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
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located (Two-letter U.S. postal code). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy`

- Literature & Writing, Social Sciences

Feature	Description
project_resource_summary	An explanation of the resources needed for the project. **Example:** • My students need hands on literacy materials to manage sensory needs!
project_essay_1	First application essay*
project_essay_2	Second application essay*
project_essay_3	Third application essay*
project_essay_4	Fourth application essay*
project_submitted_datetime	Datetime when project application was submitted. **Example:** 2016-04-28 12:43:56.245
teacher_id	A unique identifier for the teacher of the proposed project. **Example:** bdf8baa8fedef6bfeec7ae4ff1c15c56
teacher_prefix	Teacher's title. One of the following enumerated values: • nan • Dr. • Mr. • Mrs. • Ms. • Teacher.
teacher_number_of_previously_posted_projects	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
id	A `project_id` value from the `train.csv` file. **Example:** p036502
description	Desciption of the resource. **Example:** Tenor Saxophone Reeds, Box of 25
quantity	Quantity of the resource required. **Example:** 3
price	Price of the resource required. **Example:** 9.95

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

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

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

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

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

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [6]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python:
https://stackoverflow.com/a/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_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    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	897464ce9ddc600bced1151f324dd63a	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. W
e are a melting pot of refugees, immigrants, and native-born Americans bringing the gift of langua
ge to our school. \r\n\r\n We have over 24 languages represented in our English Learner program wi
th 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 o
f your world.\"-Ludwig Wittgenstein  Our English learner's have a strong support system at home th
at begs for more resources.  Many times our parents are learning to read and speak English along s
ide of their children.  Sometimes this creates barriers for parents to be able to help their child
learn phonetics, letter recognition, and other reading skills.\r\n\r\nBy providing these dvd's and
players, students are able to continue their mastery of the English language even if no one at hom
e is able to assist.  All families with students within the Level 1 proficiency status, will be a
offered to be a part of this program.  These educational videos will be specially chosen by the En
glish Learner Teacher and will be sent home regularly to watch.  The videos are to help the child
develop early reading skills.\r\n\r\nParents 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 ed
ucational dvd's for the years to come for other EL students.\r\nnannan
==================================================
The 51 fifth grade students that will cycle through my classroom this year all love learning, at l
east most of the time. At our school, 97.3% of the students receive free or reduced price lunch. O
f the 560 students, 97.3% are minority students. \r\nThe school has a vibrant community that loves
to get together and celebrate. Around Halloween there is a whole school parade to show off the bea
utiful 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 t
he hard work put in during the school year, with a dunk tank being the most popular activity.My st
udents 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 hav
e 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 us
ed by the students who need the highest amount of movement in their life in order to stay focused
on school.\r\n\r\nWhenever asked what the classroom is 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 i
n 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 ta
ken. There are always students who head over to the kidney table to get one of the stools who are
disappointed as there are not enough of them. \r\n\r\nWe ask a lot of students to sit for 7 hours
a day. The Hokki stools will be a compromise that allow my students to do desk work and move at th
e 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 schools for a child who can't sit
still.nannan
==================================================
How do you remember your days of school? Was it in a sterile environment with plain walls, rows of
desks, and a teacher in front of the room? A typical day in our room is nothing like that. I work
hard to create a warm inviting themed room for my students look forward to coming to each day.\r\n
\r\nMy class is made up of 28 wonderfully unique boys and girls of mixed races in
Arkansas.\r\nThey attend a Title I school, which means there is a high enough percentage of free a
nd reduced-price lunch to qualify. Our school is an \"open classroom\" concept, which is very uniq
ue as there are no walls separating the classrooms. These 9 and 10 year-old students are very eage
r learners; they are like sponges, absorbing all the information and experiences and keep on wanti
ng more.With these resources such as the comfy red throw pillows and the whimsical nautical hangin
g 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 pic
tures of each child with them, have them developed, and then hung in our classroom ready for their
first day of 4th grade.  This kind gesture will set the tone before even the first day of school!
The nautical thank you cards will be used throughout the year by the students as they create thank
you cards to their team groups.\r\n\r\nYour generous donations will help me to help make our
classroom a fun, inviting, learning environment from day one.\r\n\r\nIt costs lost of money out of
my own pocket on resources to get our classroom ready. Please consider helping with this project t
o make our new school year a very successful one. Thank you!nannan
```

```
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, cogniti
ve delays, gross/fine motor delays, to autism. They are eager beavers and always strive to work th
eir 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 pr
ice 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 gr
oove and move as you were in a meeting? This is how my kids feel all the time. The want to be able
to move as they learn or so they say.Wobble chairs are the answer and I love then because they dev
elop their core, which enhances gross motor and in Turn fine motor skills. \r\nThey also want to l
earn through games, my kids don't want to sit and do worksheets. They want to learn to count by ju
mping and playing. Physical engagement is the key to our success. The number toss and color and sh
ape 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 grea
t teacher inspires. -William A. Ward\r\n\r\nMy school has 803 students which is makeup is 97.6% Af
rican-American, making up the largest segment of the student body. A typical school in Dallas is m
ade up of 23.2% African-American students. Most of the students are on free or reduced lunch. We a
ren'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 smar
t, effective, efficient, and disciplined students with good character.In our classroom we can util
ize the Bluetooth for swift transitions during class. I use a speaker which doesn't amplify the so
und enough to receive the message. Due to the volume of my speaker my students can't hear videos o
r books clearly and it isn't making the lessons as meaningful. But with the bluetooth speaker my s
tudents will be able to hear and I can stop, pause and replay it at any time.\r\nThe cart will all
ow me to have more room for storage of things that are needed for the day and has an extra part to
it I can use.  The table top chart has all of the letter, words and pictures for students to learn
about different letters and it is more accessible.nannan
==================================================
```

In [11]:

```python
# 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]:

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

```
My kindergarten students have varied disabilities ranging from speech and language delays, cogniti
ve delays, gross/fine motor delays, to autism. They are eager beavers and always strive to work th
eir 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 pr
ice 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 gr
oove and move as you were in a meeting? This is how my kids feel all the time. The want to be able
to move as they learn or so they say.Wobble chairs are the answer and I love then because they dev
elop their core, which enhances gross motor and in Turn fine motor skills. \r\nThey also want to l
earn through games, my kids do not want to sit and do worksheets. They want to learn to count by j
umping and playing. Physical engagement is the key to our success. The number toss and color and s
hape 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('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

```
My kindergarten students have varied disabilities ranging from speech and language delays, cogniti
ve delays, gross/fine motor delays, to autism. They are eager beavers and always strive to work th
eir 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 eag
er to learn and explore.Have you ever felt like you had ants in your pants and you needed to groov
e and move as you were in a meeting? This is how my kids feel all the time. The want to be able to
move as they learn or so they say.Wobble chairs are the answer and I love then because they develo
p their core, which enhances gross motor and in Turn fine motor skills.   They also want to learn t
hrough 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 ma
ts can make that happen. My students will forget they are doing work and just have the fun a 6 yea
r 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 cognitiv
e 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 studen
ts 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 lea
rn and explore Have you ever felt like you had ants in your pants and you needed to groove and mov
e as you were in a meeting This is how my kids feel all the time The want to be able to move as th
ey learn or so they say Wobble chairs are the answer and I love then because they develop their co
re 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 Ph
ysical 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 nan
nan
```

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', 'e
ach', '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', "do
esn'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
[01:45<00:00, 1033.79it/s]
```

In [17]:

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

Out[17]:

```
'my kindergarten students varied disabilities ranging speech language delays cognitive delays gros
s 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 lunc
h 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 w
obble 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 de
serves nannan'
```

1.4 Preprocessing of `project_title`

In [18]:

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

In [19]:

```
# 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())
```

```
100%|
███████████████████████████████████████████████████████████████████████████████████████████████
███████████████████████████████████████████████████████████████████████████████| 109248/109248
[00:04<00:00, 24016.69it/s]
```

In [20]:

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

In [21]:

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

1.5 Preparing data for models

In [22]:

```
project_data.columns
```

Out[22]:

```
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', 'essay',
       '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 [23]:

```
# 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 [24]:

```
# 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 [25]:

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

In [26]:

```
#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 [27]:

```
#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.sha
pe)
```

```
Shape of dataframe for project_grade_category (109248, 4)
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [28]:

```
# 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 [29]:

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

In [30]:

```
# 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)
```

1.5.2.2 TFIDF vectorizer

In [31]:

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

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

1.5.2.3 Using Pretrained Models: Avg W2V

In [32]:

```
'''
# 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-sa
ve-and-load-variables-in-python/

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


'''
```

Out[32]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef
loadGloveModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFile,\'r\',
encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n        splitLine = line.split()\n
word = splitLine[0]\n        embedding = np.array([float(val) for val in splitLine[1:]])\n        m
odel[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 preproced_texts:\n    words.extend(i.split(\'
----------------------------\n\nwords = []\nfor i in preproced_texts:\n    words.extend(i.split(\
\'))\n\nfor i in preproced_titles:\n    words.extend(i.split(\' \'))\nprint("all the words in the
coupus", len(words))\nwords = set(words)\nprint("the unique words in the coupus",
len(words))\n\ninter_words = set(model.keys()).intersection(words)\nprint("The number of words 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\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 [33]:

```
# 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 [34]:

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

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

```
100%|
████████████████████████████████████████████████████████████████████████████████████████████████████
██████████████████████████████████████████████████████████████████████████████████| 109248/109248
[01:11<00:00, 1531.62it/s]
```

```
109248
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [35]:

```
# 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 [36]:

```
# 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
            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 [06:19<00:00, 287.74it/s]
```

```
109248
300
```

In [37]:

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

In [38]:

```
# 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%|
████████████████████████████████████████████████████████████████████████████████████████████████████
███████████████████████████████████████████████████████████████████████████████████| 109248/109248
[00:05<00:00, 18433.49it/s]
```

```
109248
300
```

In [39]:

```
# 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 [40]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_project_title = []; # the avg-w2v for each sentence/review is stored in this lis
t
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%|
████████████████████████████████████████████████████████████████████████████████████████████████████
███████████████████████████████████████████████████████████████████████████████████| 109248/109248
[00:09<00:00, 11938.68it/s]
```

```
109248
300
```

1.5.3 Vectorizing Numerical features

In [41]:

```
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 [42]:

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

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

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

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

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

In [43]:

```
price_standardized
```

Out[43]:

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

1.5.4 Merging all the above features

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

In [44]:

```
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 [45]:

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

Out[45]:

```
(109248, 16663)
```

In [46]:

```
# 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 [47]:

```
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
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
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 b
e available.
```

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

In [48]:

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

In [49]:

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

In [50]:

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

In [51]:

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

Assignment 8: DT

1. **Apply Decision Tree Classifier(DecisionTreeClassifier) on these feature sets**
   - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (BOW)
   - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
   - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
   - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)

2. **Hyper paramter tuning (best `depth` in range [1, 5, 10, 50, 100, 500, 100], and the best `min_samples_split` in range [5, 10, 100, 500])**
   - Find the best hyper parameter which will give the maximum AUC value
   - Find the best hyper paramter using k-fold cross validation or simple cross validation data
   - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

3. **Graphviz**
   - Visualize your decision tree with Graphviz. It helps you to understand how a decision is being made, given a new vector.
   - Since feature names are not obtained from word2vec related models, visualize only BOW & TFIDF decision trees using Graphviz
   - Make sure to print the words in each node of the decision tree instead of printing its index.
   - Just for visualization purpose, limit max_depth to 2 or 3 and either embed the generated images of graphviz in your notebook, or directly upload them as .png files.

4. **Representation of results**

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure
- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
- Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points
- Once after you plot the confusion matrix with the test data, get all the `false positive data points`
    - Plot the WordCloud WordCloud
    - Plot the box plot with the `price` of these `false positive data points`
    - Plot the pdf with the `teacher_number_of_previously_posted_projects` of these `false positive data points`

5. **[Task-2]**

    - Select 5k best features from features of Set 2 using`feature_importances_`, discard all the other remaining features and then apply any of the model of you choice i.e. (Dession tree, Logistic Regression, Linear SVM), you need to do hyperparameter tuning corresponding to the model you selected and procedure in step 2 and step 3

6. **Conclusion**

    - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

2. Decision Tree

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

In [52]:

```
# 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 [53]:

```
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,stratif
y=y1_train,random_state=42)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [54]:

```
# 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 [55]:

```
# 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 [127]:

```
terms9=list(price_standardized_train_bow)
```

In [56]:

```
# 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 [57]:

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

In [58]:

```
#onehotencoding for school_state
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 [59]:

```
#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 [60]:

```
#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 [117]:

```
terms5=one_hot_encoding_school_state_test_bow.columns.values
```

In [62]:

```
#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 [124]:

```
terms7=list(one_hot_encoding_teacher_prefix_train_bow.columns.values)
```

In [63]:

```
#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 [125]:

```
terms8=list(one_hot_encoding_teacher_prefix_cv_bow.columns.values)
```

In [64]:

```
#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 [65]:

```
#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 [66]:

```
#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 [67]:

```
#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 [109]:

```
# we use count vectorizer to convert the values into one
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer4 = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False,
binary=True)
categories_one_hot_train_bow = vectorizer4.fit_transform(X_train_bow['clean_categories'].values)
print(vectorizer4.get_feature_names())
print("Shape of matrix after one hot encodig ",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 encodig  (61179, 9)
```

In [110]:

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

categories_one_hot_cv_bow = vectorizer4.transform(X_cv_bow['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",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 encodig  (26219, 9)
```

In [111]:

```
categories_one_hot_test_bow = vectorizer4.transform(X_test_bow['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",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 encodig  (21850, 9)
```

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_train_bow =
vectorizer.fit_transform(X_train_bow['clean_subcategories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",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 encodig  (61179, 30)
```

In [72]:

```
# 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 encodig ",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 encodig  (26219, 30)
```

In [73]:

```
# 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 encodig ",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 encodig  (21850, 30)
```

2.3 Make Data Model Ready: encoding eassay, and project_title

In [74]:

```
# 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 [75]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer1 = CountVectorizer()
text_bow_essay_train = vectorizer1.fit_transform(X_train_bow['preprocessed_essays'])
print("Shape of matrix after one hot encodig ",text_bow_essay_train.shape)
```

```
Shape of matrix after one hot encodig  (61179, 44988)
```

In [76]:

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

text_bow_essay_cv = vectorizer1.transform(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, 44988)
```

In [77]:

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

text_bow_essay_test = vectorizer1.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, 44988)
```

In [78]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer3 = CountVectorizer()
text_bow_project_title_train = vectorizer3.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, 12874)
```

In [79]:

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

text_bow_project_title_cv= vectorizer3.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, 12874)
```

In [80]:

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

text_bow_project_title_test = vectorizer3.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, 12874)
```

2.4 Appling Decision Tree on different kind of featurization as mentioned in the instructions

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

In [81]:

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

2.4.1 Applying Decision Trees on BOW, SET 1

In [82]:

```
# Please write all the code with proper documentation
```

In [83]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
bow_data_matrix_train=
hstack((one_hot_encoding_school_state_train_bow,one_hot_encoding_teacher_prefix_train_bow,one_hot_e
ncoding_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[83]:

```
(61179, 57962)
```

In [84]:

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

Out[84]:

```
(26219, 57962)
```

In [85]:

```
# 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 :)
bow_data_matrix_test=
hstack((one_hot_encoding_school_state_test_bow,one_hot_encoding_teacher_prefix_test_bow,one_hot_enc
oding_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[85]:

```
(21850, 57962)
```

In [86]:

```
y_train_bow.shape
```

Out[86]:

```
(61179,)
```

In [87]:

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

In [88]:

```
new_bow_data_matrix_train=m1[:30001]
```

In [89]:

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

In [90]:

```
new_y_train_bow=y_train_bow[:30001]
```

In [91]:

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

In [92]:

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

In [93]:

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

In [94]:

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

In [95]:

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

In [96]:

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

In [97]:

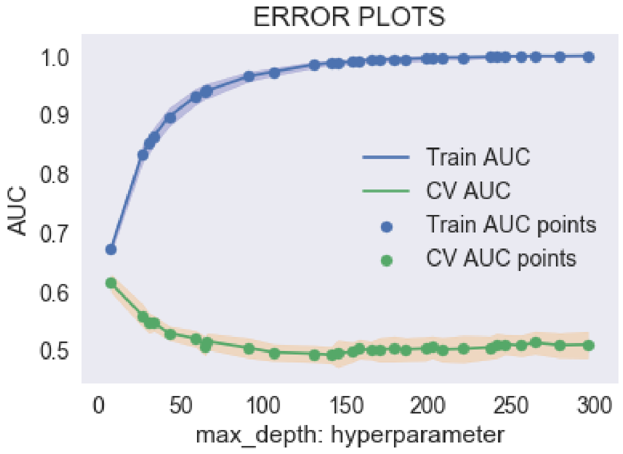
```
new_y_cv_bow=y_cv_bow[:20001]
```

In [98]:

```
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_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 unti 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
```

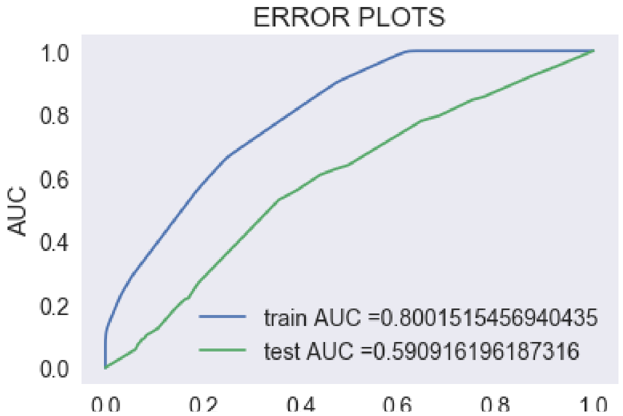
In [128]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from sklearn import tree
from scipy.stats import randint as sp_randint

g = sp_randint(2,300)
dt= tree.DecisionTreeClassifier()
param_grid = {'max_depth':sorted(g.rvs(30))}
clf = GridSearchCV(dt, 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']

plt.plot(param_grid['max_depth'], train_auc_bow, label='Train AUC')




# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(param_grid['max_depth'],train_auc_bow - train_auc_std_bow,train_auc_bow +
train_auc_std_bow,alpha=0.2,color='darkblue')

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

plt.scatter(param_grid['max_depth'], train_auc_bow, label='Train AUC points')
plt.scatter(param_grid['max_depth'], cv_auc_bow, label='CV AUC points')


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



In [176]:

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


dt= tree.DecisionTreeClassifier(max_depth=20)
dt.fit(new_bow_data_matrix_train, new_y_train_bow)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred_bow = batch_predict(dt,new_bow_data_matrix_train )
y_test_pred_bow = batch_predict(dt, 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(": hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [177]:

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

    t = threshould[np.argmax(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
```

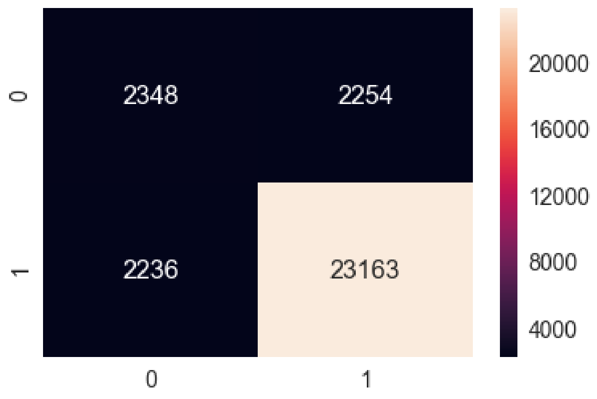
In [178]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
df_cm_train=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_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
====================================================================================================

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24989569563409972 for threshold 0.807
```

Out[178]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2dd68791c88>
```



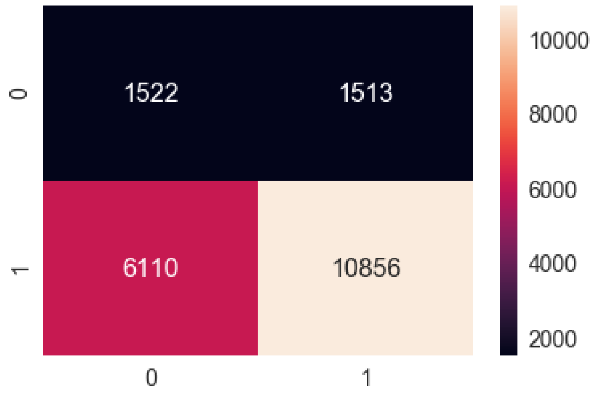
In [179]:

```
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*(1-fpr) 0.24999780159533613 for threshold 0.863
```

Out[179]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2dd66293d68>
```



2.4.1.1 Graphviz visualization of Decision Tree on BOW, SET 1

In [105]:

```
# Please write all the code with proper documentation
```

In [148]:

```
from sklearn import tree
clf_bow = tree.DecisionTreeClassifier(max_depth=2,min_samples_split=3000)
clf_bow = clf_bow.fit(new_bow_data_matrix_train, new_y_train_bow)
```

In [139]:

```
terms1=vectorizer3.get_feature_names()
```

In [140]:

```
terms3=vectorizer1.get_feature_names()
```

In [141]:

```
terms2=vectorizer.get_feature_names()
```

In [142]:

```
terms4=vectorizer4.get_feature_names()
```

In [143]:

```
terms6=list(terms5)
```

In [144]:

```
final=terms1+terms2+terms3+terms4+terms6+terms7+terms8+terms9
```

In [145]:

```
final=final_list[:57962]
```

In [149]:

```
import graphviz
dot_data = tree.export_graphviz(clf_bow, out_file=None, max_depth=2,feature_names=final)
bow_graph = graphviz.Source(dot_data)
bow_graph.render("bow_tree")
```

Out[149]:

```
'bow_tree.pdf'
```

```
    _
```

In [150]:

```
bow_graph
```

Out[150]:

2.4.2 Applying Decision Trees on TFIDF, SET 2

In [110]:

```
# Please write all the code with proper documentation
```

In [151]:

```
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 [152]:

```
# 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_tfidf['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_tfidf= price_scalar.transform(X_train_tfidf['price'].values.reshape(-1, 1)
)
```

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

In [184]:

```
terms1_tfidf=list(price_standardized_train_bow)
```

In [160]:

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

In [161]:

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

In [162]:

```
#onehotencoding for teacher_prefix
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 [185]:

```
terms2_tfidf=list(one_hot_encoding_school_state_train_tfidf.columns.values)
```

In [186]:

```
#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 [187]:

```
#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 [188]:

```
#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 [190]:

```
terms3_tfidf=list( one_hot_encoding_teacher_prefix_train_tfidf.columns.values)
```

In [191]:

```
#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 [192]:

```
#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 [193]:

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

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 [195]:

```
terms4_tfidf= list( one_hot_encoding_project_grade_category_train_tfidf.columns.values)
```

In [196]:

```
#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 [197]:

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

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 [198]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer =  TfidfVectorizer(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 [200]:

```
terms5_tfidf=vectorizer.get_feature_names()
```

In [201]:

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

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 [202]:

```
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 [203]:

```
# we use count vectorizer to convert the values into one
vectorizer1 =  TfidfVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False,
binary=True)
sub_categories_one_hot_train_tfidf= vectorizer1.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)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig  (61179, 30)
```

In [205]:

```
terms6_tfidf=vectorizer1.get_feature_names()
```

In [181]:

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


sub_categories_one_hot_cv_tfidf = vectorizer1.transform(X_cv_tfidf['clean_subcategories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",sub_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, 30)
```

In [182]:

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

sub_categories_one_hot_test_tfidf =
vectorizer1.transform(X_test_tfidf['clean_subcategories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",sub_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, 30)
```

In [206]:

```
vectorizer7 = TfidfVectorizer()
text_tfidf_essay_train = vectorizer7.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, 44988)
```

In [220]:

```
terms7_tfidf=vectorizer7.get_feature_names()
```

In [222]:

```
type(terms7_tfidf)
```

Out[222]:

```
list
```

In [209]:

```
text_tfidf_essay_cv = vectorizer7.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, 44988)
```

In [210]:

```
text_tfidf_essay_test = vectorizer7.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, 44988)
```

In [211]:

```
vectorizer8 = TfidfVectorizer()
text_tfidf_project_title_train = vectorizer8.fit_transform(X_train_tfidf['project_title_list'])
print("Shape of matrix after one hot encodig ",text_tfidf_project_title_train.shape)
```

```
Shape of matrix after one hot encodig  (61179, 12874)
```

In [212]:

```
terms8_tfidf=   vectorizer8.get_feature_names()
```

In [213]:

```
text_tfidf_project_title_cv= vectorizer8.transform(X_cv_tfidf['project_title_list'])
print("Shape of matrix after one hot encodig ",text_tfidf_project_title_cv.shape)
```

```
Shape of matrix after one hot encodig  (26219, 12874)
```

In [214]:

```
text_tfidf_project_title_test = vectorizer8.transform(X_test_tfidf['project_title_list'])
print("Shape of matrix after one hot encodig ",text_tfidf_project_title_test.shape)
```

```
Shape of matrix after one hot encodig  (21850, 12874)
```

In [256]:

```
final_tfidf=terms8_tfidf+terms7_tfidf+terms6_tfidf+terms5_tfidf+terms4_tfidf+terms3_tfidf+terms2_tf
idf+terms1_tfidf
```

In [215]:

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

Out[215]:

```
(61179, 57962)
```

In [228]:

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

Out[228]:

```
(26219, 57962)
```

In [229]:

```
# 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_ho
t_encoding_project_grade_category_test_tfidf,categories_one_hot_test_tfidf,sub_categories_one_hot_t
est_tfidf,price_standardized_test_tfidf,text_tfidf_essay_test,text_tfidf_project_title_test))
tfidf_data_matrix_test.shape
```

Out[229]:

```
(21850, 57962)
```

In [230]:

```
y_train_tfidf.shape
```

Out[230]:

```
(61179,)
```

In [231]:

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

In [232]:

```
new_tfidf_data_matrix_train=m1[:30001]
```

In [233]:

```
new_y_train_tfidf=y_train_tfidf[:30001]
```

In [234]:

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

In [235]:

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

In [236]:

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

In [237]:

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

In [238]:

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

In [239]:

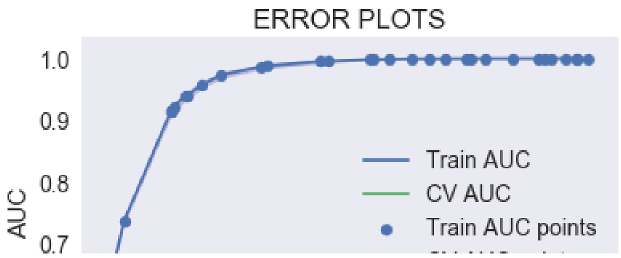
```
new_y_cv_tfidf=y_cv_tfidf[:20001]
```

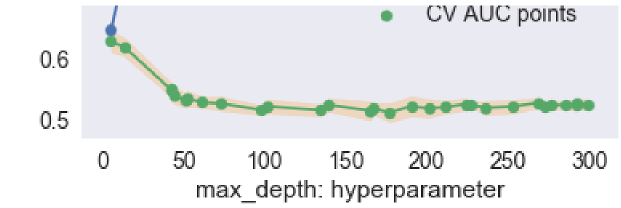
In [106]:

```
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_tfidf = []
    tr_loop_tfidf = 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 unti the last 1000 multiplier
    for i in range(0, tr_loop_tfidf, 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_tfidf:])[:,1])

    return y_data_pred_tfidf
```

In [180]:

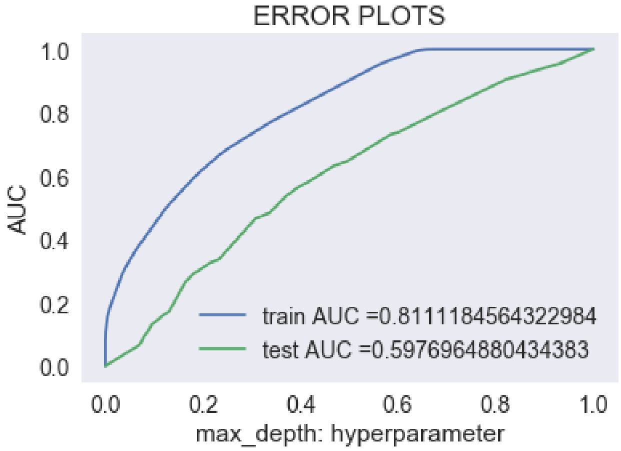
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from sklearn import tree
from scipy.stats import randint as sp_randint

g = sp_randint(2,300)
dt= tree.DecisionTreeClassifier()
param_grid = {'max_depth':sorted(g.rvs(30))}
clf = GridSearchCV(dt, 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= clf.cv_results_['std_test_score']

plt.plot(param_grid['max_depth'], train_auc_tfidf, label='Train AUC')


# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(param_grid['max_depth'],train_auc_tfidf -
train_auc_std_tfidf,train_auc_tfidf+ train_auc_std_tfidf,alpha=0.2,color='darkblue')

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

plt.scatter(param_grid['max_depth'], train_auc_tfidf, label='Train AUC points')
plt.scatter(param_grid['max_depth'], cv_auc_tfidf, label='CV AUC points')


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





In [182]:

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

dt= tree.DecisionTreeClassifier(max_depth=20)
dt.fit(new_tfidf_data_matrix_train, new_y_train_tfidf)

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

y_train_pred_tfidf = batch_predict(dt,new_tfidf_data_matrix_train )
y_test_pred_tfidf = batch_predict(dt, 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_t
fidf)))
plt.plot(test_fpr_tfidf, test_tpr_tfidf, label="test AUC ="+str(auc(test_fpr_tfidf, test_tpr_tfidf)
))
plt.legend()
plt.xlabel("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [183]:

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

    t = threshould[np.argmax(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
```

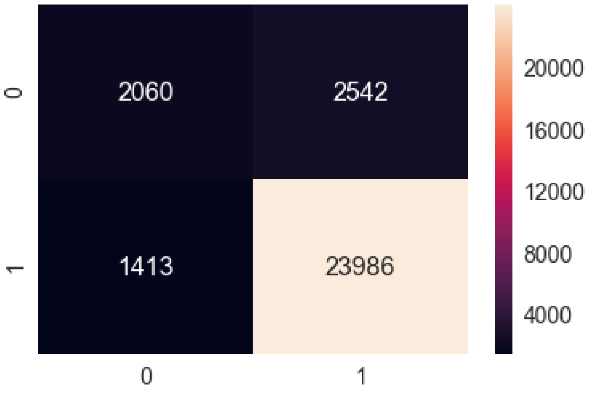
In [184]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
df_cm_train=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_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
====================================================================================================

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24725753649802884 for threshold 0.818
```

Out[184]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2dd1fa8f0f0>
```



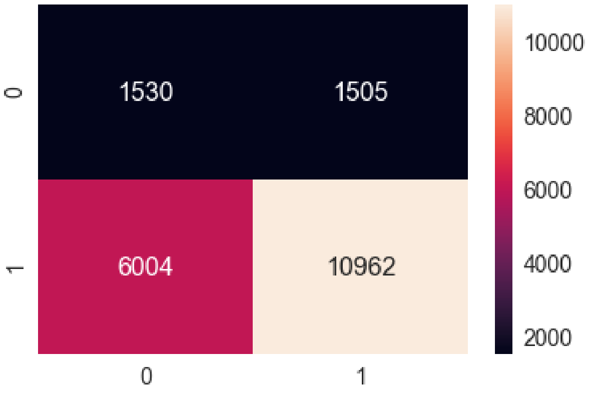
In [185]:

```
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*(1-fpr) 0.24998303700105037 for threshold 0.859
```

Out[185]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2dd79cf97f0>
```



2.4.2.1 Graphviz visualization of Decision Tree on TFIDF, SET 2

In [0]:

```
# Please write all the code with proper documentation
```

In [259]:

```
from sklearn import tree
clf_tfidf = tree.DecisionTreeClassifier(max_depth=2,min_samples_split=2000)
clf_tfidf = clf_tfidf.fit(new_tfidf_data_matrix_train,new_y_train_tfidf)
```

In [257]:

```
final=final_tfidf[:57962]
```

In [258]:

```
import graphviz
dot_data = tree.export_graphviz(clf_tfidf, out_file=None, max_depth=2,feature_names=final)
tfidf_graph = graphviz.Source(dot_data)
tfidf_graph.render("tfidf_tree")
```

Out[258]:

```
'tfidf_tree.pdf'
```

In [260]:

```
tfidf_graph
```

Out[260]:

2.4.3 Applying Decision Trees on AVG W2V, SET 3

In [0]:

```
# Please write all the code with proper documentation
```

In [186]:

```
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 [187]:

```
# 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
[00:19<00:00, 1377.12it/s]
```

```
26219
300
```

In [188]:

```
# 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%|
██████████████████████████████████████████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████████████████████████████| 61179/61179
[00:35<00:00, 1726.87it/s]
```

```
61179
300
```

In [189]:

```
# 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%|
██████████████████████████████████████████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████████████████████████████| 21850/21850
[00:12<00:00, 1794.63it/s]
```

```
21850
300
```

In [190]:

```
# 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
    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))
print(len(avg_w2v_vectors_project_title_train[0]))
```

```
100%|
██████████████████████████████████████████████████████████████████████████████████
█████████████████████████████████████████████████████████████████████| 26219/26219
[00:01<00:00, 24326.60it/s]
```

```
26219
300
```

In [191]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_project_title_cv = []; # the avg-w2v for each sentence/review is stored in this li
st
for sentence in tqdm(X_cv_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_cv.append(vector)

print(len(avg_w2v_vectors_project_title_cv))
print(len(avg_w2v_vectors_project_title_cv[0]))
```

```
100%|
██████████████████████████████████████████████████████████████████████████████████
█████████████████████████████████████████████████████████████████████| 61179/61179
[00:02<00:00, 27657.64it/s]
```

```
61179
300
```

In [192]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_project_title_test = []; # the avg-w2v for each sentence/review is stored in this
list
for sentence in tqdm(X_test_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_test.append(vector)

print(len(avg_w2v_vectors_project_title_test))
print(len(avg_w2v_vectors_project_title_test[0]))
```

```
100%|
██████████████████████████████████████████████████████████████████████████████████
█████████████████████████████████████████████████████████████████████| 21850/21850
[00:01<00:00, 19653.03it/s]
```

```
21850
```

```
21850
300
```

In [193]:

```
# 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_avg['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_avg = price_scalar.transform(X_train_avg['price'].values.reshape(-1, 1))
```

```
Mean : 294.989221938289, Standard deviation : 344.44986393419094
```

In [194]:

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

In [195]:

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

In [196]:

```
#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 [197]:

```
#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 [198]:

```
#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 [199]:

```
#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 [200]:

```
#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)
```

In [201]:

```
#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 [202]:

```
#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 [203]:

```
#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 [204]:

```
#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 [205]:

```
#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 [206]:

```
# 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)
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 [207]:

```
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 [208]:

```
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 encodig ",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 encodig  (21850, 9)
```

In [209]:

```
# 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 encodig ",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 encodig  (26219, 30)
```

In [210]:

```
# 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 encodig ",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 encodig  (61179, 30)
```

In [211]:

```
# 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_test_avg = vectorizer.transform(X_test_avg['clean_subcategories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",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 encodig  (21850, 30)
```

In [212]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
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[212]:

```
(26219, 700)
```

In [213]:

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

Out[213]:

```
(61179, 700)
```

In [214]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
avgw2v_data_matrix_test=
hstack((one_hot_encoding_school_state_test_avg,one_hot_encoding_teacher_prefix_test_avg,one_hot_enc
oding_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[214]:

```
(21850, 700)
```

In [216]:

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

In [217]:

```
new_avgw2v_data_matrix_train=n1[:10001]
```

In [218]:

```
new_y_train_avgw2v=y_train_avg[:10001]
```

In [219]:

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

In [220]:

```
new_avgw2v_data_matrix_cv=n5[:10001]
```

In [221]:

```
new_y_cv_avgw2v=y_cv_avg[:10001]
```

In [222]:

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

In [223]:

```
new_avgw2v_data_matrix_test=n3[:10001]
```

In [227]:

```
new_y_test_avgw2v=y_test_avg[:10001]
```

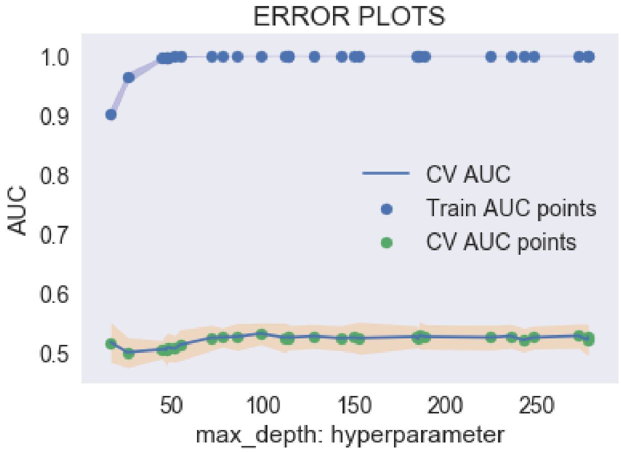
In [224]:

```
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_avgw2v = []
    tr_loop_avgw2v = 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 unti the last 1000 multiplier
    for i in range(0, tr_loop_avgw2v, 1000):
        y_data_pred_avgw2v.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    y_data_pred_avgw2v.extend(clf.predict_proba(data[tr_loop_avgw2v:])[:,1])

    return y_data_pred_avgw2v
```

In [225]:

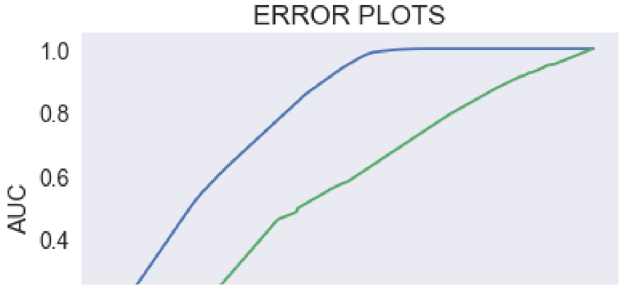
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn import tree
g = sp_randint(2,300)
dt= tree.DecisionTreeClassifier()
param_grid = {'max_depth':sorted(g.rvs(30))}
clf = GridSearchCV(dt, 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']



# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(param_grid['max_depth'],train_auc_avgw2v -
train_auc_std_avgw2v,train_auc_avgw2v + train_auc_std_avgw2v,alpha=0.2,color='darkblue')

plt.plot(param_grid['max_depth'], cv_auc_avgw2v, label='CV AUC')
```

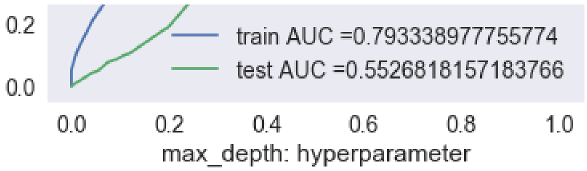
```
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(param_grid['max_depth'],cv_auc_avgw2v - cv_auc_std_avgw2v,cv_auc_avgw2v +
cv_auc_std_avgw2v,alpha=0.2,color='darkorange')

plt.scatter(param_grid['max_depth'], train_auc_avgw2v, label='Train AUC points')
plt.scatter(param_grid['max_depth'], cv_auc_avgw2v, label='CV AUC points')


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



In [230]:

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

dt= tree.DecisionTreeClassifier(max_depth=10)

dt.fit(new_avgw2v_data_matrix_train, new_y_train_avgw2v)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred_avgw2v = batch_predict(dt,new_avgw2v_data_matrix_train )
y_test_pred_avgw2v = batch_predict(dt, new_avgw2v_data_matrix_test)

train_fpr_avgw2v, train_tpr_avgw2v, tr_thresholds_avgw2v = roc_curve(new_y_train_avgw2v,
y_train_pred_avgw2v)
test_fpr_avgw2v, test_tpr_avgw2v, te_thresholds_avgw2v = roc_curve(new_y_test_avgw2v,
y_test_pred_avgw2v)

plt.plot(train_fpr_avgw2v, train_tpr_avgw2v, label="train AUC ="+str(auc(train_fpr_avgw2v, train_tp
r_avgw2v)))
plt.plot(test_fpr_avgw2v, test_tpr_avgw2v, label="test AUC ="+str(auc(test_fpr_avgw2v,
test_tpr_avgw2v)))
plt.legend()
plt.xlabel("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```





In [231]:

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

    t = threshould[np.argmax(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
```

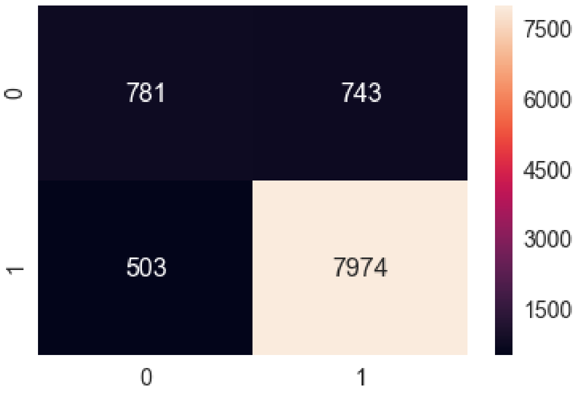
In [232]:

```
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) 0.24984456913358274 for threshold 0.833
```

Out[232]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2dd1ea2dd30>
```



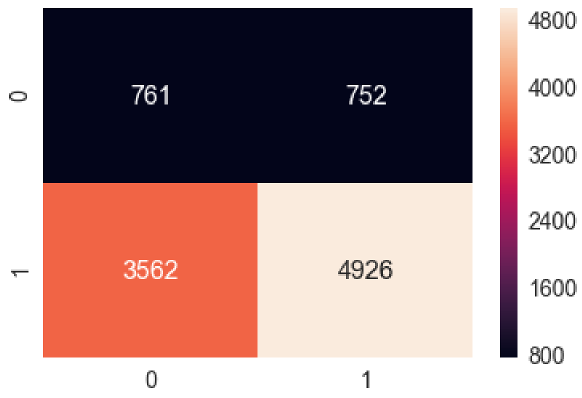
In [233]:

```
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) 0.2499911539951834 for threshold 0.889
```

Out[233]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2dd1fb2f518>
```



2.4.4 Applying Decision Trees on TFIDF W2V, SET 4

In [234]:

```
# Please write all the code with proper documentation
```

In [235]:

```
from sklearn.model_selection import train_test_split
X1_train, X_test_tfidf_w2v, y1_train, y_test_tfidf_w2v = train_test_split(
    project_data, y, test_size=0.20,stratify=y, random_state=42)
X_cv_tfidf_w2v,X_train_tfidf_w2v,y_cv_tfidf_w2v,y_train_tfidf_w2v=train_test_split(X1_train,y1_trai
n,test_size=0.20,stratify=y1_train,random_state=42)
```

In [236]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train_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 [237]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_project_title_train = []; # the avg-w2v for each sentence/review is stored in th
is list
for sentence in tqdm(X_train_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_train.append(vector)

print(len(tfidf_w2v_vectors_project_title_train))
print(len(tfidf_w2v_vectors_project_title_train[0]))
```

```
100%|
████████████████████████████████████████████████████████████████████████████████████
███████████████████████████████████████████████████████████████████████| 17480/17480
```

```
[00:01<00:00, 14672.80it/s]
```

```
17480
300
```

In [238]:

```
# 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 [239]:

```
# 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%|
████████████████████████████████████████████████████████████████████████████████████████████████
██████████████████████████████████████████████████████████████████████████████████| 69918/69918
[00:05<00:00, 13861.29it/s]
```

```
69918
300
```

In [240]:

```
# 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%|████████████████████████████████████████████████████████████████████████████████████████████████████| 69918/69918 [00:05<00:00, 13919.21it/s]
```

```
69918
300
```

In [241]:

```python
# 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 [242]:

```python
# 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 tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_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, 14902.85it/s]
```

```
21850
300
```

In [243]:

```python
# 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 [244]:

```python
# 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%|
████████████████████████████████████████████████████████████████████████████████████████████████████
███████████████████████████████████████████████████████████████████████████████████████|
17480/17480 [01:06<00:00, 263.94it/s]
```

```
17480
300
```

In [245]:

```
# 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 [246]:

```
# 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%|
████████████████████████████████████████████████████████████████████████████████████████████████████
███████████████████████████████████████████████████████████████████████████████████████|
69918/69918 [04:34<00:00, 255.05it/s]
```

```
69918
300
```

In [247]:

```
# 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 [248]:

In [248]:

```
# 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 [01:30<00:00, 240.12it/s]
```

```
21850
300
```

In [249]:

```
# 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_tfidf_w2v['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_tfidf_w2v =
price_scalar.transform(X_train_tfidf_w2v['price'].values.reshape(-1, 1))
```

```
Mean : 294.9769559496567, Standard deviation : 364.08638196754595
```

In [250]:

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

In [251]:

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

In [252]:

```
#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 (17480, 51)
```

In [253]:

```
#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 (69918, 51)
```

In [254]:

```
#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 [255]:

```
#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.sha
pe)
```

```
Shape of dataframe for teacher_prefix (17480, 5)
```

In [256]:

```
#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 (69918, 5)
```

In [257]:

```
#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.shap
e)
```

```
Shape of dataframe for teacher_prefix (21850, 5)
```

In [258]:

```
#onehotencoding for project_grade_category
one_hot_encoding_project_grade_category_train_tfidf_w2v=pd.get_dummies(X_train_tfidf_w2v.project_gr
ade_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 (17480, 4)
```

In [259]:

```
#onehotencoding for project_grade_category
one_hot_encoding_project_grade_category_train_tfidf_w2v=pd.get_dummies(X_train_tfidf_w2v.project_gr
ade_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 (17480, 4)
```

In [260]:

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

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 (69918, 4)
```

In [261]:

```
#onehotencoding for project_grade_category
one_hot_encoding_project_grade_category_test_tfidf_w2v=pd.get_dummies(X_test_tfidf_w2v.project_grac
e_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 [262]:

```
# 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 encodig ",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 encodig  (17480, 9)
```

In [263]:

```
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  (69918, 9)
```

In [264]:

```
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 [265]:

```
# 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  (17480, 30)
```

In [266]:

```
# 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  (17480, 30)
```

In [267]:

```
# 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'].va
lues)
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  (69918, 30)
```

In [268]:

```
# 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 [270]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
```

```
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_o
ne_hot_cv_tfidf_w2v,price_standardized_cv_tfidf_w2v,tfidf_w2v_vectors_preprocessed_essays_cv,tfidf
_w2v_vectors_project_title_cv))
```

In [271]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
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_categ
ories_one_hot_test_tfidf_w2v,price_standardized_test_tfidf_w2v,tfidf_w2v_vectors_preprocessed_essay
_test,tfidf_w2v_vectors_project_title_test))
```

In [273]:

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

In [274]:

```
new_tfidf_w2v_data_matrix_train=k1[:10001]
```

In [275]:

```
new_y_train_tfidf_w2v=y_train_tfidf_w2v[:10001]
```

In [276]:

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

In [277]:

```
new_tfidf_w2v_data_matrix_cv=k5[:10001]
```

In [278]:

```
new_y_cv_tfidf_w2v=y_cv_tfidf_w2v[:10001]
```

In [279]:

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

In [280]:

```
new_tfidf_w2v_data_matrix_test=k3[:10001]
```

In [281]:

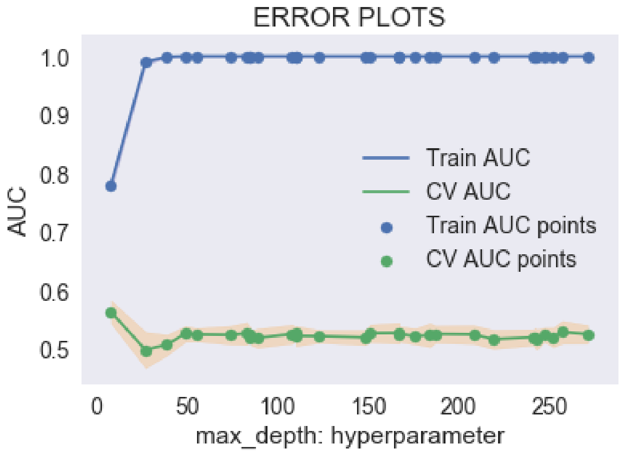
```
new_y_test_tfidf_w2v=y_test_tfidf_w2v[:10001]
```

In [282]:

```
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_w2v = []
    tr_loop_tfidf_w2v = 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 unti the last 1000 multiplier
    for i in range(0, tr_loop_tfidf_w2v, 1000):
        y_data_pred_tfidf_w2v.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    y_data_pred_tfidf_w2v.extend(clf.predict_proba(data[tr_loop_tfidf_w2v:])[:,1])

    return y_data_pred_tfidf_w2v
```

In [283]:

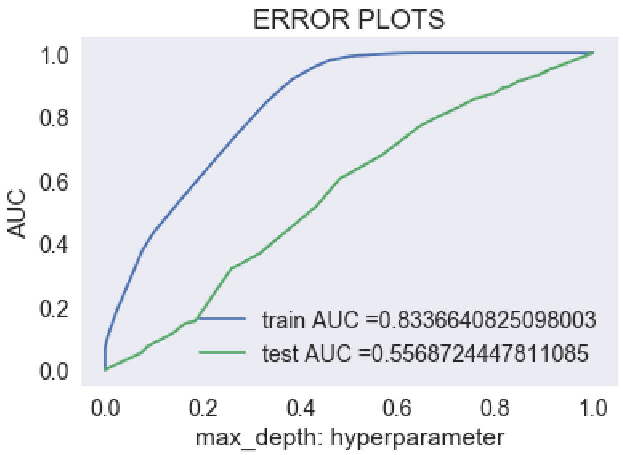
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn import tree
g = sp_randint(2,300)
dt= tree.DecisionTreeClassifier()
param_grid = {'max_depth':sorted(g.rvs(30))}

clf = GridSearchCV(dt, 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']

plt.plot(param_grid['max_depth'], train_auc_tfidf_w2v, label='Train AUC')


# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(param_grid['max_depth'],train_auc_tfidf_w2v - train_auc_std_tfidf_w2v,train_auc_tfidf_w2v + train_auc_std_tfidf_w2v,alpha=0.2,color='darkblue')

plt.plot(param_grid['max_depth'], cv_auc_tfidf_w2v, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(param_grid['max_depth'],cv_auc_tfidf_w2v - cv_auc_std_tfidf_w2v,cv_auc_tfidf_w2v+ cv_auc_std_tfidf_w2v,alpha=0.2,color='darkorange')

plt.scatter(param_grid['max_depth'], train_auc_tfidf_w2v, label='Train AUC points')
plt.scatter(param_grid['max_depth'], cv_auc_tfidf_w2v, label='CV AUC points')


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



In [286]:

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

```
dt= tree.DecisionTreeClassifier(max_depth=10)


dt.fit(new_tfidf_w2v_data_matrix_train, new_y_train_tfidf_w2v)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred_tfidf_w2v = batch_predict(dt,new_tfidf_w2v_data_matrix_train )
y_test_pred_tfidf_w2v = batch_predict(dt, 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="test AUC ="+str(auc(test_fpr_tfidf_w2v, tes
t_tpr_tfidf_w2v)))
plt.legend()
plt.xlabel("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [287]:

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

    t = threshould[np.argmax(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
```

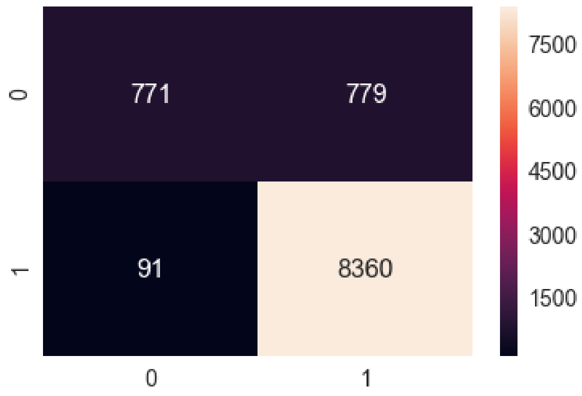
In [288]:

```
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) 0.2499933402705515 for threshold 0.617
```

Out[288]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2dd20d64f60>
```



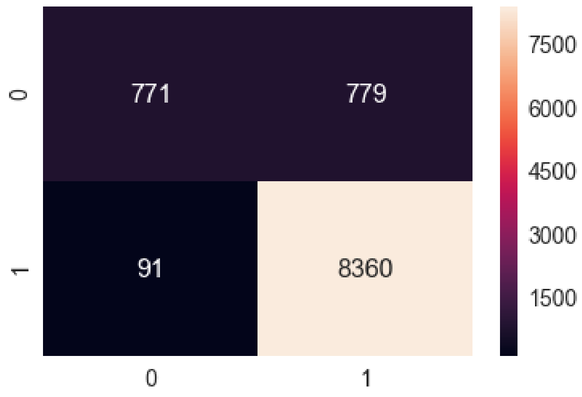
In [289]:

```python
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) 0.2499933402705515 for threshold 0.617
```

Out[289]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2dd20daeb38>
```



Top 25 features

In [221]:

```python
from sklearn import tree
clf = tree.DecisionTreeClassifier(max_depth=10,min_samples_split=100)
clf = clf.fit(new_bow_data_matrix_train,new_y_train_bow)
```

In [277]:

```
terms1=vectorizer3.get_feature_names()
```

In [278]:

```
terms3=vectorizer1.get_feature_names()
```

In [291]:

```
terms2=vectorizer.get_feature_names()
```

In [293]:

```
final_list=terms1+terms3+terms2
```

In [294]:

```
prob=clf.feature_importances_
```

In [298]:

```
sorted_array=np.argsort(-1*prob)
new_sorted_array=sorted_array[0:25]
```

In [299]:

```
new_bow_data_matrix_train
```

Out[299]:

```
<30001x57962 sparse matrix of type '<class 'numpy.float64'>'
 with 3557840 stored elements in Compressed Sparse Row format>
```

In [300]:

```
for i in new_sorted_array:

    print(final_list[i])
```

```
2d
distributive
montezuma
entitlement
iblog
notailored
kgia
philanthropist
popplets
insurance
awakens
mesopotamians
haulted
presently
bluest
rockers
tolkien
lands
couch
sec
carnivorous
prorioty
170
```

3. Conclusion

In [0]:

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

In [195]:

```
# 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","tpr*(1-fpr) for train","tpr*(1-fpr) for t
est"  ,]
x.add_row(["BOW",0.800,0.590,0.2499,0.2499 ])
x.add_row(["TFIDF",0.811,0.597,0.2472,0.2499])
x.add_row(["AVG_W2V",0.793,0.552,0.249,0.249])
x.add_row(["TFIDF_W2v",0.833,0.556,0.249,0.249])

print(x)
```

```
+---------------+-----------+----------+-----------------------+----------------------+
| Featurization | train_auc | test_auc | tpr*(1-fpr) for train | tpr*(1-fpr) for test |
+---------------+-----------+----------+-----------------------+----------------------+
|      BOW      |    0.8    |   0.59   |         0.2499        |        0.2499        |
|     TFIDF     |   0.811   |  0.597   |         0.2472        |        0.2499        |
|    AVG_W2V    |   0.793   |  0.552   |         0.249         |        0.249         |
|   TFIDF_W2v   |   0.833   |  0.556   |         0.249         |        0.249         |
+---------------+-----------+----------+-----------------------+----------------------+
```