DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

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

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

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

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

Feature	Description
`id`	A `project_id` value from the `train.csv` file. **Example:** `p036502`
`description`	Desciption of the resource. **Example:** `Tenor Saxophone Reeds, Box of 25`

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [1]:

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

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

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

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

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

from tqdm import tqdm
import os

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

1.1 Reading Data

In [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' 's
chool_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [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/473019

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

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [6]:

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

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

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

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

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

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

PREPROCESSING OF PROJECT GRADE CATEGORY

In [7]:

```
grade_categories=list(project_data['project_grade_category'].values)
clean_grades=[]
for i in grade_categories:
    temp=""
    for j in i.split(','):
        j=j.replace(' ','_')
        j=j.replace('-','_')
        temp+=j
        clean_grades.append(temp)
project_data['clean_grades']=clean_grades
project_data.drop(['project_grade_category'],axis=1,inplace=True)
```

1.3 Text preprocessing

In [8]:

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

In [9]:

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

Out[9]:

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

In [10]:

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

In [11]:

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

```
My students are English learners that are working on English as their second
or third languages. We are a melting pot of refugees, immigrants, and native
-born Americans bringing the gift of language to our school. \r\n\r\n We hav
e over 24 languages represented in our English Learner program with students
at every level of mastery.  We also have over 40 countries represented with
the families within our school.  Each student brings a wealth of knowledge a
nd experiences to us that open our eyes to new cultures, beliefs, and respec
t.\"The limits of your language are the limits of your world.\"-Ludwig Wittg
enstein  Our English learner's have a strong support system at home that beg
s for more resources.  Many times our parents are learning to read and speak
English along side of their children.  Sometimes this creates barriers for p
arents to be able to help their child learn phonetics, letter recognition, a
nd other reading skills.\r\n\r\nBy providing these dvd's and players, studen
ts are able to continue their mastery of the English language even if no one
at home is able to assist.  All families with students within the Level 1 pr
oficiency status, will be a offered to be a part of this program.  These edu
cational videos will be specially chosen by the English Learner Teacher and
will be sent home regularly to watch.  The videos are to help the child deve
lop early reading skills.\r\n\r\nParents that do not have access to a dvd pl
ayer will have the opportunity to check out a dvd player to use for the yea
r.  The plan is to use these videos and educational dvd's for the years to c
ome for other EL students.\r\nnannan
==================================================
The 51 fifth grade students that will cycle through my classroom this year a
ll love learning, at least most of the time. At our school, 97.3% of the stu
dents receive free or reduced price lunch. Of the 560 students, 97.3% are mi
nority students. \r\nThe school has a vibrant community that loves to get to
gether and celebrate. Around Halloween there is a whole school parade to sho
w off the beautiful costumes that students wear. On Cinco de Mayo we put on
a big festival with crafts made by the students, dances, and games. At the e
nd of the year the school hosts a carnival to celebrate the hard work put in
during the school year, with a dunk tank being the most popular activity.My
students will use these five brightly colored Hokki stools in place of regul
ar, stationary, 4-legged chairs. As I will only have a total of ten in the c
lassroom and not enough for each student to have an individual one, they wil
l be used in a variety of ways. During independent reading time they will be
used as special chairs students will each use on occasion. I will utilize th
em in place of chairs at my small group tables during math and reading time
s. The rest of the day they will be used by the students who need the highes
t amount of movement in their life in order to stay focused on school.\r\n\r
\nWhenever asked what the classroom is missing, my students always say more
Hokki Stools. They can't get their fill of the 5 stools we already have. Whe
n the students are sitting in group with me on the Hokki Stools, they are al
ways moving, but at the same time doing their work. Anytime the students get
to pick where they can sit, the Hokki Stools are the first to be taken. Ther
e are always students who head over to the kidney table to get one of the st
ools who are disappointed as there are not enough of them. \r\n\r\nWe ask a
```

```
lot of students to sit for 7 hours a day. The Hokki stools will be a comprom
ise that allow my students to do desk work and move at the same time. These
stools will help students to meet their 60 minutes a day of movement by allo
wing them to activate their core muscles for balance while they sit. For man
y of my students, these chairs will take away the barrier that exists in sch
ools for a child who can't sit still.nannan
==================================================
How do you remember your days of school? Was it in a sterile environment wit
h plain walls, rows of desks, and a teacher in front of the room? A typical
day in our room is nothing like that. I work hard to create a warm inviting
themed room for my students look forward to coming to each day.\r\n\r\nMy cl
ass is made up of 28 wonderfully unique boys and girls of mixed races in Ark
ansas.\r\nThey attend a Title I school, which means there is a high enough p
ercentage of free and reduced-price lunch to qualify. Our school is an \"ope
n classroom\" concept, which is very unique as there are no walls separating
the classrooms. These 9 and 10 year-old students are very eager learners; th
ey are like sponges, absorbing all the information and experiences and keep
on wanting more.With these resources such as the comfy red throw pillows and
the whimsical nautical hanging decor and the blue fish nets, I will be able
to help create the mood in our classroom setting to be one of a themed nauti
cal environment. Creating a classroom environment is very important in the s
uccess in each and every child's education. The nautical photo props will be
used with each child as they step foot into our classroom for the first time
on Meet the Teacher evening. I'll take pictures of each child with them, hav
e them developed, and then hung in our classroom ready for their first day o
f 4th grade.  This kind gesture will set the tone before even the first day
of school! The nautical thank you cards will be used throughout the year by
the students as they create thank you cards to their team groups.\r\n\r\nYou
r generous donations will help me to help make our classroom a fun, invitin
g, learning environment from day one.\r\n\r\nIt costs lost of money out of m
y own pocket on resources to get our classroom ready. Please consider helpin
g with this project to make our new school year a very successful one. Thank
you!nannan
==================================================
My kindergarten students have varied disabilities ranging from speech and la
nguage delays, cognitive delays, gross/fine motor delays, to autism. They ar
e eager beavers and always strive to work their hardest working past their l
imitations. \r\n\r\nThe materials we have are the ones I seek out for my stu
dents. I teach in a Title I school where most of the students receive free o
r reduced price lunch.  Despite their disabilities and limitations, my stude
nts love coming to school and come eager to learn and explore.Have you ever
felt like you had ants in your pants and you needed to groove and move as yo
u were in a meeting? This is how my kids feel all the time. The want to be a
ble to move as they learn or so they say.Wobble chairs are the answer and I
love then because they develop their core, which enhances gross motor and in
Turn fine motor skills. \r\nThey also want to learn through games, my kids d
on't want to sit and do worksheets. They want to learn to count by jumping a
nd playing. Physical engagement is the key to our success. The number toss a
nd color and shape mats can make that happen. My students will forget they a
re doing work and just have the fun a 6 year old deserves.nannan
==================================================
The mediocre teacher tells. The good teacher explains. The superior teacher
demonstrates. The great teacher inspires. -William A. Ward\r\n\r\nMy school
has 803 students which is makeup is 97.6% African-American, making up the la
rgest segment of the student body. A typical school in Dallas is made up of
23.2% African-American students. Most of the students are on free or reduced
lunch. We aren't receiving doctors, lawyers, or engineers children from rich
backgrounds or neighborhoods. As an educator I am inspiring minds of young c
hildren and we focus not only on academics but one smart, effective, efficie
nt, and disciplined students with good character.In our classroom we can uti
lize the Bluetooth for swift transitions during class. I use a speaker which
```

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

In [12]:

```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 [13]:

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

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

In [14]:

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

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

In [15]:

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

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

In [16]:

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

In [17]:

```
# 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)
    sent =sent.lower()
    # 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 [01:38<00:00, 1114.96it/s]
```

In [18]:

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

Out[18]:

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

1.4 Preprocessing of `project_title`

In [19]:

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

```
100%|████████████████████████████████████████████████████████████
| 109248/109248 [00:04<00:00, 24397.92it/s]
```

In [20]:

```
preprocessed_titles[0:5]
```

Out[20]:

```
['educational support english learners home',
 'wanted projector hungry learners',
 'soccer equipment awesome middle school students',
 'techie kindergarteners',
 'interactive math tools']
```

1.5 Preparing data for models

In [21]:

```
project_data.columns
```

Out[21]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', '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_approve
d',
       'clean_categories', 'clean_subcategories', 'clean_grades', 'essay'],
      dtype='object')
```

we are going to consider

```
        - school_state : categorical data
        - clean_categories : categorical data
        - clean_subcategories : categorical data
        - project_grade_category : categorical data
        - teacher_prefix : categorical data

        - project_title : text data
        - text : text data
        - project_resource_summary: text data (optinal)

        - quantity : numerical (optinal)
        - teacher_number_of_previously_posted_projects : numerical
        - price : numerical
```

1.5.1 Vectorizing Categorical data

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

In [22]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
categories_one_hot = vectorizer.fit_transform(project_data['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encoding ",categories_one_hot.shape)
```

```
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'liter
acy_language', 'math_science', 'music_arts', 'specialneeds', 'warmth']
Shape of matrix after one hot encoding  (109248, 9)
```

In [23]:

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

```
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_governmen
t', 'college_careerprep', 'communityservice', 'earlydevelopment', 'economic
s', 'environmentalscience', 'esl', 'extracurricular', 'financialliteracy',
'foreignlanguages', 'gym_fitness', 'health_lifescience', 'health_wellness',
'history_geography', 'literacy', 'literature_writing', 'mathematics', 'musi
c', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 's
ocialsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
Shape of matrix after one hot encodig  (109248, 30)
```

In [24]:

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

In [25]:

```
#vectorizing schoool state
vectorizer=CountVectorizer()
school_state_one_hot=vectorizer.fit_transform(project_data['school_state'].values)
print(vectorizer.get_feature_names())
print("shape of matrix after on ehot encoding",school_state_one_hot.shape)
```

```
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'i
a', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo',
'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'o
r', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv',
'wy']
shape of matrix after on ehot encoding (109248, 51)
```

In [26]:

```
#vectorizing project grade category
vectorizer=CountVectorizer()
project_grade_one_hot=vectorizer.fit_transform(project_data['clean_grades'].values)
print(vectorizer.get_feature_names())
print("shape of matrix after one hot encoding",project_grade_one_hot.shape)
```

```
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
shape of matrix after one hot encoding (109248, 4)
```

In [27]:

```
#vectorizing teacher prefix
x=project_data['teacher_prefix'].fillna('')
vectorizer = CountVectorizer()

teacher_prefix_one_hot = vectorizer.fit_transform(x.values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",teacher_prefix_one_hot.shape)
```

```
['dr', 'mr', 'mrs', 'ms', 'teacher']
Shape of matrix after one hot encodig  (109248, 5)
```

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 project
vectorizer = CountVectorizer(min_df=10)
text_bow = vectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",text_bow.shape)
```

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

In [29]:

```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
vectorizer=CountVectorizer(min_df=10)
title_bow=vectorizer.fit_transform(preprocessed_titles)
print("shape of matrix after one hot encoding",title_bow.shape)
```

```
shape of matrix after one hot encoding (109248, 3222)
```

1.5.2.2 TFIDF vectorizer

In [30]:

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

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

In [31]:

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

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

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 preprocessed_essays:
    words.extend(i.split(' '))

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

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

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


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

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

```
all the words in the coupus 15495364
the unique words in the coupus 58829
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-32-9d177f8ebb8a> in <module>()
     34 print("the unique words in the coupus", len(words))
     35 
---> 36 inter_words = set(model.keys()).intersection(words)
     37 print("The number of words that are present in both glove vectors an
d our coupus",      len(inter_words),"(",np.round(len(inter_words)/len(word
s)*100,3),"%)")
     38 

NameError: name 'model' is not defined
```

In [33]:

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

In [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 [00:37<00:00, 2933.45it/s]

109248
300
```

In [35]:

```
title_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles): # 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
    title_avg_w2v_vectors.append(vector)

print(len(title_avg_w2v_vectors))
print(len(title_avg_w2v_vectors[0]))
```

```
100%|████████████████████████████████████████████████
| 109248/109248 [00:02<00:00, 45500.77it/s]

109248
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [36]:

```
# 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 [37]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettir
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors.append(vector)

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

```
100%|████████████████████████████████████████████████████████████████████
█| 109248/109248 [04:18<00:00, 422.80it/s]

109248
300
```

In [38]:

```
# Similarly you can vectorize for title also
title_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettir
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    title_tfidf_w2v_vectors.append(vector)

print(len(title_tfidf_w2v_vectors))
print(len(title_tfidf_w2v_vectors[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
| 109248/109248 [00:05<00:00, 20948.94it/s]

109248
300
```

1.5.3 Vectorizing Numerical features

In [39]:

```
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 [40]:

```
project_data.columns
```

Out[40]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', '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_approve
d',
       'clean_categories', 'clean_subcategories', 'clean_grades', 'essay',
       'price', 'quantity'],
      dtype='object')
```

In [41]:

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

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

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

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

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

In [42]:

```
price_standardized
```

Out[42]:

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

In [43]:

```
projects_scalar = StandardScaler()
projects_scalar.fit(project_data['teacher_number_of_previously_posted_projects'].values.res
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])

# Now standardize the data with above maen and variance
projects_standardized = projects_scalar.transform(project_data['teacher_number_of_previousl
projects_standardized
```

```
C:\Users\HP\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\util
s\validation.py:475: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.


Mean : 298.1193425966608, Standard deviation : 367.49634838483496

C:\Users\HP\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\util
s\validation.py:475: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

Out[43]:

```
array([[-0.40152481],
       [-0.14951799],
       [-0.36552384],
       ...,
       [-0.29352189],
       [-0.40152481],
       [-0.40152481]])
```

In [44]:

```
projects_scalar = StandardScaler()
projects_scalar.fit(project_data['quantity'].values.reshape(-1,1)) # finding the mean and s
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])

# Now standardize the data with above maen and variance
quantity_standardized = projects_scalar.transform(project_data['quantity'].values.reshape(-
quantity_standardized
```

```
C:\Users\HP\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\util
s\validation.py:475: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.


Mean : 298.1193425966608, Standard deviation : 367.49634838483496

C:\Users\HP\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\util
s\validation.py:475: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

Out[44]:

```
array([[ 0.23047132],
       [-0.60977424],
       [ 0.19227834],
       ...,
       [-0.4951953 ],
       [-0.03687954],
       [-0.45700232]])
```

In [45]:

```
quantity_standardized.shape
```

Out[45]:

```
(109248, 1)
```

1.5.4 Merging all the above features

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

In [46]:

```
print(categories_one_hot.shape)
print(sub_categories_one_hot.shape)
print(teacher_prefix_one_hot.shape)
print(school_state_one_hot.shape)
print(project_grade_one_hot.shape)
print(title_bow.shape)
print(text_bow.shape)
print(price_standardized.shape)
print(projects_standardized.shape)
```

```
(109248, 9)
(109248, 30)
(109248, 5)
(109248, 51)
(109248, 4)
(109248, 3222)
(109248, 16512)
(109248, 1)
(109248, 1)
```

In [47]:

```
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,teacher_prefix_one_hot,school_state_
X.shape
```

Out[47]:

```
(109248, 19835)
```

Computing Sentiment Scores

In [48]:

```
import nltk
nltk.downloader.download('vader_lexicon')
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 stu
for learning my students learn in many different ways using all of our senses and multiple
of techniques to help all my students succeed students in my class come from a variety of d
for wonderful sharing of experiences and cultures including native americans our school is
learners which can be seen through collaborative student project based learning in and out
in my class love to work with hands on materials and have many different opportunities to p
mastered having the social skills to work cooperatively with friends is a crucial aspect of
montana is the perfect place to learn about agriculture and nutrition my students love to r
in the early childhood classroom i have had several kids ask me can we try cooking with rea
and create common core cooking lessons where we learn important math and writing concepts w
food for snack time my students will have a grounded appreciation for the work that went in
of where the ingredients came from as well as how it is healthy for their bodies this proje
nutrition and agricultural cooking recipes by having us peel our own apples to make homemad
and mix up healthy plants from our classroom garden in the spring we will also create our c
shared with families students will gain math and literature skills as well as a life long e
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
```

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

C:\Users\HP\AppData\Local\Continuum\anaconda3\lib\site-packages\nltk\twitter
\__init__.py:20: UserWarning:

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


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

In [49]:

```
y1=project_data['project_is_approved']
print(y1.shape)
y=y1[0:100000]
```

```
(109248,)
```

In [50]:

```
data1=project_data.drop(['id','teacher_id','project_essay_1','project_essay_2','project_ess
data1.head(2)
data=data1[0:100000]
data[0:2]
```

Out[50]:

	Unnamed: 0	teacher_prefix	school_state	project_submitted_datetime	project_title	project_res
0	160221	Mrs.	IN	2016-12-05 13:43:57	Educational Support for English Learners at Home	I opportı
1	140945	Mr.	FL	2016-10-25 09:22:10	Wanted: Projector for Hungry Learners	My students tc

Assignment 5: Logistic Regression

1. **[Task-1] Logistic Regression(either SGDClassifier with log loss, or LogisticRegression) on these feature sets**

    - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (`BOW with bi-grams with min_df=10 and max_features=5000`)
    - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (`TFIDF with bi-grams with min_df=10 and max_features=5000`)
    - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
    - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)

2. **Hyper paramter tuning (find best hyper parameters corresponding the algorithm that you choose)**

    - Find the best hyper parameter which will give the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
    - Find the best hyper paramter using k-fold cross validation or simple cross validation data
    - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

3. **Representation of results**

    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.

- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
- Along with plotting ROC curve, you need to print the confusion matrix (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps. (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

4. **[Task-2] Apply Logistic Regression on the below feature set Set 5 by finding the best hyper parameter as suggested in step 2 and step 3.**
5. Consider these set of features Set 5 :
   - **school_state** : categorical data
   - **clean_categories** : categorical data
   - **clean_subcategories** : categorical data
   - **project_grade_category** :categorical data
   - **teacher_prefix** : categorical data
   - **quantity** : numerical data
   - **teacher_number_of_previously_posted_projects** : numerical data
   - **price** : numerical data
   - **sentiment score's of each of the essay** : numerical data
   - **number of words in the title** : numerical data
   - **number of words in the combine essays** : numerical data

   And apply the Logistic regression on these features by finding the best hyper paramter as suggested in step 2 and step 3
6. **Conclusion**
   - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link (http://zetcode.com/python/prettytable/)

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakage, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link. (https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf)

2. Logistic Regression

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

In [51]:

```
# please write all the code with proper documentation, and proper titles for each subsectic
# 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 coc
# 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

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(data,y,test_size=0.30,stratify=y)
X_train,X_cv,y_train,y_cv=train_test_split(X_train,y_train,test_size=0.30,stratify=y_train)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [52]:

```
# please write all the code with proper documentation, and proper titles for each subsectic
# 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 coc
# 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 [53]:

```
#encoding categorical variable school state
vectorizer=CountVectorizer()
vectorizer.fit(X_train['school_state'].values)

X_train_state_encoded=vectorizer.transform(X_train['school_state'].values)
X_cv_state_encoded=vectorizer.transform(X_cv['school_state'].values)
X_test_state_encoded=vectorizer.transform(X_test['school_state'].values)

print("AFTER VECTORIZATION")
print('='*50)
print(X_train_state_encoded.shape,y_train.shape)
print(X_cv_state_encoded.shape,y_cv.shape)
print(X_test_state_encoded.shape,y_test.shape)
```

```
AFTER VECTORIZATION
==================================================
(49000, 51) (49000,)
(21000, 51) (21000,)
(30000, 51) (30000,)
```

In [54]:

```
#vectorizing teacher prefix
X_train['teacher_prefix'].fillna('',inplace=True)
X_cv['teacher_prefix'].fillna('',inplace=True)
X_test['teacher_prefix'].fillna('',inplace=True)
```

In [55]:

```
vectorizer=CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values)
X_train_prefix_encoded=vectorizer.transform(X_train['teacher_prefix'].values)
X_cv_prefix_encoded=vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_prefix_encoded=vectorizer.transform(X_test['teacher_prefix'].values)

print('AFTER VECTORIZATION')
print('='*50)
print(vectorizer.get_feature_names())
print(X_train_prefix_encoded.shape,y_train.shape)
print(X_cv_prefix_encoded.shape,y_cv.shape)
print(X_test_prefix_encoded.shape,y_test.shape)
```

```
AFTER VECTORIZATION
==================================================
['dr', 'mr', 'mrs', 'ms', 'teacher']
(49000, 5) (49000,)
(21000, 5) (21000,)
(30000, 5) (30000,)
```

In [56]:

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

Out[56]:

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

In [57]:

```
#encoding project grade category
vectorizer=CountVectorizer()
vectorizer.fit(X_train['clean_grades'].values)
X_train_grade_encoded=vectorizer.transform(X_train['clean_grades'].values)
X_cv_grade_encoded=vectorizer.transform(X_cv['clean_grades'].values)
X_test_grade_encoded=vectorizer.transform(X_test['clean_grades'].values)

print('AFTER VECTORIZATION')
print('='*50)
print(X_train_grade_encoded.shape,y_train.shape)
print(X_cv_grade_encoded.shape,y_cv.shape)
print(X_test_grade_encoded.shape,y_test.shape)
```

```
AFTER VECTORIZATION
==================================================
(49000, 4) (49000,)
(21000, 4) (21000,)
(30000, 4) (30000,)
```

In [58]:

```
#encoding clean categories
vectorizer=CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values)
X_train_categories_encoded=vectorizer.transform(X_train['clean_categories'].values)
X_cv_categories_encoded=vectorizer.transform(X_cv['clean_categories'].values)
X_test_categories_encoded=vectorizer.transform(X_test['clean_categories'].values)
print("AFTER VECTORIZATION")
print('='*50)
print(X_train_categories_encoded.shape,y_train.shape)
print(X_cv_categories_encoded.shape,y_cv.shape)
print(X_test_categories_encoded.shape,y_test.shape)
```

```
AFTER VECTORIZATION
==================================================
(49000, 9) (49000,)
(21000, 9) (21000,)
(30000, 9) (30000,)
```

In [59]:

```
#encoding subcategories
vectorizer=CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values)
X_train_subcategories_encoded=vectorizer.transform(X_train['clean_subcategories'].values)
X_cv_subcategories_encoded=vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_subcategories_encoded=vectorizer.transform(X_test['clean_subcategories'].values)
print("AFTER VECTORIZATION")
print('='*50)
print(X_train_subcategories_encoded.shape,y_train.shape)
print(X_cv_subcategories_encoded.shape,y_cv.shape)
print(X_test_subcategories_encoded.shape,y_test.shape)
```

```
AFTER VECTORIZATION
==================================================
(49000, 30) (49000,)
(21000, 30) (21000,)
(30000, 30) (30000,)
```

In [60]:

```
#encoding numerical categories---price
from sklearn.preprocessing import Normalizer
normalizer=Normalizer()
normalizer.fit(X_train['price'].values.reshape(-1,1))

X_train_price_norm=normalizer.transform(X_train['price'].values.reshape(-1,1))
X_cv_price_norm=normalizer.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_norm=normalizer.transform(X_test['price'].values.reshape(-1,1))

print("after vectorization")
print(X_train_price_norm.shape,y_train.shape)
print(X_cv_price_norm.shape,y_cv.shape)
print(X_test_price_norm.shape,y_test.shape)
```

```
after vectorization
(49000, 1) (49000,)
(21000, 1) (21000,)
(30000, 1) (30000,)
```

In [61]:

```
#encoding previous projects posted by teachers
normalizer=Normalizer()
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)

X_train_projects_norm=normalizer.transform(X_train['teacher_number_of_previously_posted_pro
X_cv_projects_norm=normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'
X_test_projects_norm=normalizer.transform(X_test['teacher_number_of_previously_posted_proje

print("after vectorization")
print(X_train_projects_norm.shape,y_train.shape)
print(X_cv_projects_norm.shape,y_cv.shape)
print(X_test_projects_norm.shape,y_test.shape)
```

```
after vectorization
(49000, 1) (49000,)
(21000, 1) (21000,)
(30000, 1) (30000,)
```

In [62]:

```
#encoding numerical category quantity
normalizer=Normalizer()
normalizer.fit(X_train['quantity'].values.reshape(-1,1))

X_train_quantity_norm=normalizer.transform(X_train['quantity'].values.reshape(-1,1))
X_cv_quantity_norm=normalizer.transform(X_cv['quantity'].values.reshape(-1,1))
X_test_quantity_norm=normalizer.transform(X_test['quantity'].values.reshape(-1,1))

print('after vectorization')
print(X_train_quantity_norm.shape,y_train.shape)
print(X_cv_quantity_norm.shape,y_cv.shape)
print(X_test_quantity_norm.shape,y_test.shape)
```

```
after vectorization
(49000, 1) (49000,)
(21000, 1) (21000,)
(30000, 1) (30000,)
```

In [ ]:

In [ ]:

2.3 Make Data Model Ready: encoding eassay, and project_title

In [63]:

```
# please write all the code with proper documentation, and proper titles for each subsectic
# 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 coc
# 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 [64]:

```
#encoding project essay
vectorizer=CountVectorizer(min_df=10,ngram_range=(1,2),max_features=5000)
vectorizer.fit(X_train['essay'].values)
X_train_essay_bow=vectorizer.transform(X_train['essay'].values)
#print(X_train_essay_bow.shape)
X_cv_essay_bow=vectorizer.transform(X_cv["essay"].values)
X_test_essay_bow=vectorizer.transform(X_test['essay'].values)

print('AFTER VECTORIZATION')
print('='*50)
print(X_train_essay_bow.shape, y_train.shape)
print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
```

```
AFTER VECTORIZATION
==================================================
(49000, 5000) (49000,)
(21000, 5000) (21000,)
(30000, 5000) (30000,)
```

In [65]:

```
#encoding project title
vectorizer=CountVectorizer(min_df=10,ngram_range=(1,2),max_features=5000)
vectorizer.fit(X_train['project_title'].values)
X_train_title_bow=vectorizer.transform(X_train['project_title'].values)
X_cv_title_bow=vectorizer.transform(X_cv['project_title'].values)
X_test_title_bow=vectorizer.transform(X_test['project_title'].values)
print("after vectorization")
print(X_train_title_bow.shape,y_train.shape)
print(X_cv_title_bow.shape,y_cv.shape)
print(X_test_title_bow.shape,y_test.shape)
```

```
after vectorization
(49000, 4709) (49000,)
(21000, 4709) (21000,)
(30000, 4709) (30000,)
```

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

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

In [66]:

```
# please write all the code with proper documentation, and proper titles for each subsectic
# 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 coc
# 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 [67]:

```
#creating final data matrix
from scipy.sparse import hstack
final_train_bow=hstack((X_train_state_encoded,X_train_prefix_encoded,X_train_grade_encoded,
final_cv_bow=hstack((X_cv_state_encoded,X_cv_prefix_encoded,X_cv_grade_encoded,X_cv_categor
final_test_bow=hstack((X_test_state_encoded,X_test_prefix_encoded,X_test_grade_encoded,X_te
print(final_train_bow.shape,y_train.shape)
print(final_cv_bow.shape,y_cv.shape)
print(final_test_bow.shape,y_test.shape)
```

```
(49000, 9811) (49000,)
(21000, 9811) (21000,)
(30000, 9811) (30000,)
```

In [ ]:

In [68]:

```
#plotting error plots
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.svm.libsvm import predict_proba

train_auc=[]
cv_auc=[]
c=[0.00001,0.0001,0.001,0.01,0.1,1,10,100]
for i in tqdm(c):
    log_reg=LogisticRegression(C=np.log(i),class_weight='balanced')
    log_reg.fit(final_train_bow,y_train)

    y_tr_pred=log_reg.predict_proba(final_train_bow)[:,1]
    y_cv_pred=log_reg.predict_proba(final_cv_bow)[:,1]

    train_auc.append(roc_auc_score(y_train,y_tr_pred))
    cv_auc.append(roc_auc_score(y_cv,y_cv_pred))

plt.plot(c,train_auc,label='TRAIN AUC')
plt.plot(c,cv_auc,label="CV AUC")

plt.scatter(c,train_auc,label='TRAIN AUC POINTS')
plt.scatter(c,cv_auc,label="CV AUC POINTS")

plt.legend()
plt.title("ERROR PLOTS")
plt.xlabel("c- HYPERPARAMETER")
plt.ylabel("auc score")
plt.grid()
plt.show()
```

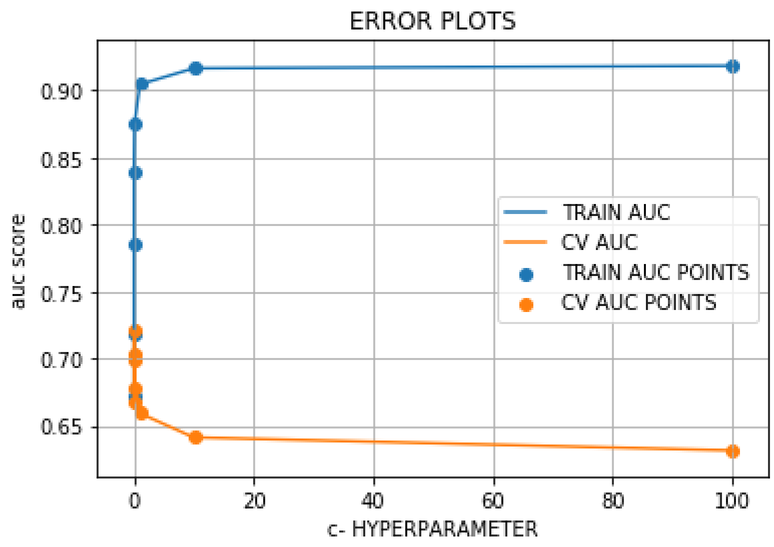
```
  0%|
| 0/8 [00:00<?, ?it/s]

---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call las
t)
<ipython-input-68-372e4a392be2> in <module>()
     10 for i in tqdm(c):
     11     log_reg=LogisticRegression(C=np.log(i),class_weight='balanced'
)
---> 12     log_reg.fit(final_train_bow,y_train)
     13 
     14     y_tr_pred=log_reg.predict_proba(final_train_bow)[:,1]

~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\linear_model
\logistic.py in fit(self, X, y, sample_weight)
   1200         if not isinstance(self.C, numbers.Number) or self.C < 0:
   1201             raise ValueError("Penalty term must be positive; got
 (C=%r)"
-> 1202                              % self.C)
   1203         if not isinstance(self.max_iter, numbers.Number) or self.m
ax_iter < 0:
   1204             raise ValueError("Maximum number of iteration must be
 positive;"
```

```
ValueError: Penalty term must be positive; got (C=-11.512925464970229)
```

In [69]:

```python
#plotting error plots
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.svm.libsvm import predict_proba

train_auc=[]
cv_auc=[]
c=[0.00001,0.0001,0.001,0.01,0.1,1,10,100]
for i in tqdm(c):
    log_reg=LogisticRegression(C=i,class_weight='balanced')
    log_reg.fit(final_train_bow,y_train)

    y_tr_pred=log_reg.predict_proba(final_train_bow)[:,1]
    y_cv_pred=log_reg.predict_proba(final_cv_bow)[:,1]

    train_auc.append(roc_auc_score(y_train,y_tr_pred))
    cv_auc.append(roc_auc_score(y_cv,y_cv_pred))

plt.plot(c,train_auc,label='TRAIN AUC')
plt.plot(c,cv_auc,label="CV AUC")

plt.scatter(c,train_auc,label='TRAIN AUC POINTS')
plt.scatter(c,cv_auc,label="CV AUC POINTS")

plt.legend()
plt.title("ERROR PLOTS")
plt.xlabel("c- HYPERPARAMETER")
plt.ylabel("auc score")
plt.grid()
plt.show()
```

```
100%|████████████████████████████████████████
█████████| 8/8 [11:58<00:00, 174.61s/it]
```



#OBSERVATIONS: Here we plotted for 10 different values of c ranging between 0.00001 to 100. And we choose 0.1 as our best c.

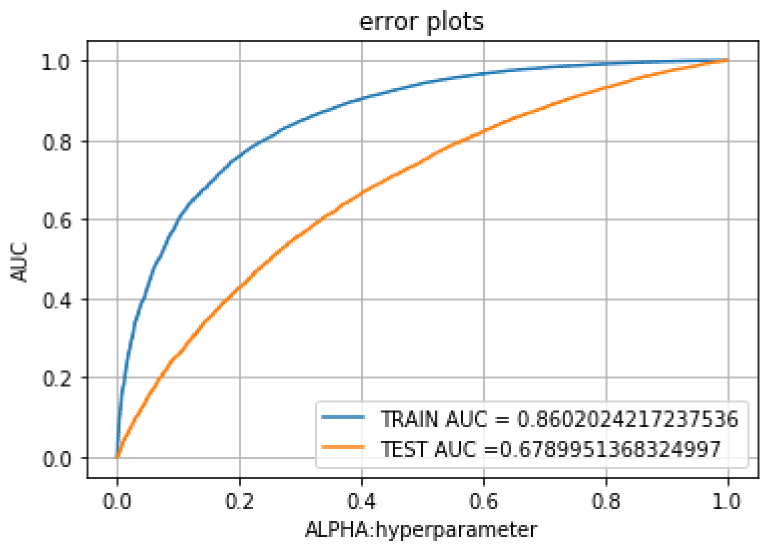
In [76]:

```
from sklearn.metrics import roc_curve,auc
from sklearn.svm.libsvm import predict_proba

log_reg=LogisticRegression(C=np.log(0.1))
log_reg.fit(final_train_bow,y_train)

y_train_pred=log_reg.predict_proba(final_train_bow)[:,1]
y_test_pred=log_reg.predict_proba(final_test_bow)[:,1]

train_fpr,train_tpr,tr_threshold=roc_curve(y_train,y_train_pred)
test_fpr,test_tpr,test_threshold=roc_curve(y_test,y_test_pred)

plt.plot(train_fpr,train_tpr,label="TRAIN AUC = "+str(auc(train_fpr,train_tpr)))
plt.plot(test_fpr,test_tpr,label="TEST AUC ="+str(auc(test_fpr,test_tpr)))

plt.legend()
plt.xlabel('ALPHA:hyperparameter')
plt.ylabel('AUC')
plt.title("error plots")
plt.grid()
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-76-5136eeeea06c> in <module>()
      3 
      4 log_reg=LogisticRegression(C=np.log(0.1))
----> 5 log_reg.fit(final_train_bow,y_train)
      6 
      7 y_train_pred=log_reg.predict_proba(final_train_bow)[:,1]

~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\linear_model\l
ogistic.py in fit(self, X, y, sample_weight)
   1200         if not isinstance(self.C, numbers.Number) or self.C < 0:
   1201             raise ValueError("Penalty term must be positive; got (C
=%r)"
-> 1202                              % self.C)
   1203         if not isinstance(self.max_iter, numbers.Number) or self.max
_iter < 0:
   1204             raise ValueError("Maximum number of iteration must be po
sitive;"

ValueError: Penalty term must be positive; got (C=-2.3025850929940455)
```

In [75]:

```
from sklearn.metrics import roc_curve,auc
from sklearn.svm.libsvm import predict_proba

log_reg=LogisticRegression(C=0.1)
log_reg.fit(final_train_bow,y_train)

y_train_pred=log_reg.predict_proba(final_train_bow)[:,1]
y_test_pred=log_reg.predict_proba(final_test_bow)[:,1]

train_fpr,train_tpr,tr_threshold=roc_curve(y_train,y_train_pred)
test_fpr,test_tpr,test_threshold=roc_curve(y_test,y_test_pred)

plt.plot(train_fpr,train_tpr,label="TRAIN AUC = "+str(auc(train_fpr,train_tpr)))
plt.plot(test_fpr,test_tpr,label="TEST AUC ="+str(auc(test_fpr,test_tpr)))

plt.legend()
plt.xlabel('ALPHA:hyperparameter')
plt.ylabel('AUC')
plt.title("error plots")
plt.grid()
plt.show()
```



OBSERVATIONS: For c=0.1 we got train auc as 85.98% which is a decent value. We got test auc score of 68.14%. The difference between train and test auc scores is very high.

In [77]:

```
def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t
    return t

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

In [78]:

```
#printing confusion matrix
print('='*100)
from sklearn.metrics import confusion_matrix
best_t=find_best_threshold(tr_threshold,train_fpr,train_tpr)
print('TRAIN CONFUSION MATRIX')
print(confusion_matrix(y_train,predict_with_best_t(y_train_pred,best_t)))
print('test confusion matrix')
print(confusion_matrix(y_test,predict_with_best_t(y_test_pred,best_t)))
```

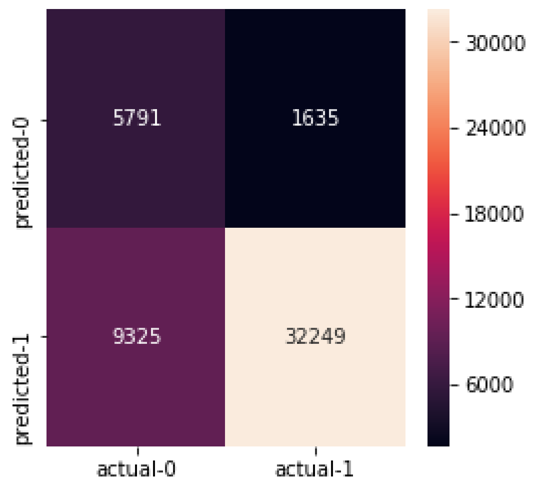
```
===========================================================================
========================
the maximum value of tpr*(1-fpr) 0.6102455178388375 for threshold 0.825
TRAIN CONFUSION MATRIX
[[ 5803  1623]
 [ 9108 32466]]
test confusion matrix
[[ 2345  2201]
 [ 6793 18661]]
```

In [79]:

```
#printing heatmap for train confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

array=[[5791,1635],[9325,32249]]

train=pd.DataFrame(array,index=['predicted-0','predicted-1'],columns=['actual-0','actual-1'
plt.figure(figsize=(4,4))
sn.heatmap(train,annot=True,fmt='d')
```

Out[79]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2558cb75438>
```



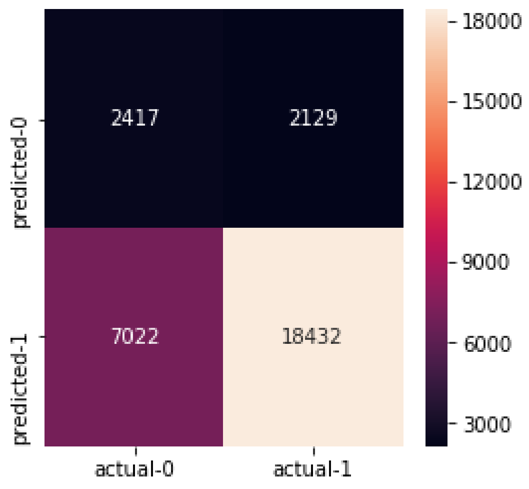
OBSERVATIONS: Here we plotted heatmap of training confusion matrix. Here tnr and tpr have a decent values.

In [80]:

```
#printing heatmap for test confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

testarray=[[2417,2129],[7022,18432]]

test=pd.DataFrame(testarray,index=['predicted-0','predicted-1'],columns=['actual-0','actual
plt.figure(figsize=(4,4))
sn.heatmap(test,annot=True,fmt='d')
```

Out[80]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2558c1b0ef0>
```



OBSERVATIONS: From the test confusion matrix above we have pretty good tpr value . But tnr value is low and fnr value is larger.

SET 2 APPLYING ON TFIDF

In [81]:

```
#tfidf encoding  of text
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,2),max_features=5000)
vectorizer.fit(X_train['essay'].values)
train_tfidf=vectorizer.transform(X_train['essay'].values)
cv_tfidf=vectorizer.transform(X_cv['essay'].values)
test_tfidf=vectorizer.transform(X_test['essay'].values)
print("Shape of matrix after one hot encodig ",train_tfidf.shape)
print("Shape of matrix after one hot encodig ",cv_tfidf.shape)
print("Shape of matrix after one hot encodig ",test_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (49000, 5000)
Shape of matrix after one hot encodig  (21000, 5000)
Shape of matrix after one hot encodig  (30000, 5000)
```

In [82]:

```
#tfidf encoding of title
vectorizer=TfidfVectorizer(min_df=10,ngram_range=(1,2),max_features=5000)
vectorizer.fit(X_train['project_title'].values)
title_train_tfidf=vectorizer.transform(X_train['project_title'].values)
title_cv_tfidf=vectorizer.transform(X_cv['project_title'].values)
title_test_tfidf=vectorizer.transform(X_test['project_title'].values)

print("Shape of matrix after one hot encodig ",title_train_tfidf.shape)
print("Shape of matrix after one hot encodig ",title_cv_tfidf.shape)
print("Shape of matrix after one hot encodig ",title_test_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (49000, 4709)
Shape of matrix after one hot encodig  (21000, 4709)
Shape of matrix after one hot encodig  (30000, 4709)
```

In [83]:

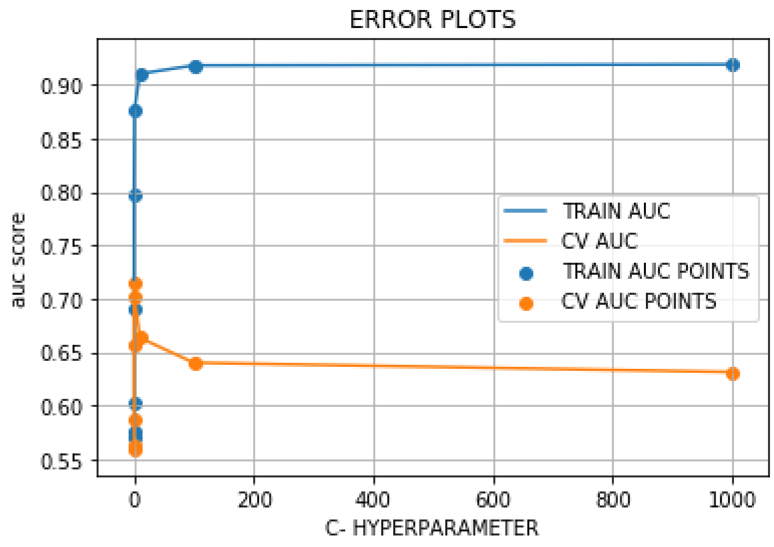
```
#creating final data matrix
from scipy.sparse import hstack
final_train_tfidf=hstack((X_train_state_encoded,X_train_prefix_encoded,X_train_grade_encode
final_cv_tfidf=hstack((X_cv_state_encoded,X_cv_prefix_encoded,X_cv_grade_encoded,X_cv_categ
final_test_tfidf=hstack((X_test_state_encoded,X_test_prefix_encoded,X_test_grade_encoded,X_
print(final_train_tfidf.shape,y_train.shape)
print(final_cv_tfidf.shape,y_cv.shape)
print(final_test_tfidf.shape,y_test.shape)
```

```
(49000, 9811) (49000,)
(21000, 9811) (21000,)
(30000, 9811) (30000,)
```

In [85]:

```
#plotting error plots
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.svm.libsvm import predict_proba

train_auc=[]
cv_auc=[]
c=[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000]
for i in tqdm(c):
    log_reg=LogisticRegression(C=i,class_weight='balanced')
    log_reg.fit(final_train_tfidf,y_train)

    y_tr_pred=log_reg.predict_proba(final_train_tfidf)[:,1]
    y_cv_pred=log_reg.predict_proba(final_cv_tfidf)[:,1]

    train_auc.append(roc_auc_score(y_train,y_tr_pred))
    cv_auc.append(roc_auc_score(y_cv,y_cv_pred))

plt.plot(c,train_auc,label='TRAIN AUC')
plt.plot(c,cv_auc,label="CV AUC")

plt.scatter(c,train_auc,label='TRAIN AUC POINTS')
plt.scatter(c,cv_auc,label="CV AUC POINTS")

plt.legend()
plt.title("ERROR PLOTS")
plt.xlabel("C- HYPERPARAMETER")
plt.ylabel("auc score")
plt.grid()
plt.show()
```

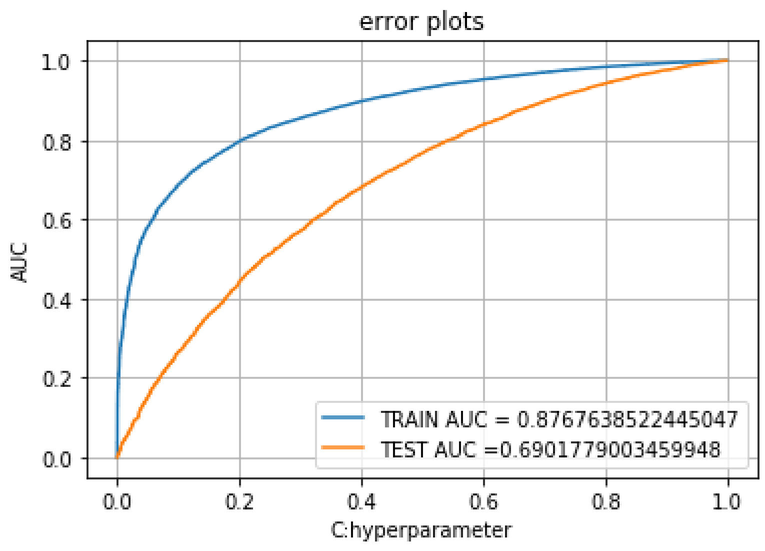
```
100%|████████████████████████████████████████████████
██████████| 9/9 [05:47<00:00, 83.67s/it]
```



OBSERVATIONS: Here we plotted train and cv auc for 9 different values of c ranging between 0.00001 to 1000. And we choose 1 as our best c.

In [87]:

```
from sklearn.metrics import roc_curve,auc
from sklearn.svm.libsvm import predict_proba
best_c=1

log_reg=LogisticRegression(C=best_c,class_weight='balanced')
log_reg.fit(final_train_tfidf,y_train)

y_train_pred=log_reg.predict_proba(final_train_tfidf)[:,1]
y_test_pred=log_reg.predict_proba(final_test_tfidf)[:,1]

train_fpr,train_tpr,tr_threshold=roc_curve(y_train,y_train_pred)
test_fpr,test_tpr,test_threshold=roc_curve(y_test,y_test_pred)

plt.plot(train_fpr,train_tpr,label="TRAIN AUC = "+str(auc(train_fpr,train_tpr)))
plt.plot(test_fpr,test_tpr,label="TEST AUC ="+str(auc(test_fpr,test_tpr)))

plt.legend()
plt.xlabel('C:hyperparameter')
plt.ylabel('AUC')
plt.title("error plots")
plt.grid()
plt.show()
```



OBSERVATIONS: For c=1 we got better train auc score of 87.5% which is good. We got test auc score of 69.1% but the gap between train and test scores is a bit large.

In [88]:

```
#printing confusion matrix
print('='*100)
from sklearn.metrics import confusion_matrix
best_t=find_best_threshold(tr_threshold,train_fpr,train_tpr)
print('TRAIN CONFUSION MATRIX')
print(confusion_matrix(y_train,predict_with_best_t(y_train_pred,best_t)))
print('test confusion matrix')
print(confusion_matrix(y_test,predict_with_best_t(y_test_pred,best_t)))
```

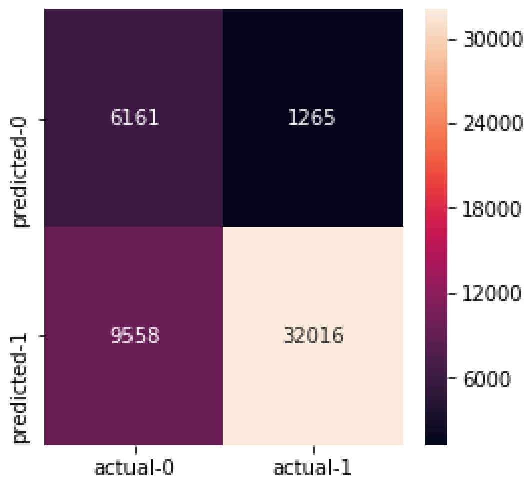
```
============================================================================
========================
the maximum value of tpr*(1-fpr) 0.639277542103625 for threshold 0.509
TRAIN CONFUSION MATRIX
[[ 6206  1220]
 [ 9772 31802]]
test confusion matrix
[[ 2541  2005]
 [ 7172 18282]]
```

In [89]:

```
#printing heatmap for train confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

array=[[6161,1265],[9558,32016]]

train=pd.DataFrame(array,index=['predicted-0','predicted-1'],columns=['actual-0','actual-1'
plt.figure(figsize=(4,4))
sn.heatmap(train,annot=True,fmt='d')
```

Out[89]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2558c00e0f0>
```



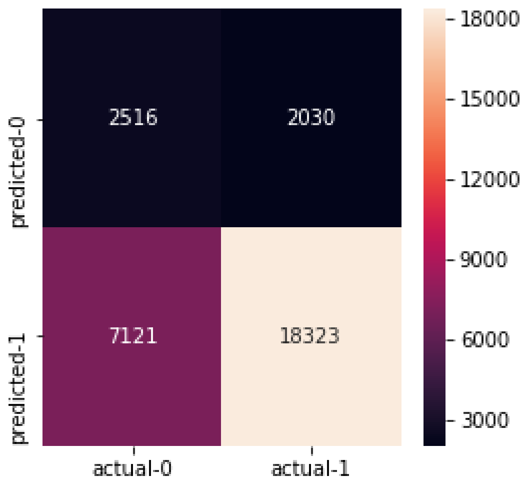
OBSERVATIONS: Here we got a pretty good tnr and tpr scores for training data.

In [90]:

```
#printing heatmap for test confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

testarray=[[2516,2030],[7121,18323]]

test=pd.DataFrame(testarray,index=['predicted-0','predicted-1'],columns=['actual-0','actual
plt.figure(figsize=(4,4))
sn.heatmap(test,annot=True,fmt='d')
```

Out[90]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2558bff1a20>
```



OBSERVATIONS: Here for test data we got a good tpr but fnr is higher which is not good.

SET 3 APPLYING ON AVG W2V

In [91]:

```
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [92]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essay'].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_train.append(vector)

print(len(avg_w2v_train))
print(len(avg_w2v_train[0]))
print(avg_w2v_train[0])
```

```
100%|████████████████████████████████████████████████████████████████
███| 49000/49000 [00:15<00:00, 3225.23it/s]

49000
300
[-4.27690043e-02  2.99505519e-02  2.99708725e-02 -1.29091532e-01
 -2.86888406e-03 -9.21968551e-02 -2.83555971e+00  1.13565651e-01
  4.57965420e-02  9.21222319e-02 -9.71747681e-02  8.33936522e-03
  2.53777826e-03 -1.98172565e-01 -8.37032839e-02 -3.45082565e-02
  4.58569275e-02  5.41827391e-02  3.06074928e-02  1.94089565e-02
 -7.76621449e-03  1.22326232e-02 -1.62297536e-02  4.06333855e-02
  6.60852536e-02  5.26848551e-03  1.30039986e-01  5.73913188e-02
 -9.75773609e-03 -1.02342506e-01 -2.11765777e-01 -4.78608116e-02
 -3.73764710e-02  1.29499783e-01 -1.25238812e-02 -6.97636667e-02
 -5.57457420e-02 -1.50344348e-02 -6.26880072e-02  1.07784475e-01
 -1.05833551e-01  5.91162232e-02 -5.00878246e-02 -1.42280116e-01
  4.37009725e-02 -1.50013261e-01  1.60440826e-01 -6.08131884e-03
  1.29276216e-01 -1.25196116e-01  1.36884174e-02 -5.73795261e-02
  7.36711304e-03 -1.86340420e-02 -1.55386232e-02 -4.74082478e-02
  8.06220435e-03 -3.88924072e-02 -9.25119319e-02  5.90021884e-02
 -5.38923420e-02 -2.80889043e-02  3.58357609e-02 -1.02765014e-01
 -1.05471936e-01  1.34805362e-01  4.98515232e-02 -7.99055667e-02
  1.61829942e-01 -1.10454860e-01 -5.23225145e-02  1.12991507e-02
 -5.21474958e-02 -1.01321145e-01 -6.05622522e-02 -1.68959940e-01
 -8.21421939e-03 -3.57840478e-02  4.95480870e-03  4.32016957e-03
  7.18568420e-02 -2.72985420e-01  6.38506957e-02 -5.03786841e-02
 -2.10519949e-01  4.14755797e-03  1.13352612e-01 -1.26148695e-01
  1.69603826e-01 -1.20843899e-02  6.82917580e-02  5.48898232e-02
 -8.72453565e-02  1.27071216e-01 -3.51445362e-03 -1.39288946e-01
 -2.06118819e+00  1.07709406e-01  1.13823275e-01  2.15524754e-01
 -1.93470748e-01  6.27792726e-02  2.72364029e-02 -5.57799188e-02
  1.10392487e-01 -2.57991116e-02 -6.49770083e-02 -2.23420870e-01
  8.50228842e-02  8.80438551e-03 -1.02422114e-01 -3.41880855e-02
 -9.25892319e-03  2.84398217e-01  1.04062346e-01  4.23817826e-02
 -3.44258986e-01 -1.92490145e-02  7.93945420e-02 -8.30233826e-02
  3.04694203e-02  6.21934580e-02  7.92435272e-02 -1.42555275e-01
  7.45528536e-02 -7.74934710e-02  1.48963348e-02  8.98091739e-03
 -8.14867536e-02  1.64977281e-01  1.46672259e-01 -7.20430290e-03
 -3.17549275e-02 -3.88027536e-02  3.09359275e-02 -1.27439174e-01
  9.47590638e-02 -2.02293014e-02  8.66030261e-02  1.78121283e-01
 -7.89420145e-03 -1.10338986e-03  1.46193614e-01  2.71645072e-02
 -1.10136114e-01  5.58176713e-02 -1.79101000e-02 -3.99104464e-02
  1.08108128e-01  2.23908696e-03 -2.31099754e-02  1.68656232e-02
  1.06084246e-01 -8.98601739e-03 -2.34960246e-02  9.26166217e-02
  6.95872464e-03  7.83508841e-03 -1.16398559e-03  1.12686087e-02
  2.13917575e-02  4.90236841e-02 -5.93842159e-02 -6.93920101e-02
 -8.68038551e-02 -2.94938870e-03 -7.57495471e-02  1.39582049e-01
  3.15803333e-02 -3.28448406e-02 -1.14929894e-01 -8.40411014e-02
 -1.47096096e-02 -8.97495565e-02  6.57587319e-02  2.97878551e-02
  1.13074638e-02  5.01145348e-02 -2.22592928e-01 -2.77057493e-02
  6.68108835e-02  2.44769777e-01  1.70768420e-02 -2.39467826e-02
 -8.27637290e-02  4.36507551e-02 -5.84303449e-02  1.42068855e-02
 -2.55875145e-02 -3.14141783e-02 -7.60351014e-02 -2.94371348e-02
 -1.45958309e-01  3.26725081e-02 -1.66806667e-02 -6.62439130e-02
 -6.91944275e-02 -3.24029957e-02  6.99056377e-02  1.07881845e-01
  2.25324464e-01 -4.50294826e-02 -4.43054391e-02  4.24844855e-02
 -1.64470072e-01  3.01624994e-02  5.88479043e-02 -1.69619181e-02
  1.80341231e-01 -5.57106232e-03  7.67122754e-02  3.79890870e-03
 -1.63871235e-02 -1.61938049e-01 -1.49397306e-01  9.43309986e-03
 -9.19083667e-02 -2.74875377e-02  1.50745275e-02  3.22141014e-02
 -1.49191440e-01 -9.92242174e-02 -1.48547574e-01  2.25739870e-02
 -1.60346740e+00 -1.62118493e-02 -9.25101130e-02  4.07726667e-03
 -1.09287371e-01 -9.49219696e-02 -2.09077652e-02 -7.95148841e-02
 -3.08446304e-02  5.92624464e-03 -9.39174000e-02 -6.12510493e-02
  8.84004391e-02 -8.00795855e-02 -6.47750725e-03  1.43621452e-01
 -2.12662826e-02  3.08402464e-02 -1.26728261e-01  1.78001783e-02
 -5.29732188e-02  8.99124058e-02 -1.89304232e-02 -5.64684110e-02
 -2.51271565e-02 -3.33447507e-02 -1.20408101e-02  1.05048420e-01
  3.23409362e-02 -1.05471851e-01  1.65341212e-01  7.73657652e-02
  5.14667391e-02 -2.46889855e-02  1.08218222e-01 -1.65523946e-01
  1.02392812e-02  2.52036420e-02  2.90652725e-02  1.18513768e-02
 -8.21090449e-02 -6.45677681e-02 -8.17372275e-02  3.22387551e-02
  1.01349160e-01  5.74028025e-02  1.01381080e-02  2.07025928e-02
 -1.45874232e-01  9.24600652e-02 -1.72193478e-02  3.58491304e-02
 -1.37008551e-03 -9.65873913e-03 -7.52338986e-02  5.69609386e-02
  2.50385362e-02 -2.97747754e-02 -2.03469565e-02  5.00191681e-02
 -2.56624203e-02  1.40437968e-01  1.75913043e-04 -3.10119565e-02
 -1.02059232e-02  2.03352377e-02  2.25936275e-02 -5.60839420e-02
  3.10194348e-02  3.15820290e-04  5.61000319e-02  7.25754014e-02
 -8.98637391e-03  1.75197866e-01  1.39516068e-01  6.13657130e-02]
```

In [93]:

```
avg_w2v_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['essay'].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_cv.append(vector)

print(len(avg_w2v_cv))
print(len(avg_w2v_cv[0]))
print(avg_w2v_cv[0])
```

```
100%|████████████████████████████████████████████████████████████████████████
███| 21000/21000 [00:06<00:00, 3184.24it/s]

21000
300
[-8.88774066e-02  1.26692626e-02  5.69602286e-02 -1.90971475e-01
 -8.09821132e-02  1.54789198e-02 -2.77451495e+00  1.52608890e-01
  7.24366000e-02  2.00835665e-01 -8.55655956e-02  1.26207065e-01
 -1.18977868e-01 -8.63281912e-02  3.62239923e-02  2.34982066e-02
  5.03359330e-02 -1.68552362e-02  1.17804609e-01 -4.93694505e-02
 -5.02430308e-02  1.48060062e-01 -6.92699824e-02 -9.14709484e-02
  2.03861319e-02  2.45660703e-02  1.81989295e-01 -4.26678791e-03
 -8.98350747e-02 -7.77294934e-02 -2.46149147e-01 -1.23816360e-01
  7.70696967e-02  3.06670670e-02  2.30626582e-02 -4.51647132e-02
 -3.52089846e-02  7.61103132e-02 -1.12678523e-01  3.21370033e-02
 -1.56937984e-01 -2.59854945e-03 -2.68642473e-02 -1.32693791e-01
  1.49372088e-02 -1.55989225e-01  9.96714636e-02 -2.41848571e-03
  3.60502626e-02 -1.53110699e-01  5.90991538e-02  6.00701105e-02
  8.82202308e-03 -3.71167429e-02  1.19283297e-02 -1.07350982e-01
 -1.99375824e-03 -4.95857308e-02 -2.06446332e-01  1.68527268e-01
  8.94627473e-03 -1.93771792e-02 -4.38609399e-02 -9.67585604e-02
 -2.44404253e-02  6.14606374e-02  8.02785385e-03 -5.84619603e-02
  1.66805737e-01 -1.15349326e-01 -2.16737451e-02  8.20073319e-02
  1.18554945e-02 -1.65292163e-01 -1.09113503e-01 -1.24066119e-01
 -1.36884195e-01 -4.55905824e-03 -9.22754286e-03  3.71498571e-02
  2.51630213e-02 -4.77353803e-01  4.70979087e-02 -1.30393923e-01
 -1.54592626e-01 -8.49436077e-02  5.10300000e-02 -7.51326198e-02
  1.33971549e-01 -3.20441703e-02  1.42828846e-01 -2.41903253e-02
 -8.96452692e-02  6.67565824e-03  2.00383375e-02 -8.69392659e-02
 -2.14306209e+00  1.10461787e-01  7.84033187e-02  1.44850242e-01
 -1.17661527e-01 -5.19658462e-02  1.52157604e-01 -3.64795862e-02
  1.12481605e-01  9.53725319e-02  9.73560505e-02 -1.60735751e-01
 -8.50436154e-03  1.60636176e-01 -1.05705967e-01 -1.88745626e-02
  2.28670725e-02  2.18783318e-01  4.96178904e-02 -5.05404196e-02
 -3.10127342e-01  1.22012407e-02 -4.49758723e-02  8.78375055e-03
  2.54389132e-02  5.82756374e-02 -6.26657582e-03 -1.25727414e-01
  8.00314473e-02  3.01066703e-02  2.73203640e-02  4.47103355e-02
 -2.15075209e-02  1.85538482e-01  1.25016899e-01  7.46365813e-02
  2.23671462e-02  4.40896560e-02  1.30704809e-01 -2.42141376e-01
  9.49153956e-02  6.00924440e-02  3.78097637e-02 -1.78432967e-03
  6.46148725e-02  5.46813418e-02  7.64265126e-02 -5.46791198e-02
 -6.57222173e-02  7.20448220e-02 -1.13434187e-02 -5.63041758e-02
  1.39102931e-01 -3.61281681e-02  4.54661827e-02  1.58648256e-01
  7.67910559e-02  4.42882582e-02 -1.30720730e-01  1.10351404e-01
  3.82221945e-02 -5.85598209e-02 -2.01954825e-02  3.41674088e-02
  8.88244110e-03  7.34200769e-02  1.47274954e-02  1.95168352e-02
 -8.60708725e-02  5.10535495e-02 -2.76061143e-02  4.29293253e-02
  5.43165508e-02 -8.53414978e-02 -2.68729231e-02  8.42290110e-04
 -4.02420537e-02 -1.01956396e-02  5.25319308e-02  3.53966000e-03
  7.34327110e-02  1.23353520e-01 -2.04637056e-01 -2.36499396e-02
 -4.74226297e-02  1.96104654e-01 -2.03478428e-02 -8.02126978e-02
 -2.27025863e-01 -1.39764341e-02 -9.39637742e-02  2.43474451e-02
 -5.75797352e-02  6.48025275e-02 -1.07051270e-01 -2.52500956e-02
 -1.37356067e-01  5.36221978e-03 -5.56620352e-02  1.88429341e-02
 -1.38439955e-01  2.70896132e-02  3.25967538e-02  1.19459484e-01
  2.60401923e-01 -2.84380418e-02 -1.21911773e-01  4.02811868e-02
 -9.22895527e-02  1.05025111e-01  2.54218692e-02 -1.13175514e-01
  4.79173593e-02  2.10538352e-02 -8.68553626e-02  4.70664846e-02
  8.83040385e-02 -2.22951886e-01 -2.41470363e-01 -2.89906297e-02
 -6.02366923e-02 -2.20547405e-02  8.08835165e-03 -2.71022857e-03
 -1.11479765e-01 -1.87843462e-01 -9.91483198e-02 -6.45525648e-02
 -1.20029541e+00  2.69757220e-02 -6.15233110e-02 -1.33437847e-01
  4.82865055e-02 -1.80159320e-01  5.71403637e-03 -5.08232063e-02
 -7.81927747e-02 -6.62439560e-03 -1.27783589e-01 -7.25052644e-02
  7.08344253e-02  4.24757143e-02  3.01313516e-02  1.82372436e-02
  8.43451209e-02  4.48521121e-02 -2.29272637e-01 -1.31245818e-01
 -2.51738363e-02  8.49398582e-02 -5.04401735e-02 -7.30558901e-03
  2.44378165e-02 -6.28323379e-02  1.13442107e-01  1.33127008e-01
  1.03505385e-03 -8.04045560e-02  1.06516304e-01  5.32391538e-02
  1.05608953e-01 -8.34585495e-03  1.95910999e-01 -1.80320025e-01
  2.43640593e-02 -1.60386484e-02 -5.72151000e-02  1.15308000e-01
 -2.63479967e-02 -1.09238688e-01 -1.17672200e-01 -1.70163143e-02
  1.05878009e-01  9.31423297e-02  4.67240512e-02  4.10822143e-02
 -8.32767286e-02  1.28994279e-01 -3.34612165e-02  2.21586297e-01
  2.46864022e-02 -2.97027308e-02 -1.51292458e-01  1.08413442e-01
 -1.49035390e-01 -4.48905495e-02 -8.06048427e-02  4.22295802e-02
  1.54343846e-02  1.41120592e-01  1.17460846e-01  9.67995604e-03
  8.56202593e-02 -6.20874198e-02  6.93133297e-02 -4.09655429e-02
 -3.80914516e-02  5.96483637e-02  1.35487286e-01  8.31964099e-02
  6.12260989e-02  2.61369939e-01  1.39565720e-01  1.05498690e-02]
```

In [94]:

```python
avg_w2v_test =[];
for sentence in tqdm(X_test['essay'].values):
    vector=np.zeros(300)
    cnt_words = 0;
    for word in sentence.split():
        if word in glove_words:
            vector += model[word]
            cnt_words +=1
    if cnt_words != 0:
            vector /= cnt_words
    avg_w2v_test.append(vector)
print(len(avg_w2v_test))
```

```
100%|████████████████████████████████████████████████████████████████████
███| 30000/30000 [00:09<00:00, 3171.79it/s]

30000
```

In [95]:

```
title_train_avgw2v=[]
for sentence in tqdm(X_train['project_title'].values):
    vector=np.zeros(300)
    cnt_words=0;
    for word in sentence.split():
        if word in glove_words:
            vector+=model[word]
            cnt_words+=1
    if cnt_words !=0:
        vector/=cnt_words
    title_train_avgw2v.append(vector)
print(len(title_train_avgw2v))
```

```
100%|████████████████████████████████████████████████████████████████████████████
█| 49000/49000 [00:00<00:00, 105504.24it/s]

49000
```

In [96]:

```
title_cv_avgw2v=[]
for sentence in tqdm(X_cv['project_title'].values):
    vector=np.zeros(300)
    cnt_words=0;
    for word in sentence.split():
        if word in glove_words:
            vector+=model[word]
            cnt_words+=1
    if cnt_words !=0:
        vector/=cnt_words
    title_cv_avgw2v.append(vector)

print(len(title_cv_avgw2v))
```

```
100%|████████████████████████████████████████████████████████████████████████████
██| 21000/21000 [00:00<00:00, 91756.33it/s]

21000
```

In [97]:

```
title_test_avgw2v=[]
for sentence in tqdm(X_test['project_title'].values):
    vector=np.zeros(300)
    cnt_ords=0;
    for word in sentence.split():
        if word in glove_words:
            vector+=model[word]
            cnt_words+=1
    if cnt_words !=0:
        vector/=cnt_words
    title_test_avgw2v.append(vector)
print(len(title_test_avgw2v))
```

```
100%|████████████████████████████████████████████████████████████████████████████
██| 30000/30000 [00:00<00:00, 68975.51it/s]

30000
```

In [98]:

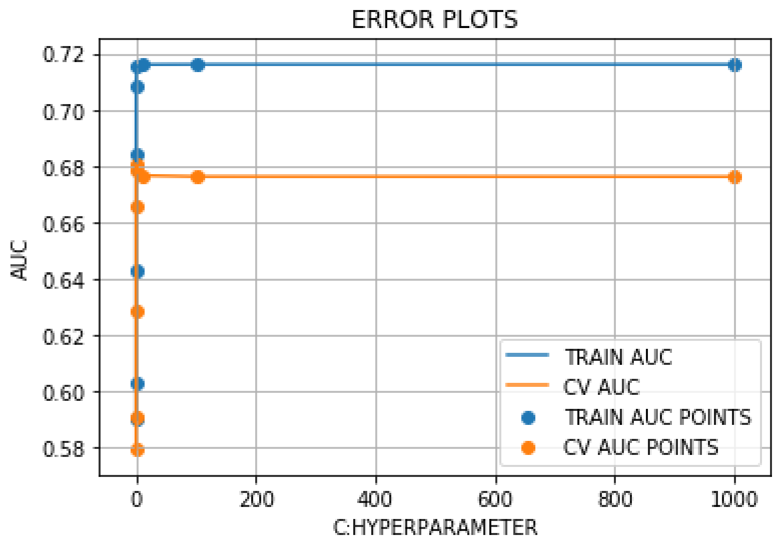
```
from scipy.sparse import hstack
final_train_avgw2v=hstack((X_train_state_encoded,X_train_prefix_encoded,X_train_grade_encod
final_cv_avgw2v=hstack((X_cv_state_encoded,X_cv_prefix_encoded,X_cv_grade_encoded,X_cv_cate
final_test_avgw2v=hstack((X_test_state_encoded,X_test_prefix_encoded,X_test_grade_encoded,X
print(final_train_avgw2v.shape,y_train.shape)
print(final_cv_avgw2v.shape,y_cv.shape)
print(final_test_avgw2v.shape,y_test.shape)
```

```
(49000, 702) (49000,)
(21000, 702) (21000,)
(30000, 702) (30000,)
```

In [99]:

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

avg_w2v_train=[]
avg_w2v_cv=[]
c=[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000]
for i in tqdm(c):
    log_reg=LogisticRegression(C=i,class_weight='balanced')
    log_reg.fit(final_train_avgw2v,y_train)

    y_tr_pred=log_reg.predict_proba(final_train_avgw2v)[:,1]
    y_cv_pred=log_reg.predict_proba(final_cv_avgw2v)[:,1]

    avg_w2v_train.append(roc_auc_score(y_train,y_tr_pred))
    avg_w2v_cv.append(roc_auc_score(y_cv,y_cv_pred))

plt.plot(c,avg_w2v_train,label='TRAIN AUC')
plt.plot(c,avg_w2v_cv,label='CV AUC')

plt.scatter(c,avg_w2v_train,label="TRAIN AUC POINTS")
plt.scatter(c,avg_w2v_cv,label="CV AUC POINTS")
plt.title("ERROR PLOTS")
plt.legend()
plt.xlabel('C:HYPERPARAMETER')
plt.ylabel('AUC')
plt.grid()
plt.show()
```

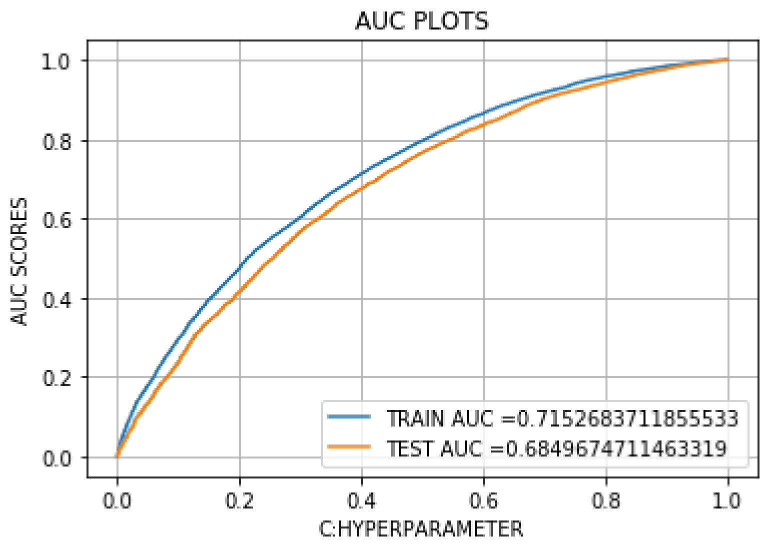
```
100%|██████████████████████████████████████████████████████████████████████
█████████████| 9/9 [03:37<00:00, 44.51s/it]
```



OBSERVATIONS: Here we plotted train and cv auc for 9 different values of c ranging between 0.00001 to 1000. And we choose 1 as our best c.

In [100]:

```
from sklearn.metrics import roc_curve,auc
best_c=1

log_reg=LogisticRegression(C=best_c,class_weight='balanced')
log_reg.fit(final_train_avgw2v,y_train)

y_train_pred=log_reg.predict_proba(final_train_avgw2v)[:,1]
y_test_pred=log_reg.predict_proba(final_test_avgw2v)[:,1]

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

plt.plot(train_fpr,train_tpr,label='TRAIN AUC ='+str(auc(train_fpr,train_tpr)))
plt.plot(test_fpr,test_tpr,label="TEST AUC ="+str(auc(test_fpr,test_tpr)))
plt.legend()
plt.title("AUC PLOTS")
plt.xlabel('C:HYPERPARAMETER')
plt.ylabel("AUC SCORES")
plt.grid()
plt.show()
```



OBSERVATIONS: For c=1 we got train auc of 71.51% and test auc of 68.33%. The gap between train and test auc scores reduced.

In [101]:

```
#printing confusion matrix
print('='*100)
from sklearn.metrics import confusion_matrix
best_t=find_best_threshold(tr_threshold,train_fpr,train_tpr)
print('TRAIN CONFUSION MATRIX')
print(confusion_matrix(y_train,predict_with_best_t(y_train_pred,best_t)))
print('test confusion matrix')
print(confusion_matrix(y_test,predict_with_best_t(y_test_pred,best_t)))
```

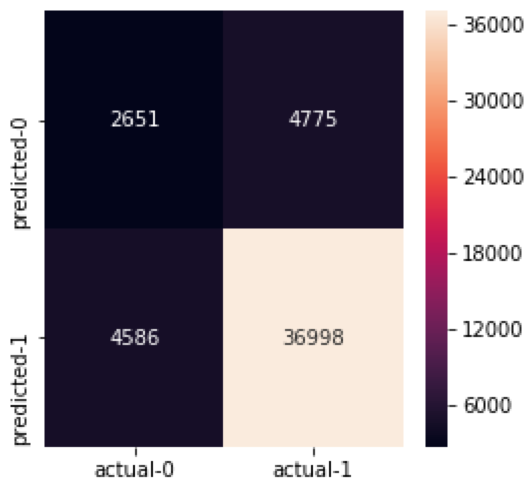
```
============================================================================
========================
the maximum value of tpr*(1-fpr) 0.43211216207544206 for threshold 0.377
TRAIN CONFUSION MATRIX
[[ 2971  4455]
 [ 5636 35938]]
test confusion matrix
[[ 1607  2939]
 [ 3404 22050]]
```

In [102]:

```
#printing heatmap for train confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

array=[[2651,4775],[4586,36998]]

train=pd.DataFrame(array,index=['predicted-0','predicted-1'],columns=['actual-0','actual-1'
plt.figure(figsize=(4,4))
sn.heatmap(train,annot=True,fmt='d')
```

Out[102]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2558aecf9b0>
```



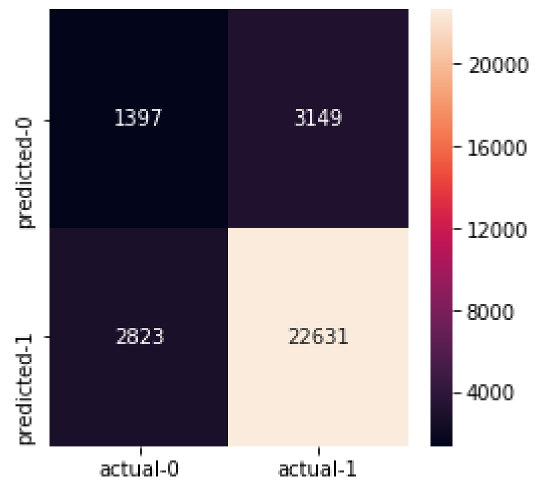
OBSERVATIONS: Here we got good tpr but we got very low tnr which was lower compared to fnr.

In [103]:

```
#printing heatmap for test confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

testarray=[[1397,3149],[2823,22631]]

test=pd.DataFrame(testarray,index=['predicted-0','predicted-1'],columns=['actual-0','actual
plt.figure(figsize=(4,4))
sn.heatmap(test,annot=True,fmt='d')
```

Out[103]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2558aea3320>
```



OBSERVATIONS: Here we got good tpr but we got very low tnr which was lower compared to fnr.

SET 4 APPLYING ON TFIDF WEIGHED W2V

In [105]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['essay'])
# 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 [106]:

```
# compute average word2vec for each review.
tfidf_w2v_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettir
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_train.append(vector)

print(len(tfidf_w2v_train))
print(len(tfidf_w2v_train[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
████| 49000/49000 [02:08<00:00, 381.16it/s]

49000
300
```

In [107]:

```
tfidf_w2v_cv=[]
for sentence in tqdm(X_cv['essay']):
    vector=np.zeros(300)
    tf_idf_weight=0;
    for word in sentence.split():
        if (word in glove_words) and (word in tfidf_words):
            vec=model[word]
            tf_idf=dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector +=(vec * tf_idf)
            tf_idf_weight+= tf_idf
    if tf_idf_weight !=0:
        vector /=tf_idf_weight
    tfidf_w2v_cv.append(vector)

print(len(tfidf_w2v_cv))
print(len(tfidf_w2v_cv[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
████| 21000/21000 [00:54<00:00, 385.62it/s]

21000
300
```

In [108]:

```python
tfidf_w2v_test=[]
for sentence in tqdm(X_test['essay']):
    vector=np.zeros(300)
    tf_idf_weight=0;
    for word in sentence.split():
        if (word in glove_words)and(word in tfidf_words):
            vec=model[word]
            tf_idf=dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector+=(vec*tf_idf)
            tf_idf_weight+=tf_idf
    if tf_idf_weight !=0:
        vector /=tf_idf_weight
    tfidf_w2v_test.append(vector)
print(len(tfidf_w2v_test))
print(len(tfidf_w2v_test[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
████| 30000/30000 [01:18<00:00, 382.08it/s]

30000
300
```

In [109]:

```python
#tfidf weighted w2v vectorizing project title
title_train_tfidfw2v=[]
for sentence in tqdm(X_train['project_title']):
    vector=np.zeros(300)
    tf_idf_weight=0;
    for word in sentence.split():
        if(word in glove_words)and(word in tfidf_words):
            vec=model[word]
            tf_idf=dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector+=(vec*tf_idf)
            tf_idf_weight+=tf_idf
    if tf_idf_weight !=0:
        vector /=tf_idf_weight
    title_train_tfidfw2v.append(vector)
print(len(title_train_tfidfw2v))
print(len(title_train_tfidfw2v[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
█| 49000/49000 [00:00<00:00, 103912.84it/s]

49000
300
```

In [110]:

```
title_cv_tfidfw2v=[]
for sentence in tqdm(X_cv['project_title']):
    vector=np.zeros(300)
    tf_idf_weight=0;
    for word in sentence.split():
        if (word in glove_words) and (word in tfidf_words):
            vec=model[word]
            tf_idf=dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector+=(vec*tf_idf)
            tf_idf_weight+=tf_idf
    if tf_idf_weight !=0:
        vector /=tf_idf_weight
    title_cv_tfidfw2v.append(vector)
print(len(title_cv_tfidfw2v))
print(len(title_cv_tfidfw2v[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
██| 21000/21000 [00:00<00:00, 84726.57it/s]

21000
300
```

In [111]:

```
title_test_tfidfw2v=[]
for sentence in tqdm(X_test['project_title']):
    vector=np.zeros(300)
    tf_idf_weight=0;
    for word in sentence.split():
        if (word in glove_words)and(word in tfidf_words):
            vec=model[word]
            tf_idf=dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector+=(vec*tf_idf)
            tf_idf_weight+=tf_idf
    if tf_idf_weight !=0:
        vector /=tf_idf_weight
    title_test_tfidfw2v.append(vector)

print(len(title_test_tfidfw2v))
print(len(title_test_tfidfw2v[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
██| 30000/30000 [00:00<00:00, 88709.95it/s]

30000
300
```

In [112]:

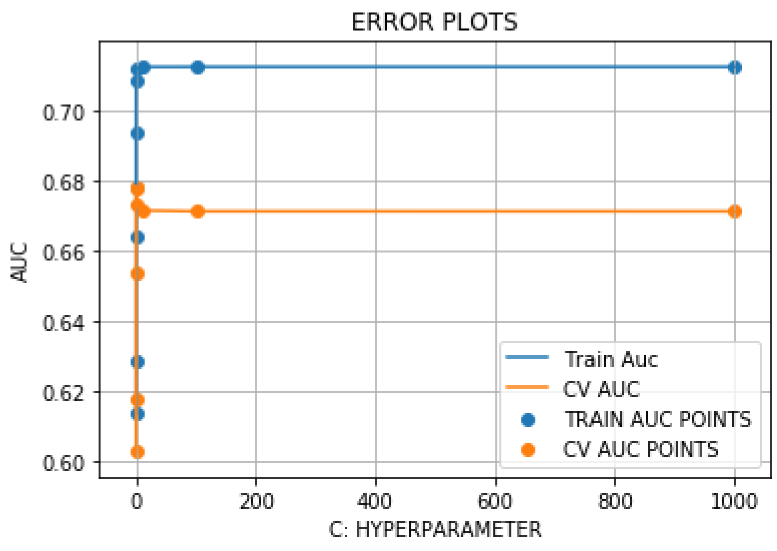
```
#creating data matrix
from scipy.sparse import hstack
final_train_tfidfw2v=hstack((X_train_state_encoded,X_train_prefix_encoded,X_train_grade_enc
final_cv_tfidfw2v=hstack((X_cv_state_encoded,X_cv_prefix_encoded,X_cv_grade_encoded,X_cv_ca
final_test_tfidfw2v=hstack((X_test_state_encoded,X_test_prefix_encoded,X_test_grade_encodec
print(final_train_tfidfw2v.shape,y_train.shape)
print(final_cv_tfidfw2v.shape,y_cv.shape)
print(final_test_tfidfw2v.shape,y_test.shape)
```

```
(49000, 702) (49000,)
(21000, 702) (21000,)
(30000, 702) (30000,)
```

In [113]:

```python
#plotting error plots
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

train_auc=[]
cv_auc=[]
c=[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000]
for i in tqdm(c):
    log_reg=LogisticRegression(C=i,class_weight='balanced')
    log_reg.fit(final_train_tfidfw2v,y_train)

    y_tr_pred=log_reg.predict_proba(final_train_tfidfw2v)[:,1]
    y_cv_pred=log_reg.predict_proba(final_cv_tfidfw2v)[:,1]

    train_auc.append(roc_auc_score(y_train,y_tr_pred))
    cv_auc.append(roc_auc_score(y_cv,y_cv_pred))

plt.plot(c,train_auc,label='Train Auc')
plt.plot(c,cv_auc,label='CV AUC')
plt.scatter(c,train_auc,label='TRAIN AUC POINTS')
plt.scatter(c,cv_auc,label='CV AUC POINTS')
plt.title('ERROR PLOTS')
plt.xlabel('C: HYPERPARAMETER')
plt.ylabel("AUC")
plt.legend()
plt.grid()
plt.show()
```

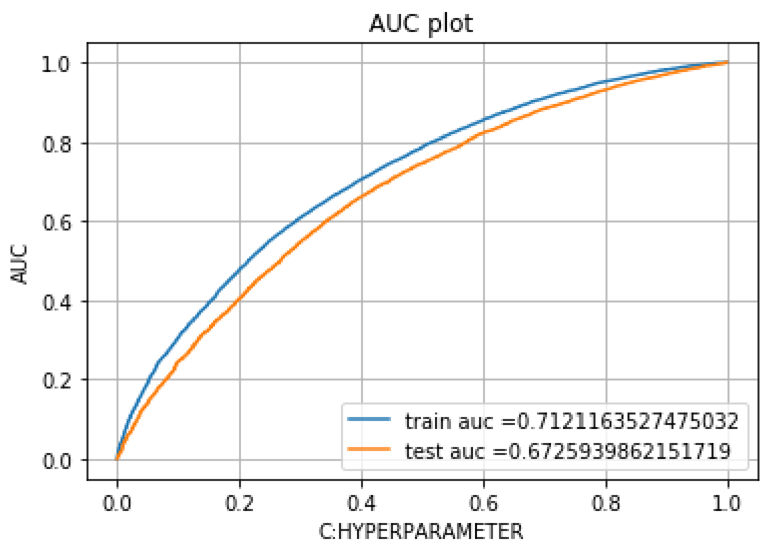
```
100%|████████████████████████████████████████████████████
█████████| 9/9 [03:08<00:00, 38.19s/it]
```



OBSERVATIONS: Here we plotted train and cv auc for 9 different values of c ranging between 0.00001 to 1000. And we choose 1 as our best c.

In [119]:

```
from sklearn.metrics import roc_curve,auc
from sklearn.linear_model import LogisticRegression

best_c=1
log_reg=LogisticRegression(C=best_c,class_weight='balanced')
log_reg.fit(final_train_tfidfw2v,y_train)

y_train_pred=log_reg.predict_proba(final_train_tfidfw2v)[:,1]
y_test_pred=log_reg.predict_proba(final_test_tfidfw2v)[:,1]

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

plt.plot(train_fpr,train_tpr,label="train auc ="+str(auc(train_fpr,train_tpr)))
plt.plot(test_fpr,test_tpr,label='test auc ='+str(auc(test_fpr,test_tpr)))
plt.legend()
plt.title("AUC plot")
plt.xlabel("C:HYPERPARAMETER")
plt.ylabel("AUC")
plt.grid()
plt.show()
```



OBSERVATIONS: For c=1 we got train auc of 71.10% and test auc of 67.11%.

In [115]:

```
#printing confusion matrix
print('='*100)
from sklearn.metrics import confusion_matrix
best_t=find_best_threshold(tr_threshold,train_fpr,train_tpr)
print('TRAIN CONFUSION MATRIX')
print(confusion_matrix(y_train,predict_with_best_t(y_train_pred,best_t)))
print('test confusion matrix')
print(confusion_matrix(y_test,predict_with_best_t(y_test_pred,best_t)))
```

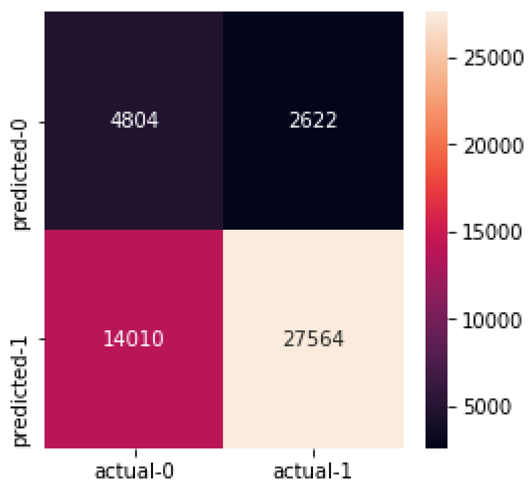
```
============================================================================
========================
the maximum value of tpr*(1-fpr) 0.4289122828184156 for threshold 0.49
TRAIN CONFUSION MATRIX
[[ 4804  2622]
 [14010 27564]]
test confusion matrix
[[ 2782  1764]
 [ 8914 16540]]
```

In [116]:

```
#printing heatmap for train confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

array=[[4804,2622],[14010,27564]]

train=pd.DataFrame(array,index=['predicted-0','predicted-1'],columns=['actual-0','actual-1'
plt.figure(figsize=(4,4))
sn.heatmap(train,annot=True,fmt='d')
```

Out[116]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2559ed671d0>
```



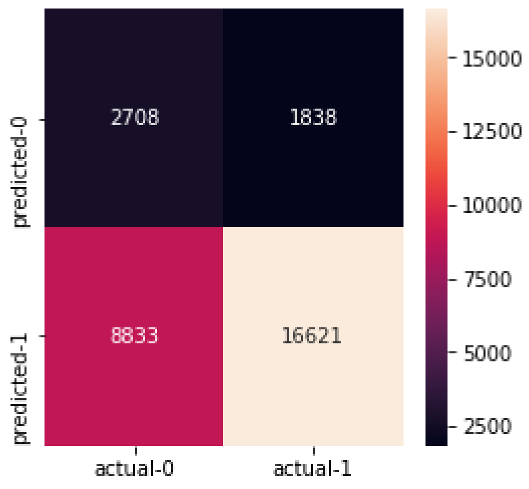
OBSERVATIONS: Here we got a decent tnr and tpr for training data. But fpr values are a bit high.

In [116]:

```
#printing heatmap for test confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

testarray=[[2708,1838],[8833,16621]]

test=pd.DataFrame(testarray,index=['predicted-0','predicted-1'],columns=['actual-0','actual
plt.figure(figsize=(4,4))
sn.heatmap(test,annot=True,fmt='d')
```

Out[116]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2ac0a503eb8>
```



OBSERVATIONS: Here we got a decent tnr and tpr for test data. Both fpr and fnr values are a bit high.

2.5 Logistic Regression with added Features `Set 5`

In [0]:

```
# please write all the code with proper documentation, and proper titles for each subsectic
# 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 coc
# 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 [120]:

```
project_data.columns
```

Out[120]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', '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_approve
d',
       'clean_categories', 'clean_subcategories', 'clean_grades', 'essay',
       'price', 'quantity'],
      dtype='object')
```

CALCULATING SENTIMENT SCORES

In [121]:

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

# import nltk
# nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()
sentiment_score=[]
for sentence in tqdm(project_data['essay'].values):
    senti=sid.polarity_scores(sentence)
    sentiment_score.append(senti)

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

```
[nltk_data] Error loading vader_lexicon: <urlopen error [Errno 11001]
[nltk_data]     getaddrinfo failed>

100%|████████████████████████████████████████████████████████████████████████
█| 109248/109248 [08:42<00:00, 209.05it/s]

109248
4
```

In [122]:

```
sentiment_score[0:5]
```

Out[122]:

```
[{'neg': 0.008, 'neu': 0.911, 'pos': 0.081, 'compound': 0.9611},
 {'neg': 0.037, 'neu': 0.851, 'pos': 0.112, 'compound': 0.9267},
 {'neg': 0.058, 'neu': 0.764, 'pos': 0.179, 'compound': 0.995},
 {'neg': 0.052, 'neu': 0.733, 'pos': 0.214, 'compound': 0.9931},
 {'neg': 0.016, 'neu': 0.897, 'pos': 0.087, 'compound': 0.9192}]
```

GETTING NUMBER OF WORDS IN A TITLE

In [123]:

```
title_words=[]
for sentence in tqdm(project_data['project_title'].values):
    sent=sentence.split()
    title_words.append(len(sent))

print(len(title_words))
```

```
100%|██████████████████████████████████████████████████████████████████████|
109248/109248 [00:00<00:00, 513195.38it/s]

109248
```

In [124]:

```
title_words[0:5]
```

Out[124]:

```
[7, 5, 7, 2, 3]
```

GETTING NUMBER OF WORDS IN ESSAY

In [125]:

```
essay_words=[]
for sentence in tqdm(project_data['essay'].values):
    sent=sentence.split()
    essay_words.append(len(sent))

print(len(essay_words))
```

```
100%|███████████████████████████████████████████████████████████████████████
| 109248/109248 [00:03<00:00, 34855.26it/s]

109248
```

In [126]:

```
essay_words[0:12]
```

Out[126]:

```
[272, 221, 361, 213, 234, 291, 196, 301, 302, 235, 260, 473]
```

In [127]:

```
#normalizing number of words in essay
#https://scipython.com/book/chapter-2-the-core-python-language-i/questions/normalizing-a-li
essaymin=min(essay_words)
essaymax=max(essay_words)
for i,val in enumerate(essay_words):
    essay_words[i]=(val-essaymin)/(essaymax-essaymin)

print(len(essay_words))
```

```
109248
```

In [128]:

```
essay_words[0:5]
```

Out[128]:

```
[0.38734177215189874,
 0.2582278481012658,
 0.6126582278481013,
 0.2379746835443038,
 0.2911392405063291]
```

In [129]:

```
#normalizing number of words in title
titlemin=min(title_words)
titlemax=max(title_words)
for i,val in enumerate(title_words):
    title_words[i]=(val-titlemin)/(titlemax-titlemin)
print(len(title_words))
```

```
109248
```

In [130]:

```
title_words[0:5]
```

Out[130]:

```
[0.5, 0.3333333333333333, 0.5, 0.08333333333333333, 0.16666666666666666]
```

In [131]:

```
import pandas as pd
essay_words_df=pd.DataFrame(essay_words)
print(essay_words_df.shape)
```

```
(109248, 1)
```

In [132]:

```
import pandas as pd
title_words_df=pd.DataFrame(title_words)
print(title_words_df.shape)
```

```
(109248, 1)
```

In [133]:

```
import pandas as pd
senti_df=pd.DataFrame(sentiment_score)
print(senti_df.shape)
```

```
(109248, 4)
```

In [134]:

```
senti_df.columns
```

Out[134]:

```
Index(['compound', 'neg', 'neu', 'pos'], dtype='object')
```

In [135]:

```
#getting final data matrix
from scipy.sparse import hstack
finaldata=hstack((categories_one_hot,sub_categories_one_hot,school_state_one_hot,project_gr
finaldata.shape
```

Out[135]:

```
(109248, 108)
```

In [136]:

```
finaldata1=finaldata[0:100000]
finaldata1.shape
```

Out[136]:

```
(100000, 108)
```

In [137]:

```
#splitting final data into train,cv test
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(finaldata1,y,test_size=0.30,stratify=y)
X_train,X_cv,y_train,y_cv=train_test_split(X_train,y_train,test_size=0.30,stratify=y_train)
```

In [138]:

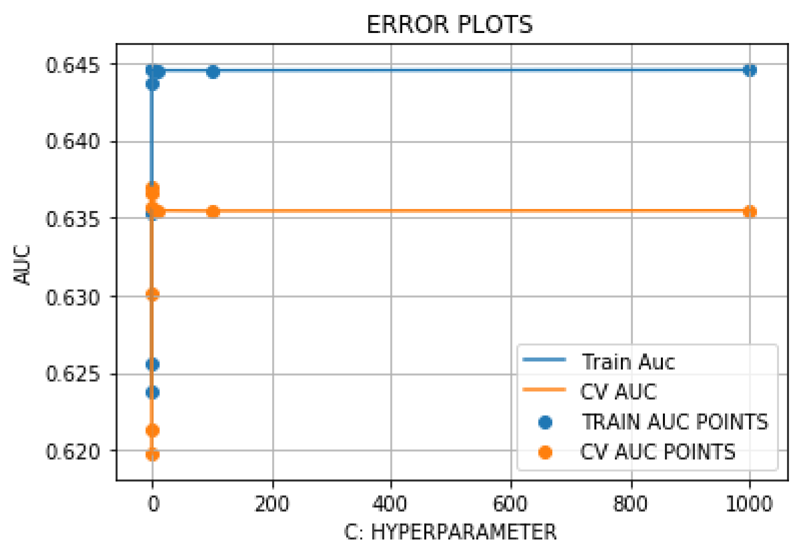
```
print(X_train.shape,y_train.shape)
print(X_cv.shape,y_cv.shape)
print(X_test.shape,y_test.shape)
```

```
(49000, 108) (49000,)
(21000, 108) (21000,)
(30000, 108) (30000,)
```

In [139]:

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

train_auc=[]
cv_auc=[]
c=[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000]
for i in tqdm(c):
    log_reg=LogisticRegression(C=i,class_weight='balanced')
    log_reg.fit(X_train,y_train)

    y_tr_pred=log_reg.predict_proba(X_train)[:,1]
    y_cv_pred=log_reg.predict_proba(X_cv)[:,1]

    train_auc.append(roc_auc_score(y_train,y_tr_pred))
    cv_auc.append(roc_auc_score(y_cv,y_cv_pred))

plt.plot(c,train_auc,label='Train Auc')
plt.plot(c,cv_auc,label='CV AUC')
plt.scatter(c,train_auc,label='TRAIN AUC POINTS')
plt.scatter(c,cv_auc,label='CV AUC POINTS')
plt.title('ERROR PLOTS')
plt.xlabel('C: HYPERPARAMETER')
plt.ylabel("AUC")
plt.legend()
plt.grid()
plt.show()
```

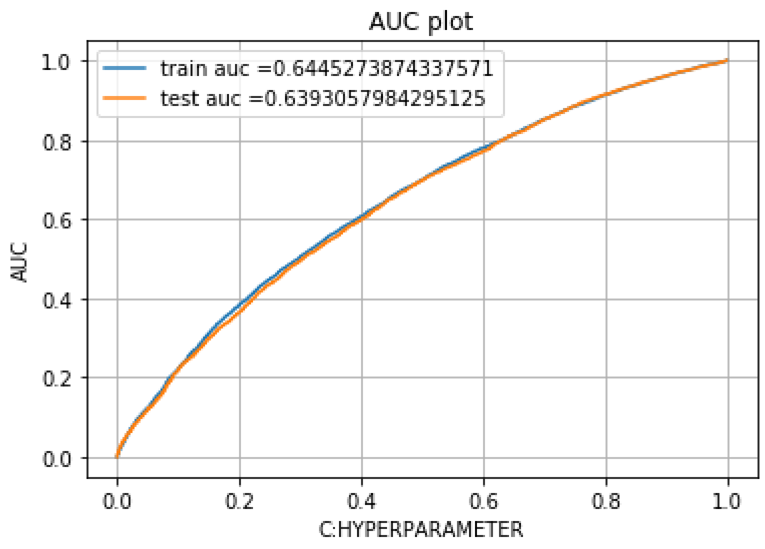
```
100%|████████████████████████████████████████████████
█████████| 9/9 [00:20<00:00,  4.67s/it]
```



OBSERVATIONS: Here we plotted train and cv auc for 9 different values of c ranging between 0.00001 to 1000. And we choose 10 as our best c.

In [140]:

```
from sklearn.metrics import roc_curve,auc
from sklearn.linear_model import LogisticRegression

best_c=10
log_reg=LogisticRegression(C=best_c,class_weight='balanced')
log_reg.fit(X_train,y_train)

y_train_pred=log_reg.predict_proba(X_train)[:,1]
y_test_pred=log_reg.predict_proba(X_test)[:,1]

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

plt.plot(train_fpr,train_tpr,label="train auc ="+str(auc(train_fpr,train_tpr)))
plt.plot(test_fpr,test_tpr,label='test auc ='+str(auc(test_fpr,test_tpr)))
plt.legend()
plt.title("AUC plot")
plt.xlabel("C:HYPERPARAMETER")
plt.ylabel("AUC")
plt.grid()
plt.show()
```



OBSERVATIONS: For c=10 we got train auc of 64.8% and test auc of 63.38%. Here the train and test auc are almost similar.

In [141]:

```
#printing confusion matrix
print('='*100)
from sklearn.metrics import confusion_matrix
best_t=find_best_threshold(tr_threshold,train_fpr,train_tpr)
print('TRAIN CONFUSION MATRIX')
print(confusion_matrix(y_train,predict_with_best_t(y_train_pred,best_t)))
print('test confusion matrix')
print(confusion_matrix(y_test,predict_with_best_t(y_test_pred,best_t)))
```

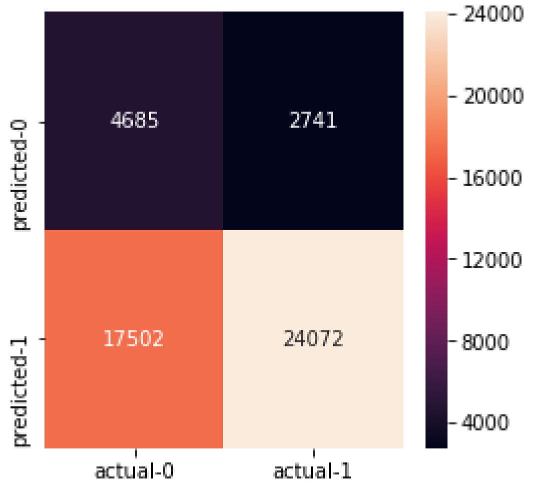
```
============================================================================
========================
the maximum value of tpr*(1-fpr) 0.3652960812911476 for threshold 0.505
TRAIN CONFUSION MATRIX
[[ 4685  2741]
 [17502 24072]]
test confusion matrix
[[ 2821  1725]
 [10666 14788]]
```

In [142]:

```
#printing heatmap for train confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

array=[[4685,2741],[17502,24072]]

train=pd.DataFrame(array,index=['predicted-0','predicted-1'],columns=['actual-0','actual-1'
plt.figure(figsize=(4,4))
sn.heatmap(train,annot=True,fmt='d')
```

Out[142]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2558cfb6748>
```



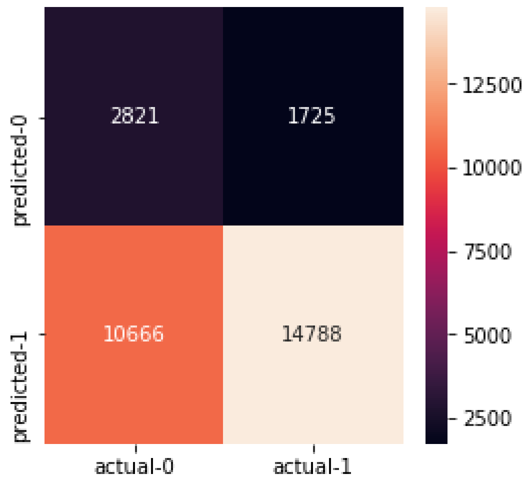
OBSERVATIONS: Here we got a decent tnr and tpr scores but fnr and fpr scores are a bit large for training data.

In [143]:

```
#printing heatmap for test confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

testarray=[[2821,1725],[10666,14788]]

test=pd.DataFrame(testarray,index=['predicted-0','predicted-1'],columns=['actual-0','actual
plt.figure(figsize=(4,4))
sn.heatmap(test,annot=True,fmt='d')
```

Out[143]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2558aeba8d0>
```



OBSERVATIONS: Here we got a decent tnr and tpr scores but fnr and fpr scores are a bit large for training data.

3. Conclusion

In [145]:

```
# Please compare all your models using Prettytable library
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
x=PrettyTable(['vectorizer','best_c','train_auc','test_auc'])
x.add_row(["bag of words",0.1,0.859835,0.681432])
x.add_row(["avgw2v",1,0.715164,0.683335])
x.add_row(["tfidf",1,0.875818,0.691514])
x.add_row(["tfidf_w2v",1,0.711037,0.671196])
x.add_row(['Without text data',10,0.648312,0.633895])

print(x.get_string(start=0,end=7))
```

```
+-------------------+--------+-----------+----------+
|     vectorizer    | best_c | train_auc | test_auc |
+-------------------+--------+-----------+----------+
|    bag of words   |  0.1   |  0.859835 | 0.681432 |
|       avgw2v      |   1    |  0.715164 | 0.683335 |
|       tfidf       |   1    |  0.875818 | 0.691514 |
|     tfidf_w2v     |   1    |  0.711037 | 0.671196 |
| Without text data |   10   |  0.648312 | 0.633895 |
+-------------------+--------+-----------+----------+
```

OBSERVATIONS: We plotted for different vectorizers. For bag of words and tfidf we got train auc of more than 85%.For avgw2v and tfidfw2v we got train auc of more than 70%. The test auc scores are almost similar for all cases. Without using text data we got train and test auc's as 64.8 and 62.333 respectively. We chose our best C =1 for most of the cases.

In [ ]: