results_['std_test_score']
train_auc_tfidf_w2v_new=train_auc_tfidf_w2v[11]
cv_auc_tfidf_w2v_new=cv_auc_tfidf_w2v[11]
train_auc_std_tfidf_w2v_new=train_auc_std_tfidf_w2v[11]

cv_auc_std_tfidf_w2v_new =cv_auc_std_tfidf_w2v[11]
```

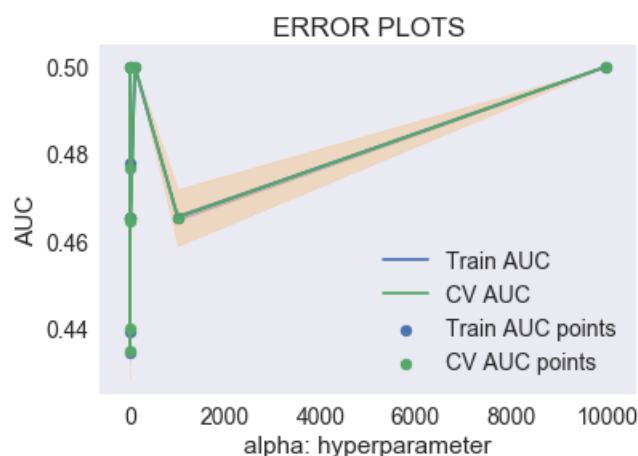
```
plt.plot(param_grid['alpha'], train_auc_tfidf_w2v_new, label='Train AUC')
```

```
plt.plot(param_grid['alpha'], train_auc_tfidf_w2v_new, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(param_grid['alpha'], train_auc_tfidf_w2v_new - train_auc_std_tfidf_w2v_new,
train_auc_tfidf_w2v_new + train_auc_std_tfidf_w2v_new, alpha=0.2, color='darkblue')

plt.plot(param_grid['alpha'], cv_auc_tfidf_w2v_new, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(param_grid['alpha'], cv_auc_tfidf_w2v_new -
cv_auc_std_tfidf_w2v_new, cv_auc_tfidf_w2v_new +
cv_auc_std_tfidf_w2v_new, alpha=0.2, color='darkorange')

plt.scatter(param_grid['alpha'], train_auc_tfidf_w2v_new, label='Train AUC points')
plt.scatter(param_grid['alpha'], cv_auc_tfidf_w2v_new, label='CV AUC points')

plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
```



In [318]:

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

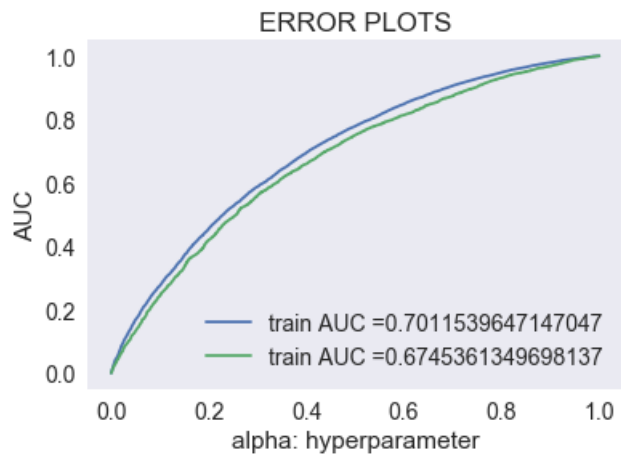
svm = linear_model.SGDClassifier(alpha=0.0001)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs
# https://www.kaggle.com/mpearmain/calibrated-sgdclassifier
model_calib_tfidf_w2v = CalibratedClassifierCV(base_estimator=svm, method='isotonic')
model_calib_tfidf_w2v.fit(new_tfidf_w2v_data_matrix_train, new_y_train_tfidf_w2v)
# model_calib.fit((new_bow_data_matrix_train,))

# test_labels = model_calib.predict_proba(np.array(test.values))[:,1]
y_train_pred_tfidf_w2v = batch_predict(model_calib_tfidf_w2v, new_tfidf_w2v_data_matrix_train)
y_test_pred_tfidf_w2v = batch_predict(model_calib_tfidf_w2v, new_tfidf_w2v_data_matrix_test)

train_fpr_tfidf_w2v, train_tpr_tfidf_w2v, tr_thresholds_tfidf_w2v =
roc_curve(new_y_train_tfidf_w2v, y_train_pred_tfidf_w2v)
test_fpr_tfidf_w2v, test_tpr_tfidf_w2v, te_thresholds_tfidf_w2v = roc_curve(new_y_test_tfidf_w2v,
y_test_pred_tfidf_w2v)

plt.plot(train_fpr_tfidf_w2v, train_tpr_tfidf_w2v, label="train AUC =" + str(auc(train_fpr_tfidf_w2v,
train_tpr_tfidf_w2v)))
plt.plot(test_fpr_tfidf_w2v, test_tpr_tfidf_w2v, label="train AUC =" + str(auc(test_fpr_tfidf_w2v, te
st_tpr_tfidf_w2v)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [319]:

```
# we are writing our own function for predict, with defined threshold
# we will pick a threshold that will give the least fpr
def predict(proba, threshold, fpr, tpr):

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

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

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [321]:

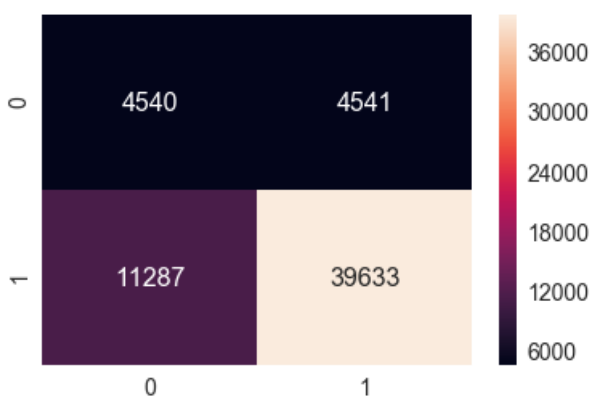
```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
df_cm_train=confusion_matrix(new_y_train_tfidf_w2v, predict(y_train_pred_tfidf_w2v,
tr_thresholds_tfidf_w2v, train_fpr_tfidf_w2v, train_fpr_tfidf_w2v))

sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2499999969683947 for threshold 0.82

Out[321]:

<matplotlib.axes._subplots.AxesSubplot at 0x1eb254c8828>



In [322]:

```
print("Test confusion matrix")

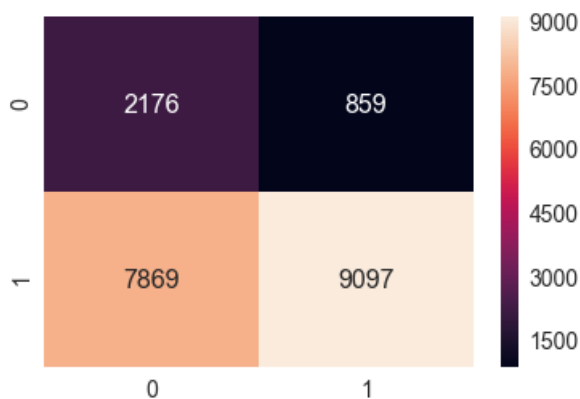
df_cm_test=confusion_matrix(new_y_test_tfidf_w2v, predict(y_test_pred_tfidf_w2v,
tr_thresholds_tfidf_w2v, test_fpr_tfidf_w2v, test_fpr_tfidf_w2v))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

Test confusion matrix

the maximum value of $tpr \cdot (1 - fpr)$ 0.2499999728592017 for threshold 0.857

Out[322]:

<matplotlib.axes._subplots.AxesSubplot at 0x1eb26358be0>



2.5 Support Vector Machines with added Features `Set 5`

In [51]:

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

In [52]:

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

16540843

In [53]:

```
project_data['my_counter_essay'] = my_counter_essay
```

In [54]:

```
project_data['neg']=0.01
```

In [55]:

```
project_data['neu']=0.745
```

In [56]:

```
project_data['pos']=0.245
```

In [57]:

```
project_data['compound']=0.9975
```

In [58]:

```
project_data.drop(['project_is_approved','project_essay_4','project_resource_summary','project_subn  
itted_datetime','teacher_id'], axis=1, inplace=True)
```

In [59]:

```
project_data.columns
```

Out[59]:

```
Index(['Unnamed: 0', 'id', 'teacher_prefix', 'school_state',  
      'project_grade_category', 'project_essay_1', 'project_essay_2',  
      'project_essay_3', 'teacher_number_of_previously_posted_projects',  
      'clean_categories', 'clean_subcategories', 'preprocessed_essays',  
      'project_title_list', 'price', 'quantity', 'my_counter_essay', 'neg',  
      'neu', 'pos', 'compound'],  
      dtype='object')
```

In [60]:

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

In [61]:

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

Out[61]:

	Unnamed: 0	id	teacher_prefix	school_state	project_grade_category	project_essay_1	project_essay_2	project_es
0	160221	p253737	Mrs.	IN	Grades PreK-2	My students are English learners that are work...	"The limits of your language are the limits o...	NaN
1	140945	p258326	Mr.	FL	Grades 6-8	Our students arrive to our school eager to lea...	The projector we need for our school is very c...	NaN

In [62]:

```
from sklearn.model_selection import train_test_split  
X1_train, X_test_new, y1_train, y_test_new = train_test_split(  
    project_data, y, test_size=0.20, stratify=y, random_state=42)  
X_cv_new, X_train_new, y_cv_new, y_train_new = train_test_split(X1_train, y1_train, test_size=0.70, stratif  
y=y1_train, random_state=42)
```

In [63]:

```
# We are considering only the words which appeared in at least 10 documents (rows or projects).  
vectorizer = TfidfVectorizer(ngram_range=(1, 2), min_df=10, max_features=5000)  
text_tfidf_essay_train = vectorizer.fit_transform(X_train_new['preprocessed_essays'])  
print("Shape of matrix after one hot encoding ", text_tfidf_essay_train.shape)
```

Shape of matrix after one hot encoding (61179, 5000)

In [65]:

```
#https://github.com/Dhruv5950/SVD-on-Amazon-Fine-food-Review/blob/master/Word-Vec_SVD_Amazon.ipynb
'''
Plotting a CDF on number of components and cumulative_variance_explained to choose the best value
of number of components
such that maximum variance is explained.
'''

from sklearn.decomposition import TruncatedSVD

svd_tf = TruncatedSVD(n_components = 140)
svd_data_tf = svd_tf.fit_transform(text_tfidf_essay_train)

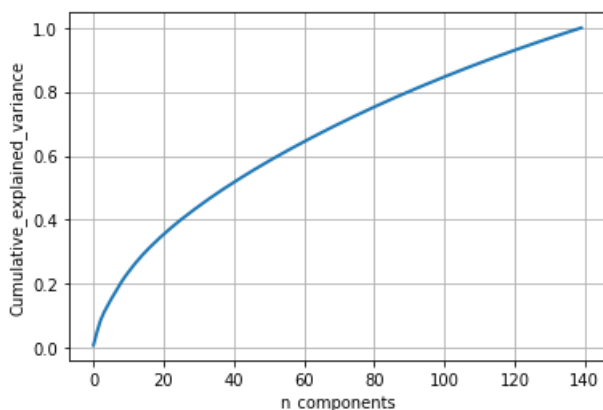
percentage_var_explained_tf = svd_tf.explained_variance_ / np.sum(svd_tf.explained_variance_);

cum_var_explained_tf = np.cumsum(percentage_var_explained_tf)

# Plot the PCA spectrum
plt.figure(1, figsize=(6, 4))

plt.clf()
plt.plot(cum_var_explained_tf, linewidth=2)
plt.axis('tight')
plt.grid()
plt.xlabel('n_components')
plt.ylabel('Cumulative_explained_variance')
plt.show()

# If we take 130-dimensions, approx. 98% of variance is explained.
```



In [74]:

```
#https://stackoverflow.com/questions/48424084/number-of-components-truncated-svd
vectorizer = TfidfVectorizer(ngram_range=(1, 2), min_df=10,max_features=5000)
svd = TruncatedSVD(n_components=130, algorithm='arpack')
tfidf_df = vectorizer.fit_transform(X_train_new['preprocessed_essays'])
df_essay_train = svd.fit_transform(tfidf_df)
```

In [125]:

```
df_essay_train.shape
```

Out[125]:

```
(61179, 130)
```

In [126]:

```
#https://stackoverflow.com/questions/48424084/number-of-components-truncated-svd
vectorizer = TfidfVectorizer(ngram_range=(1, 2), min_df=10,max_features=5000)
svd = TruncatedSVD(n_components=130, algorithm='arpack')
tfidf_df = vectorizer.fit_transform(X_cv_new['preprocessed_essays'])
```

```
df_essay_cv = svd.fit_transform(tfidf_df)
```

In [127]:

```
df_essay_cv.shape
```

Out[127]:

```
(26219, 130)
```

In [128]:

```
#https://stackoverflow.com/questions/48424084/number-of-components-truncated-svd
vectorizer = TfidfVectorizer(ngram_range=(1, 2), min_df=10,max_features=5000)
svd = TruncatedSVD(n_components=130, algorithm='arpack')
tfidf_df = vectorizer.fit_transform(X_test_new['preprocessed_essays'])
df_essay_test = svd.fit_transform(tfidf_df)
```

In [129]:

```
df_essay_test.shape
```

Out[129]:

```
(21850, 130)
```

In [75]:

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

# price_standardized = standardScaler.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329. ... 399. 287.
73 5.5 ].
# Reshape your data either using array.reshape(-1, 1)

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

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

```
Mean : 298.3190839994116, Standard deviation : 370.8546387731735
```

In [76]:

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

In [77]:

```
#onehotencoding for school_state
one_hot_encoding_school_state_train_new=pd.get_dummies(X_train_new.school_state)
print("Shape of dataframe for school_state", one_hot_encoding_school_state_train_new.shape)
```

```
Shape of dataframe for school_state (61179, 51)
```

In [78]:

```
#onehotencoding for school_state
one_hot_encoding_school_state_cv_new=pd.get_dummies(X_cv_new.school_state)
print("Shape of dataframe for school_state", one_hot_encoding_school_state_cv_new.shape)
```

Shape of dataframe for school_state (26219, 51)

In [79]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings("ignore")
# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329. ... 399. 287.
73 5.5 ].
# Reshape your data either using array.reshape(-1, 1)

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

# Now standardize the data with above maen and variance.
quantity_standardized_train_new = quantity_scalar.transform(X_train_new['quantity'].values.reshape
(-1, 1))
```

Mean : 17.09831805031138, Standard deviation : 27.280938801274228

In [80]:

```
# Now standardize the data with above maen and variance.
quantity_standardized_cv_new = quantity_scalar.transform(X_cv_new['quantity'].values.reshape(-1, 1)
)
```

In [81]:

```
#onehotencoding for teacher_prefix
one_hot_encoding_teacher_prefix_train_new=pd.get_dummies(X_train_new.teacher_prefix)

print("Shape of dataframe for teacher_prefix", one_hot_encoding_teacher_prefix_train_new.shape)
```

Shape of dataframe for teacher_prefix (61179, 5)

In [82]:

```
#onehotencoding for teacher_prefix
one_hot_encoding_teacher_prefix_cv_new=pd.get_dummies(X_cv_new.teacher_prefix)

print("Shape of dataframe for teacher_prefix", one_hot_encoding_teacher_prefix_cv_new.shape)
```

Shape of dataframe for teacher_prefix (26219, 5)

In [83]:

```
#onehotencoding for project_grade_category
one_hot_encoding_project_grade_category_train_new=pd.get_dummies(X_train_new.project_grade_category
)

print("Shape of dataframe for project_grade_category",
one_hot_encoding_project_grade_category_train_new.shape)
```

Shape of dataframe for project_grade_category (61179, 4)

In [84]:

```
#onehotencoding for project_grade_category
one_hot_encoding_project_grade_category_cv_new=pd.get_dummies(X_cv_new.project_grade_category)

print("Shape of dataframe for project grade category",
```

```
one_hot_encoding_project_grade_category_cv_new.shape)
```

Shape of dataframe for project_grade_category (26219, 4)

In [85]:

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

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding (61179, 9)
```

In [86]:

```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True)
categories_one_hot_cv_new = vectorizer.transform(X_cv_new['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encoding ",categories_one_hot_cv_new.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding (26219, 9)
```

In [87]:

```
categories_one_hot_test_new = vectorizer.transform(X_test_new['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encoding ",categories_one_hot_test_new.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding (21850, 9)
```

In [88]:

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

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL',
'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encoding (61179, 30)
```

In [89]:

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

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL',
'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig (26219, 30)
```

In [90]:

```
# we use count vectorizer to convert the values into one

sub_categories_one_hot_test_new = vectorizer.transform(X_test_new['clean_subcategories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ", sub_categories_one_hot_test_new.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL',
'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig (21850, 30)
```

In [91]:

```
price_standardized_test_new = price_scalar.transform(X_test_new['price'].values.reshape(-1, 1))
```

In [92]:

```
import warnings
warnings.filterwarnings("ignore")
quantity_standardized_test_new = price_scalar.transform(X_test_new['quantity'].values.reshape(-1, 1))
```

In [93]:

```
#onehotencoding for school_state
one_hot_encoding_school_state_test_new=pd.get_dummies(X_test_new.school_state)
print("Shape of dataframe for school_state", one_hot_encoding_school_state_test_new.shape)
```

Shape of dataframe for school_state (21850, 51)

In [94]:

```
#onehotencoding for teacher_prefix
one_hot_encoding_teacher_prefix_test_new=pd.get_dummies(X_test_new.teacher_prefix)

print("Shape of dataframe for teacher_prefix", one_hot_encoding_teacher_prefix_test_new.shape)
```

Shape of dataframe for teacher_prefix (21850, 5)

In [95]:

```
#onehotencoding for project_grade_category
one_hot_encoding_project_grade_category_test_new=pd.get_dummies(X_test_new.project_grade_category)

print("Shape of dataframe for project_grade_category",
one_hot_encoding_project_grade_category_test_new.shape)
```

Shape of dataframe for project_grade_category (21850, 4)

In [96]:

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

```

from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings("ignore")
# price_standardized = standardScaler.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329. ... 399. 287.
73 5.5 ].
# Reshape your data either using array.reshape(-1, 1)

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

# Now standardize the data with above maen and variance.
teacher_number_of_previously_posted_projects_standardized_train_new =
teacher_number_of_previously_posted_projects_scalar.transform(X_train_new['teacher_number_of_previc
usly_posted_projects'].values.reshape(-1, 1))

```

Mean : 11.1445757531179, Standard deviation : 27.875468900349503

In [97]:

```

import warnings
warnings.filterwarnings("ignore")
# Now standardize the data with above maen and variance.
teacher_number_of_previously_posted_projects_standardized_cv_new
=teacher_number_of_previously_posted_projects_scalar.transform(X_cv_new['teacher_number_of_previous
ly_posted_projects'].values.reshape(-1, 1))

```

In [98]:

```

import warnings
warnings.filterwarnings("ignore")
teacher_number_of_previously_posted_projects_standardized_test_new
=teacher_number_of_previously_posted_projects_scalar.transform(X_test_new['teacher_number_of_previc
usly_posted_projects'].values.reshape(-1, 1))

```

In [99]:

```

# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-
learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings("ignore")
# price_standardized = standardScaler.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329. ... 399. 287.
73 5.5 ].
# Reshape your data either using array.reshape(-1, 1)
my_counter_essay_scalar = StandardScaler()
my_counter_essay_scalar.fit(X_train_new['my_counter_essay'].values.reshape(-1,1)) # finding the mea
n and standard deviation of this data
print(f"Mean : {my_counter_essay_scalar.mean_[0]}, Standard deviation :
{np.sqrt(my_counter_essay_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
my_counter_essay_standardized_train_new =
my_counter_essay_scalar.transform(X_train_new['my_counter_essay'].values.reshape(-1, 1))

```

Mean : 16540843.0, Standard deviation : 0.0

In [100]:

```

import warnings
warnings.filterwarnings("ignore")
# Now standardize the data with above maen and variance.
my_counter_essay_standardized_cv_new =
my_counter_essay_scalar.transform(X_cv_new['my_counter_essay'].values.reshape(-1, 1))

```


In [101]:

```
import warnings
warnings.filterwarnings("ignore")
my_counter_essay_standardized_test_new =
my_counter_essay_scalar.transform(X_test_new['my_counter_essay'].values.reshape(-1, 1))
```

In [102]:

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

# price_standardized = standardScaler.fit(p
import warnings
warnings.filterwarnings("ignore")
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329. ... 399. 287.
73 5.5 ].
# Reshape your data either using array.reshape(-1, 1)
compound_scalar = StandardScaler()
compound_scalar.fit(X_train_new['compound'].values.reshape(-1,1)) # finding the mean and standard
deviation of this data
print(f"Mean : {compound_scalar.mean_[0]}, Standard deviation :
{np.sqrt(compound_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
compound_standardized_train_new =compound_scalar.transform(X_train_new['compound'].values.reshape(-1, 1))
```

Mean : 0.9975000000000003, Standard deviation : 2.220446049250313e-16

In [103]:

```
# Now standardize the data with above maen and variance.
compound_standardized_cv_new = compound_scalar.transform(X_cv_new['compound'].values.reshape(-1, 1)
)
```

In [104]:

```
compound_standardized_test_new = compound_scalar.transform(X_test_new['compound'].values.reshape(-1, 1))
```

In [105]:

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

# price_standardized = standardScaler.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329. ... 399. 287.
73 5.5 ].
# Reshape your data either using array.reshape(-1, 1)

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

# Now standardize the data with above maen and variance.
pos_standardized_train_new = pos_scalar.transform(X_train_new['pos'].values.reshape(-1, 1))
```

Mean : 0.24500000000000008, Standard deviation : 8.326672684688674e-17

In [106]:

```
# Now standardize the data with above maen and variance.
```

```
pos_standardized_cv_new = pos_scalar.transform(X_cv_new['pos'].values.reshape(-1, 1))
```

In [107]:

```
pos_standardized_test_new = pos_scalar.transform(X_test_new['pos'].values.reshape(-1, 1))
```

In [108]:

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

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329. ... 399. 287.
73 5.5 ].
# Reshape your data either using array.reshape(-1, 1)

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

# Now standardize the data with above maen and variance.
neg_standardized_train_new = neg_scalar.transform(X_train_new['neg'].values.reshape(-1, 1))
```

Mean : 0.010000000000000002, Standard deviation : 1.734723475976807e-18

In [109]:

```
# Now standardize the data with above maen and variance.
neg_standardized_cv_new = neg_scalar.transform(X_cv_new['neg'].values.reshape(-1, 1))
```

In [110]:

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

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329. ... 399. 287.
73 5.5 ].
# Reshape your data either using array.reshape(-1, 1)

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

# Now standardize the data with above maen and variance.
neu_standardized_train_new = neu_scalar.transform(X_train_new['neu'].values.reshape(-1, 1))
```

Mean : 0.74499999999999998, Standard deviation : 2.220446049250313e-16

In [111]:

```
neg_standardized_test_new = neg_scalar.transform(X_test_new['neg'].values.reshape(-1, 1))
```

In [112]:

```
# Now standardize the data with above maen and variance.
neu_standardized_cv_new = neu_scalar.transform(X_cv_new['neu'].values.reshape(-1, 1))
```

In [113]:

```
neu_standardized_test_new = neu_scalar.transform(X_test_new['neu'].values.reshape(-1, 1))
```

In [114]:

```
project_data.columns
```

Out[114]:

```
Index(['Unnamed: 0', 'id', 'teacher_prefix', 'school_state',  
      'project_grade_category', 'project_essay_1', 'project_essay_2',  
      'project_essay_3', 'teacher_number_of_previously_posted_projects',  
      'clean_categories', 'clean_subcategories', 'preprocessed_essays',  
      'price', 'quantity', 'my_counter_essay', 'neg', 'neu', 'pos',  
      'compound'],  
      dtype='object')
```

In [116]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039  
from scipy.sparse import hstack  
# with the same hstack function we are concatenating a sparse matrix and a dense matrix :)  
new_data_matrix_train=  
hstack((one_hot_encoding_school_state_train_new, one_hot_encoding_teacher_prefix_train_new, one_hot_encoding_project_grade_category_train_new, categories_one_hot_train_new, sub_categories_one_hot_train_new, price_standardized_train_new, neu_standardized_train_new, neg_standardized_train_new, pos_standardized_train_new, compound_standardized_train_new, df_essay_train, teacher_number_of_previously_posted_projects_standardized_train_new, quantity_standardized_train_new ))  
new_data_matrix_train.shape
```

Out[116]:

```
(61179, 236)
```

In [131]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039  
from scipy.sparse import hstack  
# with the same hstack function we are concatenating a sparse matrix and a dense matrix :)  
new_data_matrix_test=  
hstack((one_hot_encoding_school_state_test_new, one_hot_encoding_teacher_prefix_test_new, one_hot_encoding_project_grade_category_test_new, categories_one_hot_test_new, sub_categories_one_hot_test_new, price_standardized_test_new, neu_standardized_test_new, neg_standardized_test_new, pos_standardized_test_new, compound_standardized_test_new, df_essay_test, teacher_number_of_previously_posted_projects_standardized_test_new, quantity_standardized_test_new ))  
new_data_matrix_test.shape
```

Out[131]:

```
(21850, 236)
```

In [132]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039  
from scipy.sparse import hstack  
# with the same hstack function we are concatenating a sparse matrix and a dense matrix :)  
new_data_matrix_cv=  
hstack((one_hot_encoding_school_state_cv_new, one_hot_encoding_teacher_prefix_cv_new, one_hot_encoding_project_grade_category_cv_new, categories_one_hot_cv_new, sub_categories_one_hot_cv_new, price_standardized_cv_new, neu_standardized_cv_new, neg_standardized_cv_new, pos_standardized_cv_new, compound_standardized_cv_new, df_essay_cv, teacher_number_of_previously_posted_projects_standardized_cv_new, quantity_standardized_cv_new ))  
new_data_matrix_cv.shape
```

Out[132]:

```
(26219, 236)
```

In [133]:

```
from scipy.sparse import coo_matrix  
m = coo_matrix(new_data_matrix_train)  
m1 = m.tocsr()
```

In [134]:

```
new_data_matrix_train_f=m1[:60001]
```

In [135]:

```
new_data_matrix_train_f.shape
```

Out[135]:

```
(60001, 236)
```

In [136]:

```
new_y_train_f=y_train_new[:60001]
```

In [137]:

```
new_y_train_f.shape
```

Out[137]:

```
(60001,)
```

In [138]:

```
from scipy.sparse import coo_matrix
m2 = coo_matrix(new_data_matrix_test)
m3 = m2.tocsr()
```

In [139]:

```
new_data_matrix_test_f=m3[:20001]
```

In [140]:

```
new_y_test_f=y_test_new[:20001]
```

In [141]:

```
from scipy.sparse import coo_matrix
m4 = coo_matrix(new_data_matrix_cv)
m5 = m4.tocsr()
```

In [142]:

```
new_data_matrix_cv_f=m5[:20001]
```

In [143]:

```
new_y_cv_new=y_cv_new[:20001]
```

In [144]:

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

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate until the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
```

```
# we will be predicting for the last data points
y_data_pred.extend(clf.predict_proba(data[tr_loop:][:,1])

return y_data_pred
```

In [145]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
from sklearn.linear_model import SGDClassifier
svm = linear_model.SGDClassifier(loss='hinge')
param_grid = {'alpha':[10000,1000,100,10,1,0.1,0.01,0.001,0.0001,0.00001,.000001], 'penalty':['l1','l2']}

clf = GridSearchCV(svm, param_grid, cv=10, scoring='roc_auc')
clf.fit(new_data_matrix_train_f,new_y_train_f)

train_auc_f= clf.cv_results_['mean_train_score']
train_auc_std_f= clf.cv_results_['std_train_score']
cv_auc_f= clf.cv_results_['mean_test_score']
cv_auc_std_f= clf.cv_results_['std_test_score']
train_auc_f_new=train_auc_f[:11]
cv_auc_f_new=cv_auc_f[:11]
train_auc_std_f_new=train_auc_std_f[:11]

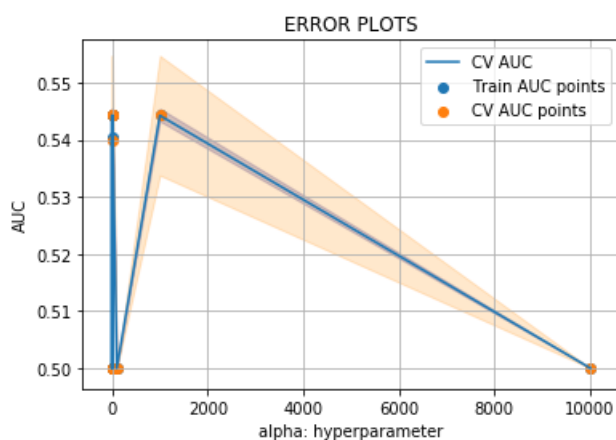
cv_auc_std_f_new =cv_auc_std_f[:11]

plt.gca().fill_between(param_grid['alpha'],train_auc_f_new - train_auc_std_f_new,train_auc_f_new +
train_auc_std_f_new,alpha=0.2,color='darkblue')

plt.plot(param_grid['alpha'], cv_auc_f_new, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(param_grid['alpha'],cv_auc_f_new - cv_auc_std_f_new,cv_auc_f_new +
cv_auc_std_f_new,alpha=0.2,color='darkorange')

plt.scatter(param_grid['alpha'], train_auc_f_new, label='Train AUC points')
plt.scatter(param_grid['alpha'], cv_auc_f_new, label='CV AUC points')

plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
```



In [148]:

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

from sklearn.calibration import CalibratedClassifierCV
svm = linear_model.SGDClassifier(alpha=0.001)
```

```

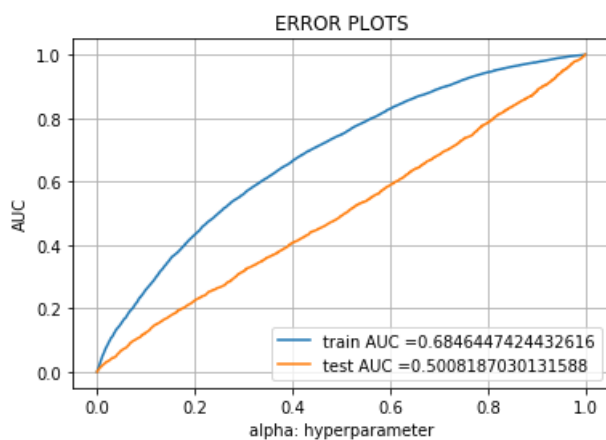
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted output
#https://www.kaggle.com/mpearmain/calibrated-sgdclassifier
model_calib_f = CalibratedClassifierCV(base_estimator=svm,method='isotonic')
model_calib_f.fit(new_data_matrix_train_f,new_y_train_f)

y_train_pred_f = batch_predict(model_calib_f,new_data_matrix_train_f )
y_test_pred_f = batch_predict(model_calib_f,new_data_matrix_test_f)

train_fpr_f, train_tpr_f, tr_thresholds_f = roc_curve(new_y_train_f, y_train_pred_f)
test_fpr_f, test_tpr_f, te_thresholds_f = roc_curve(new_y_test_f, y_test_pred_f)

plt.plot(train_fpr_f, train_tpr_f, label="train AUC =" +str(auc(train_fpr_f, train_tpr_f)))
plt.plot(test_fpr_f, test_tpr_f, label="test AUC =" +str(auc(test_fpr_f, test_tpr_f)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()

```



In [150]:

```

# we are writing our own function for predict, with defined threshold
# we will pick a threshold that will give the least fpr
def predict(proba, threshold, fpr, tpr):

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

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

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions

```

In [151]:

```

print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
df_cm_train=confusion_matrix(new_y_train_f, predict(y_train_pred_f, tr_thresholds_f, train_fpr_f, t
rain_fpr_f))

sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm_train, annot=True,annot_kws={"size": 16}, fmt='g')

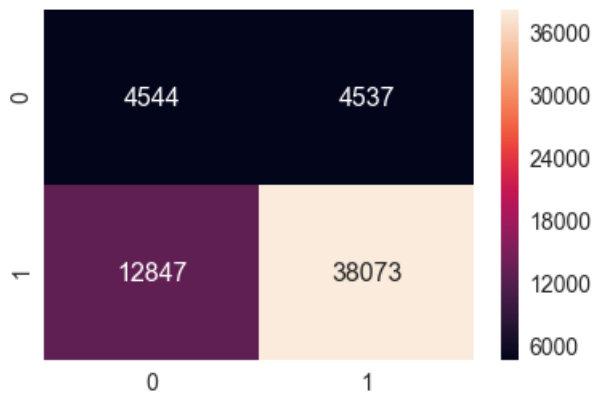
```

=====

Train confusion matrix
the maximum value of $tpr \cdot (1 - fpr)$ 0.2499998514513404 for threshold 0.825

Out[151]:

<matplotlib.axes._subplots.AxesSubplot at 0x220e704bf28>



In [152]:

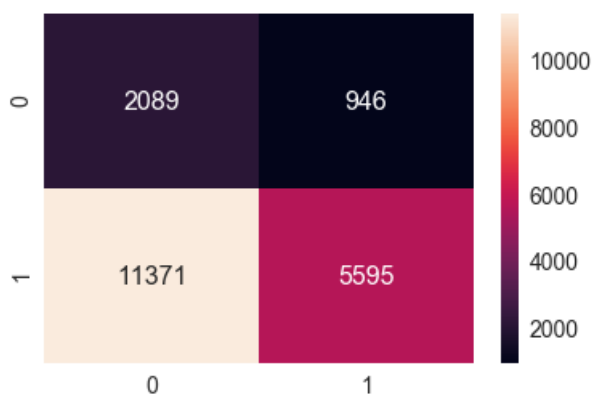
```
print("="*100)
from sklearn.metrics import confusion_matrix
print("test confusion matrix")
df_cm_test=confusion_matrix(new_y_test_f, predict(y_test_pred_f, tr_thresholds_f, test_fpr_f,
test_fpr_f))

sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

test confusion matrix
the maximum value of $tpr \cdot (1 - fpr)$ 0.2499999728592017 for threshold 0.876

Out[152]:

<matplotlib.axes._subplots.AxesSubplot at 0x220ea4994e0>



3. Conclusion

In [394]:

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

In [1]:

```
# http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
```

#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable

```
x = PrettyTable()
x.field_names = ["Featurization", "train_auc", "test_auc", "threshold_for train", "tpr*(1-fpr) for train", "threshold_for_test", "tpr*(1-fpr) for test" ,]
x.add_row(["BOW", 0.79, 0.70, 0.78, 0.249, 0.866, 0.249 ])
x.add_row(["TFIDF", 0.78, 0.5, 0.781, 0.2499, 0.010, 1.996 ])
x.add_row(["AVG_W2V", 0.69, 0.65, 0.827, 0.25, 0.838, 0.249])
x.add_row(["TFIDF_W2v", 0.70, 0.67, 0.82, 0.249, 0.857, 0.249])
x.add_row(["Engineered features", 0.68, 0.50, 0.825, 0.249, 0.876, .249])
print(x)
```

	Featurization	train_auc	test_auc	threshold_for train	tpr*(1-fpr) for train	threshold_for_test	tpr*(1-fpr) for test
866	BOW	0.79	0.7	0.78	0.249		
	TFIDF	0.78	0.5	0.781	0.2499		
.01		1.996					
838	AVG_W2V	0.69	0.65	0.827	0.25		
		0.249					
	TFIDF_W2v	0.7	0.67	0.82	0.249		
857		0.249					
	Engineered features	0.68	0.5	0.825	0.249		
.876		0.249					