DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

Feature	Description
`project_id`	A unique identifier for the proposed p **Example:** `p036502`

Feature	Description
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the follow enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) su categories for the project from the fo enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Ma & Science`

Feature	Description
school_state	State where school is located (Two-l U.S. postal code). **Example:** WY
project_subject_subcategories	One or more (comma-separated) su subcategories for the project. **Exam** • Literacy • Literature & Writing, Social Sciences
project_resource_summary	An explanation of the resources nee the project. **Example:** • My students need hands literacy materials to manage sensory needs!
project_essay_1	First application essay*
project_essay_2	Second application essay*
project_essay_3	Third application essay*
project_essay_4	Fourth application essay*
project_submitted_datetime	Datetime when project application w submitted. **Example:** 2016-04-28 12:43:56.245
teacher_id	A unique identifier for the teacher of proposed project. **Example:** bdf8baa8fedef6bfeec7ae4ff1c

Feature	Description
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previ submitted by the same teacher. **Exa** 2

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
```

```
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

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

from tqdm import tqdm
import os

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

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.
```

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_prefi
x' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [4]:
```
print("Number of data points in train data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

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

Out[4]:

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

1.2 preprocessing of `project_subject_categories`

In [5]:
```
catogories = list(project_data['project_subject_categories'].values)
```

```
# remove special characters from list of strings python: https://stacko
verflow.com/a/47301924/4084039

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

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

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

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

1.3 preprocessing of project_subject_subcategories

In [6]:
```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039

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

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

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

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
```

```
my_counter = Counter()
for word in project_data['clean_subcategories'].values:
    my_counter.update(word.split())

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

1.3 Text preprocessing

In [7]:
```
# merge two column text dataframe:
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(str) + \
                        project_data["project_essay_4"].map(str)
```

In [8]:
```
project_data.head(2)
```

Out[8]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL

In [9]:
```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

In [10]:
```
# printing some random reviews
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
print(project_data['essay'].values[1000])
print("="*50)
print(project_data['essay'].values[20000])
print("="*50)
print(project_data['essay'].values[99999])
print("="*50)
```

```
My students are English learners that are working on English as their s
econd 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 have over 24 languages represented in our English Learner p
rogram with students at every level of mastery.  We also have over 40 c
ountries represented with the families within our school.  Each student
brings a wealth of knowledge and experiences to us that open our eyes t
o new cultures, beliefs, and respect.\"The limits of your language are
the limits of your world.\"-Ludwig Wittgenstein  Our English learner's
have a strong support system at home that begs for more resources.  Man
y times our parents are learning to read and speak English along side o
```

f their children. Sometimes this creates barriers for parents to be ab
le to help their child learn phonetics, letter recognition, and other r
eading skills.\r\n\r\nBy providing these dvd's and players, students ar
e able to continue their mastery of the English language even if no one
at home is able to assist. All families with students within the Level
1 proficiency status, will be a offered to be a part of this program.
These educational videos will be specially chosen by the English Learne
r Teacher and will be sent home regularly to watch. The videos are to
help the child develop early reading skills.\r\n\r\nParents that do not
have access to a dvd player will have the opportunity to check out a dv
d player to use for the year. The plan is to use these videos and educ
ational dvd's for the years to come for other EL students.\r\nnannan
==========
The 51 fifth grade students that will cycle through my classroom this y
ear all love learning, at least most of the time. At our school, 97.3%
of the students receive free or reduced price lunch. Of the 560 student
s, 97.3% are minority students. \r\nThe school has a vibrant community
that loves to get together and celebrate. Around Halloween there is a w
hole school parade to show off the beautiful costumes that students wea
r. On Cinco de Mayo we put on a big festival with crafts made by the st
udents, dances, and games. At the end of the year the school hosts a ca
rnival 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 fi
ve brightly colored Hokki stools in place of regular, stationary, 4-leg
ged chairs. As I will only have a total of ten in the classroom and not
enough for each student to have an individual one, they will be used in
a variety of ways. During independent reading time they will be used as
special chairs students will each use on occasion. I will utilize them
in place of chairs at my small group tables during math and reading tim
es. The rest of the day they will be used by the students who need the
highest amount of movement in their life in order to stay focused on sc
hool.\r\n\r\nWhenever asked what the classroom is missing, my students
always say more Hokki Stools. They can't get their fill of the 5 stools
we already have. When the students are sitting in group with me on the
Hokki Stools, they are always moving, but at the same time doing their
work. Anytime the students get to pick where they can sit, the Hokki St
ools are the first to be taken. There are always students who head over
to the kidney table to get one of the stools who are disappointed as th
ere are not enough of them. \r\n\r\nWe ask a lot of students to sit for

7 hours a day. The Hokki stools will be a compromise that allow my stud
ents to do desk work and move at the same time. These stools will help
students to meet their 60 minutes a day of movement by allowing them to
activate their core muscles for balance while they sit. For many of my
students, these chairs will take away the barrier that exists in school
s for a child who can't sit still.nannan
==========
How do you remember your days of school? Was it in a sterile environmen
t with plain walls, rows of desks, and a teacher in front of the room?
A typical day in our room is nothing like that. I work hard to create a
warm inviting themed room for my students look forward to coming to eac
h day.\r\n\r\nMy class is made up of 28 wonderfully unique boys and gir
ls of mixed races in Arkansas.\r\nThey attend a Title I school, which m
eans there is a high enough percentage of free and reduced-price lunch
to qualify. Our school is an \"open classroom\" concept, which is very
unique as there are no walls separating the classrooms. These 9 and 10
year-old students are very eager learners; they are like sponges, absor
bing all the information and experiences and keep on wanting more.With
these resources such as the comfy red throw pillows and the whimsical n
autical hanging decor and the blue fish nets, I will be able to help cr
eate the mood in our classroom setting to be one of a themed nautical e
nvironment. Creating a classroom environment is very important in the s
uccess in each and every child's education. The nautical photo props wi
ll 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 chil
d with them, have them developed, and then hung in our classroom ready
for their first day of 4th grade. This kind gesture will set the tone
before even the first day of school! The nautical thank you cards will
be used throughout the year by the students as they create thank you ca
rds to their team groups.\r\n\r\nYour generous donations will help me t
o help make our classroom a fun, inviting, learning environment from da
y one.\r\n\r\nIt costs lost of money out of my own pocket on resources
to get our classroom ready. Please consider helping with this project t
o make our new school year a very successful one. Thank you!nannan
==========
My kindergarten students have varied disabilities ranging from speech a
nd language delays, cognitive delays, gross/fine motor delays, to autis
m. They are eager beavers and always strive to work their hardest worki
ng past their limitations. \r\n\r\nThe materials we have are the ones I

```
seek out for my students. I teach in a Title I school where most of the
students receive free or reduced price lunch.  Despite their disabiliti
es 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 an
d you needed to groove and move as you were in a meeting? This is how m
y kids feel all the time. The want to be able to move as they learn or
so they say.Wobble chairs are the answer and I love then because they d
evelop their core, which enhances gross motor and in Turn fine motor sk
ills. \r\nThey also want to learn through games, my kids don't want to
sit and do worksheets. They want to learn to count by jumping and playi
ng. Physical engagement is the key to our success. The number toss and
color and shape mats can make that happen. My students will forget they
are doing work and just have the fun a 6 year old deserves.nannan
==================================================
The mediocre teacher tells. The good teacher explains. The superior tea
cher demonstrates. The great teacher inspires. -William A. Ward\r\n\r\n
My school has 803 students which is makeup is 97.6% African-American, m
aking up the largest segment of the student body. A typical school in D
allas is made up of 23.2% African-American students. Most of the studen
ts are on free or reduced lunch. We aren't receiving doctors, lawyers,
or engineers children from rich backgrounds or neighborhoods. As an edu
cator I am inspiring minds of young children and we focus not only on a
cademics but one smart, effective, efficient, and disciplined students
with good character.In our classroom we can utilize the Bluetooth for s
wift transitions during class. I use a speaker which doesn't amplify th
e sound enough to receive the message. Due to the volume of my speaker
my students can't hear videos or books clearly and it isn't making the
lessons as meaningful. But with the bluetooth speaker my students will
be able to hear and I can stop, pause and replay it at any time.\r\nThe
cart will allow me to have more room for storage of things that are nee
ded for the day and has an extra part to it I can use.  The table top c
hart has all of the letter, words and pictures for students to learn ab
out different letters and it is more accessible.nannan
==================================================
```

In [11]:

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

def decontracted(phrase):
```

```
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

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

In [12]:
```
sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

```
My kindergarten students have varied disabilities ranging from speech a
nd language delays, cognitive delays, gross/fine motor delays, to autis
m. They are eager beavers and always strive to work their hardest worki
ng past their limitations. \r\n\r\nThe materials we have are the ones I
seek out for my students. I teach in a Title I school where most of the
students receive free or reduced price lunch.  Despite their disabiliti
es 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 an
d you needed to groove and move as you were in a meeting? This is how m
y kids feel all the time. The want to be able to move as they learn or
so they say.Wobble chairs are the answer and I love then because they d
evelop their core, which enhances gross motor and in Turn fine motor sk
ills. \r\nThey also want to learn through games, my kids do not want to
sit and do worksheets. They want to learn to count by jumping and playi
ng. Physical engagement is the key to our success. The number toss and
color and shape mats can make that happen. My students will forget they
are doing work and just have the fun a 6 year old deserves.nannan
==================================================
```

In [13]:
```
# \r \n \t remove from string python: http://texthandler.com/info/remov
```

```
e-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 a
nd language delays, cognitive delays, gross/fine motor delays, to autis
m. They are eager beavers and always strive to work their hardest worki
ng past their limitations.     The materials we have are the ones I see
k out for my students. I teach in a Title I school where most of the st
udents receive free or reduced price lunch.  Despite their disabilities
and limitations, my students love coming to school and come eager to le
arn and explore.Have you ever felt like you had ants in your pants and
you needed to groove and move as you were in a meeting? This is how my
kids feel all the time. The want to be able to move as they learn or so
they say.Wobble chairs are the answer and I love then because they deve
lop their core, which enhances gross motor and in Turn fine motor skill
s.   They also want to learn through games, my kids do not want to sit
and do worksheets. They want to learn to count by jumping and playing.
Physical engagement is the key to our success. The number toss and colo
r and shape mats can make that happen. My students will forget they are
doing work and just have the fun a 6 year old deserves.nannan
```

In [14]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

```
My kindergarten students have varied disabilities ranging from speech a
nd language delays cognitive delays gross fine motor delays to autism T
hey are eager beavers and always strive to work their hardest working p
ast their limitations The materials we have are the ones I seek out for
my students I teach in a Title I school where most of the students rece
ive free or reduced price lunch Despite their disabilities and limitati
ons my students love coming to school and come eager to learn and explo
re Have you ever felt like you had ants in your pants and you needed to
groove and move as you were in a meeting This is how my kids feel all t
he time The want to be able to move as they learn or so they say Wobble
chairs are the answer and I love then because they develop their core w
hich enhances gross motor and in Turn fine motor skills They also want
```

```
to learn through games my kids do not want to sit and do worksheets The
y 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 mak
e that happen My students will forget they are doing work and just have
the fun a 6 year old deserves nannan
```

In [15]:

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

In [16]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████| 109248/109248 [01:59<00:00, 913.11it/s]
```

In [17]:
```
# after preprocesing
preprocessed_essays[20000]
```

Out[17]:
```
'my kindergarten students varied disabilities ranging speech language d
elays cognitive delays gross fine motor delays autism they eager beaver
s always strive work hardest working past limitations the materials one
s i seek students i teach title i school students receive free reduced
price lunch despite disabilities limitations students love coming schoo
l come eager learn explore have ever felt like ants pants needed groove
move meeting this kids feel time the want able move learn say wobble ch
airs answer i love develop core enhances gross motor turn fine motor sk
ills they also want learn games kids not want sit worksheets they want
learn count jumping playing physical engagement key success the number
toss color shape mats make happen my students forget work fun 6 year ol
d deserves nannan'
```

In [18]:
```
project_data["preprocessed_essays"] = preprocessed_essays

essay_word_count = []

for sentence in project_data["preprocessed_essays"] :
    word = len(sentence.split())
```

```
    essay_word_count.append(word)

project_data["essay_word_count"] = essay_word_count

project_data.head(5)
```

Out[18]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
4	172407	p104768	be1f7507a41f8479dc06f047086a39ec	Mrs.	TX

In [20]:
```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer


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

for a in tqdm(project_data["preprocessed_essays"]) :
    b = sid.polarity_scores(a)['neg']
    c = sid.polarity_scores(a)['pos']
    d = sid.polarity_scores(a)['neu']
    e = sid.polarity_scores(a)['compound']
    neg.append(b)
    pos.append(c)
    neu.append(d)
    compound.append(e)


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

```
project_data.head(5)
```

```
100%|██████████| 109248/109248 [30:10<00:00, 63.30it/s]
```

Out[20]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
4	172407	p104768	be1f7507a41f8479dc06f047086a39ec	Mrs.	TX

5 rows × 24 columns

1.4 Preprocessing of `project_title`

In [21]:
```
# printing some project titles.
for i in range (0,21):

    print(project_data['project_title'].values[i])
    print("="*50)
```

```
Educational Support for English Learners at Home
==================================================
Wanted: Projector for Hungry Learners
==================================================
Soccer Equipment for AWESOME Middle School Students
==================================================
Techie Kindergarteners
==================================================
Interactive Math Tools
==================================================
Flexible Seating for Mrs. Jarvis' Terrific Third Graders!!
==================================================
Chromebooks for Special Education Reading Program
==================================================
```

```
It's the 21st Century
==================================================
Targeting More Success in Class
==================================================
Just For the Love of Reading--\r\nPure Pleasure
==================================================
Reading Changes Lives
==================================================
Elevating Academics and Parent Rapports Through Technology
==================================================
Building Life Science Experiences
==================================================
Everyone deserves to be heard!
==================================================
TABLETS CAN SHOW US THE WORLD
==================================================
Making Recess Active
==================================================
Making Great LEAP's With Leapfrog!
==================================================
Technology Teaches Tomorrow's Talents Today
==================================================
Test Time
==================================================
Wiggling Our Way to Success
==================================================
Magic Carpet Ride in Our Library
==================================================
```

In [22]:

```
preprocessed_titles = []

for dataset in tqdm(project_data['project_title'].values):
    data = decontracted(dataset)  # Replacing some specific and general
 short form into proper word/stopword.
    data = re.sub(r"it's", "it is", data) # Replacing it's with it is a
s it is not part of function decontracted
    data = data.replace('\\r', ' ') # Replacing \r with space
    data = data.replace('\\"', ' ') # Replacing \ with space
    data = data.replace('\\n', ' ') # Replacing \n with space
```

```
    data = re.sub('[^A-Za-z0-9]+', ' ', data) # Replacing special characters with space
    data = re.sub("\S*\d\S*", "", data).strip() # Trimming numbers containing digits

    data = ' '.join(e for e in data.split() if e not in stopwords) # Removing stopwords
    preprocessed_titles.append(data.lower().strip()) # Creating array in all the lower cases.
```

```
100%|██████████| 109248/109248 [00:07<00:00, 15524.18it/s]
```

In [23]:
```
for i in range (0,21):
    print(preprocessed_titles[i])
    print("="*50)
```

```
educational support english learners home
==================================================
wanted projector hungry learners
==================================================
soccer equipment awesome middle school students
==================================================
techie kindergarteners
==================================================
interactive math tools
==================================================
flexible seating mrs jarvis terrific third graders
==================================================
chromebooks special education reading program
==================================================
it century
==================================================
targeting more success class
==================================================
just for love reading pure pleasure
==================================================
reading changes lives
==================================================
elevating academics parent rapports through technology
```

```
==================================================
building life science experiences
==================================================
everyone deserves heard
==================================================
tablets can show us the world
==================================================
making recess active
==================================================
making great leap with leapfrog
==================================================
technology teaches tomorrow talents today
==================================================
test time
==================================================
wiggling our way success
==================================================
magic carpet ride our library
==================================================
```

In [24]:
```
project_data["preprocessed_titles"] = preprocessed_titles

title_word_count = []

for sentence in project_data["preprocessed_titles"] :
    word = len(sentence.split())
    title_word_count.append(word)

project_data["title_word_count"] = title_word_count

project_data.head(5)
```

Out[24]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
4	172407	p104768	be1f7507a41f8479dc06f047086a39ec	Mrs.	TX

5 rows × 26 columns

1.5 Preparing data for models

In [25]:
```
project_data.columns
```

Out[25]:
```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_stat
e',
       'project_submitted_datetime', 'project_grade_category', '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_appr
oved',
       'clean_categories', 'clean_subcategories', 'essay',
       'preprocessed_essays', 'essay_word_count', 'pos', 'neg', 'neu',
       'compound', 'preprocessed_titles', 'title_word_count'],
      dtype='object')
```

we are going to consider

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

```
- teacher_prefix : categorical data

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

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

1.5.1 Vectorizing Categorical data

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

In [26]:
```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True)
categories_one_hot = vectorizer.fit_transform(project_data['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",categories_one_hot.shape)
```

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

In [27]:
```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=True)
sub_categories_one_hot = vectorizer.fit_transform(project_data['clean_s
```

```
ubcategories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",sub_categories_one_hot.s
hape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolveme
nt', 'Extracurricular', 'Civics_Government', 'ForeignLanguages', 'Nutri
tionEducation', 'Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingA
rts', 'CharacterEducation', 'TeamSports', 'Other', 'College_CareerPre
p', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopme
nt', 'ESL', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Healt
h_Wellness', 'AppliedSciences', 'SpecialNeeds', 'Literature_Writing',
'Mathematics', 'Literacy']
Shape of matrix after one hot encodig  (109248, 30)
```

In [28]:
```
school_state_vectorizer = CountVectorizer(lowercase=False, binary=True)
school_state_vectorizer.fit(project_data['school_state'].values)
print(school_state_vectorizer.get_feature_names())

school_state_one_hot = school_state_vectorizer.transform(project_data[
'school_state'].values)
print("Shape of matrix after one hot encodig ",school_state_one_hot.sha
pe)
print("the type of count vectorizer ",type(school_state_one_hot))
```

```
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'H
I', 'IA', 'ID', 'IL', 'IN', 'KS', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI',
'MN', 'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM', 'NV', 'NY',
'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT',
'WA', 'WI', 'WV', 'WY']
Shape of matrix after one hot encodig  (109248, 51)
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
```

In [29]:
```
# https://www.geeksforgeeks.org/python-pandas-dataframe-fillna-to-repla
ce-null-values-in-dataframe/
project_data["teacher_prefix"].fillna("No_Prefix", inplace = True)

teacher_prefix_vectorizer = CountVectorizer(lowercase=False, binary=Tru
e)
```

```
teacher_prefix_vectorizer.fit(project_data['teacher_prefix'].values)
print(teacher_prefix_vectorizer.get_feature_names())

teacher_prefix_one_hot = teacher_prefix_vectorizer.transform(project_da
ta['teacher_prefix'].values)
print("Shape of matrix after one hot encodig ",teacher_prefix_one_hot.s
hape)
```

```
['Dr', 'Mr', 'Mrs', 'Ms', 'No_Prefix', 'Teacher']
Shape of matrix after one hot encodig  (109248, 6)
```

In [30]:

```
my_grade_counter = Counter()

for project_grade in project_data['project_grade_category'].values:

    if (' ' in project_grade):

        project_grade = project_grade.replace(" ", "~")


    my_grade_counter.update(project_grade.split())


project_grade_cat_dict = dict(my_grade_counter)
sorted_project_grade_cat_dict = dict(sorted(project_grade_cat_dict.item
s(), key=lambda kv: kv[1]))

grade_cat_vectorizer = CountVectorizer(vocabulary=list(sorted_project_g
rade_cat_dict.keys()), lowercase=False, binary=True)
grade_cat_vectorizer.fit(project_data['project_grade_category'].values)
print(grade_cat_vectorizer.get_feature_names())

grade_cat_one_hot = grade_cat_vectorizer.transform(project_data['projec
t_grade_category'].values)
print("Shape of matrix after one hot encodig ",grade_cat_one_hot.shape)
```

```
['Grades~9-12', 'Grades~6-8', 'Grades~3-5', 'Grades~PreK-2']
Shape of matrix after one hot encodig  (109248, 4)
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [31]:
```
# We are considering only the words which appeared in at least 10 docum
ents(rows or projects).
vectorizer = CountVectorizer(min_df=10)
text_bow = vectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",text_bow.shape)
```

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

In [32]:
```
titles_vectorizer = CountVectorizer(min_df=10)
titles_bow = titles_vectorizer.fit_transform(preprocessed_titles)
print("some sample features(unique words in the corpus)",titles_vectori
zer.get_feature_names()[0:10])
print("Shape of matrix after one hot encodig ",titles_bow.shape)
print("the type of count vectorizer ",type(titles_bow))
print("the number of unique words ", titles_bow.get_shape()[1])
```

```
some sample features(unique words in the corpus) ['abc', 'abcs', 'abili
ties', 'ability', 'able', 'aboard', 'about', 'above', 'abstract', 'acad
emic']
Shape of matrix after one hot encodig  (109248, 3290)
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the number of unique words  3290
```

1.5.2.2 TFIDF vectorizer

In [33]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
text_tfidf = vectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",text_tfidf.shape)
```

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

1.5.2.3 Using Pretrained Models: Avg W2V

In [34]:
```
'''
# Reading glove vectors in python: https://stackoverflow.com/a/3823034
9/4084039
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print ("Done.",len(model)," words loaded!")
    return model
model = loadGloveModel('glove.42B.300d.txt')

# ============================
Output:

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

# ============================

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

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

inter_words = set(model.keys()).intersection(words)
print("The number of words that are present in both glove vectors and o
```

```
ur coupus", \
      len(inter_words),"(",np.round(len(inter_words)/len(words)*100,
3),"%)")

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


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

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


'''
```

Out[34]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230
349/4084039\ndef loadGloveModel(gloveFile):\n    print ("Loading Glove
Model")\n    f = open(gloveFile,\'r\', encoding="utf8")\n    model = {}
\n    for line in tqdm(f):\n        splitLine = line.split()\n        w
ord = splitLine[0]\n        embedding = np.array([float(val) for val in
splitLine[1:]])\n        model[word] = embedding\n    print ("Done.",le
n(model)," words loaded!")\n    return model\nmodel = loadGloveModel
(\'glove.42B.300d.txt\')\n\n# ============================\nOutput:\n
  \nLoading Glove Model\n1917495it [06:32, 4879.69it/s]\nDone. 1917495
words loaded!\n\n# ============================\n\nwords = []\nfor i in
preproced_texts:\n    words.extend(i.split(\' \'))\n\nfor i in preproce
d_titles:\n    words.extend(i.split(\' \'))\nprint("all the words in th
e coupus", len(words))\nwords = set(words)\nprint("the unique words in
the coupus", len(words))\n\ninter_words = set(model.keys()).intersectio
n(words)\nprint("The number of words that are present in both glove vec
tors and our coupus",       len(inter_words),"(",np.round(len(inter_wor
ds)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nwords_glove = set(mo
del.keys())\nfor i in words:\n    if i in words_glove:\n        words_c
```

```
ourpus[i] = model[i]\nprint("word 2 vec length", len(words_courpus))\n
\n\n# stronging variables into pickle files python: http://www.jessicay
ung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimpo

rt pickle\nwith open(\'glove_vectors\', \'wb\') as f:\n    pickle.dump
(words_courpus, f)\n\n\n'
```

In [35]:
```
# stronging variables into pickle files python: http://www.jessicayung.
com/how-to-use-pickle-to-save-and-load-variables-in-python/
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [36]:
```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored
 in this list
for sentence in tqdm(preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors.append(vector)

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

```
100%|██████████████████████████████████████████████████████████████
██████| 109248/109248 [01:04<00:00, 1689.32it/s]

109248
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [37]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_essays)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model
.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [38]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is store
d 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 sentenc
e/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 t
he tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentenc
e.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors.append(vector)

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

```
100%|██████████████████████████████████████████████████████████████
██████| 109248/109248 [08:08<00:00, 223.71it/s]

109248
300
```

In [39]:
```
# TFIDF on project titles
titles_tfidf_vectorizer = TfidfVectorizer(min_df=10)
titles_tfidf = titles_tfidf_vectorizer.fit_transform(preprocessed_titles)
print("some sample features(unique words in the corpus)",titles_tfidf_vectorizer.get_feature_names()[10:21])
print("Shape of matrix after one hot encodig ",titles_tfidf.shape)
```

```
some sample features(unique words in the corpus) ['academics', 'academy', 'acceptance', 'access', 'accessibility', 'accessible', 'accessing', 'accessories', 'ace', 'achieve', 'achievement']
Shape of matrix after one hot encodig  (109248, 3290)
```

In [40]:
```
# AVG W2V on project title
avg_w2v_titles_vectors = [];

for sentence in tqdm(preprocessed_titles):

    vector_titles = np.zeros(300)
    cnt_words_titles = 0;

    for word in sentence.split():

        if word in glove_words:

            vector += model[word]
            cnt_words_titles += 1

    if cnt_words_titles != 0:

        vector_titles /= cnt_words_titles

    avg_w2v_titles_vectors.append(vector_titles)

print(len(avg_w2v_titles_vectors))
print(len(avg_w2v_titles_vectors[0]))
```

```
100%|
```

```
100%|| 109248/109248 [00:03<00:00, 33119.22it/s]

109248
300
```

In [41]:
```
# TFIDF weighted W2V on project_title
titles_tfidf_model = TfidfVectorizer()
titles_tfidf_model.fit(preprocessed_titles)
titles_dictionary = dict(zip(titles_tfidf_model.get_feature_names(), li
st(titles_tfidf_model.idf_)))
titles_tfidf_words = set(titles_tfidf_model.get_feature_names())
```

In [42]:
```
titles_tfidf_w2v_vectors = [];

for titles_sentence in tqdm(preprocessed_titles):

    titles_vector = np.zeros(300)
    titles_tfidf_weight = 0;

    for word in titles_sentence.split():

        if (word in glove_words) and (word in titles_tfidf_words):

            titles_vec = model[word]

            titles_tf_idf = titles_dictionary[word]*(titles_sentence.co
unt(word)/len(titles_sentence.split()))
            titles_vector += (titles_vec * titles_tf_idf)
            titles_tfidf_weight += titles_tf_idf

    if titles_tfidf_weight != 0:

        titles_vector /= titles_tfidf_weight

    titles_tfidf_w2v_vectors.append(titles_vector)

print(len(titles_tfidf_w2v_vectors))
print(len(titles_tfidf_w2v_vectors[0]))
```

```
100%|██████████| 109248/109248 [00:06<00:00, 16490.88it/s]
```

```
109248
300
```

1.5.3 Vectorizing Numerical features

In [43]:
```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity'
:'sum'}).reset_index()
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

In [44]:
```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/gene
rated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 21
3.03 329.   ... 399.   287.73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)

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

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

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

In [45]:
```
price_standardized
```

Out[45]:
```
array([[-0.3905327 ],
```

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

1.5.4 Merging all the above features

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

In [46]:
```
print(categories_one_hot.shape)
print(sub_categories_one_hot.shape)
print(text_bow.shape)
print(price_standardized.shape)
```

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

In [47]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X = hstack((categories_one_hot, sub_categories_one_hot, text_bow, price_standardized))
X.shape
```

Out[47]:
```
(109248, 16663)
```

Computing Sentiment Scores

In [48]:
```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

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

sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i te
ach the smallest students with the biggest enthusiasm \
for learning my students learn in many different ways using all of our
 senses and multiple intelligences i use a wide range\
of techniques to help all my students succeed students in my class come
 from a variety of different backgrounds which makes\
for wonderful sharing of experiences and cultures including native amer
icans our school is a caring community of successful \
learners which can be seen through collaborative student project based
 learning in and out of the classroom kindergarteners \
in my class love to work with hands on materials and have many differen
t opportunities to practice a skill before it is\
mastered having the social skills to work cooperatively with friends is
 a crucial aspect of the kindergarten curriculum\
montana is the perfect place to learn about agriculture and nutrition m
y students love to role play in our pretend kitchen\
in the early childhood classroom i have had several kids ask me can we
 try cooking with real food i will take their idea \
and create common core cooking lessons where we learn important math an
d writing concepts while cooking delicious healthy \
food for snack time my students will have a grounded appreciation for t
he work that went into making the food and knowledge \
of where the ingredients came from as well as how it is healthy for the
ir bodies this project would expand our learning of \
nutrition and agricultural cooking recipes by having us peel our own ap
ples to make homemade applesauce make our own bread \
and mix up healthy plants from our classroom garden in the spring we wi
ll also create our own cookbooks to be printed and \
shared with families students will gain math and literature skills as w
ell as a life long enjoyment for healthy cooking \
nannan'
ss = sid.polarity_scores(for_sentiment)
```

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

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

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

Assignment 7: SVM

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

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

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

    - Find the best hyper parameter which will give the maximum AUC 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 with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

4. **[Task-2] Apply the Support Vector Machines on these features by finding the best hyper paramter as suggested in step 2 and step 3**

- Consider these set of features Set 5 :
  - **school_state** : categorical data
  - **clean_categories** : categorical data
  - **clean_subcategories** : categorical data
  - **project_grade_category** :categorical data
  - **teacher_prefix** : categorical data
  - **quantity** : numerical data
  - **teacher_number_of_previously_posted_projects** : numerical data
  - **price** : numerical data
  - **sentiment score's of each of the essay** : numerical data
  - **number of words in the title** : numerical data
  - **number of words in the combine essays** : numerical data
  - **Apply TruncatedSVD on TfidfVectorizer of essay text, choose the number of components (`n_components`) using elbow method** : numerical data

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

Note: Data Leakage

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

2. Support Vector Machines

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

In [49]:
```
approved_project = project_data['project_is_approved'].values
project_data.drop(['project_is_approved'], axis=1, inplace=True)
project_data.head(1)
```

Out[49]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	p
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2

1 rows × 27 columns

In [50]:
```
# Data splitting

from sklearn.model_selection import train_test_split

# Splitting in train and test
X_train, X_test, y_train, y_test = train_test_split(project_data, appro
ved_project, test_size=0.33, stratify=approved_project)

# Splitting in Train Test and Cross Validation
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_
size=0.33, stratify=y_train)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [51]:
```
# Vectorizing Categories on Train, Test and CV data
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), l
owercase=False, binary=True)

# Fit only to train data
vectorizer.fit(X_train['clean_categories'].values)

# Transform to train,test and CV data
X_Train_categories_one_hot = vectorizer.transform(X_train['clean_catego
ries'].values)
X_Test_categories_one_hot = vectorizer.transform(X_test['clean_categori
es'].values)
X_CV_categories_one_hot = vectorizer.transform(X_cv['clean_categories']
.values)

print("Shape of train matrix after one hot encodig ",X_Train_categories
_one_hot.shape)
print("Shape of test matrix after one hot encodig ",X_Test_categories_o
ne_hot.shape)
```

```
print("Shape of cv matrix after one hot encodig ",X_CV_categories_one_h
ot.shape)
```

```
Shape of train matrix after one hot encodig  (49041, 9)
Shape of test matrix after one hot encodig  (36052, 9)
Shape of cv matrix after one hot encodig  (24155, 9)
```

In [52]:
```
# Vectorizing subcategories on train, test and cv

vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys
()), lowercase=False, binary=True)
vectorizer.fit(X_train['clean_subcategories'].values)

X_Train_sub_categories_one_hot = vectorizer.transform(X_train['clean_su
bcategories'].values)
X_Test_sub_categories_one_hot = vectorizer.transform(X_test['clean_subc
ategories'].values)
X_CV_sub_categories_one_hot = vectorizer.transform(X_cv['clean_subcateg
ories'].values)

print("Shape of train matrix after one hot encodig ",X_Train_sub_catego
ries_one_hot.shape)
print("Shape of test matrix after one hot encodig ",X_Test_sub_categori
es_one_hot.shape)
print("Shape of cv matrix after one hot encodig ",X_CV_sub_categories_o
ne_hot.shape)
```

```
Shape of train matrix after one hot encodig  (49041, 30)
Shape of test matrix after one hot encodig  (36052, 30)
Shape of cv matrix after one hot encodig  (24155, 30)
```

In [53]:
```
# Vectorizing school state on train , test and cv

school_state_vectorizer = CountVectorizer(lowercase=False, binary=True)

school_state_vectorizer.fit(X_train['school_state'].values)
print(school_state_vectorizer.get_feature_names())

X_Train_school_state_one_hot = school_state_vectorizer.transform(X_trai
```

```
n['school_state'].values)
X_Test_school_state_one_hot = school_state_vectorizer.transform(X_test[
'school_state'].values)
X_CV_school_state_one_hot = school_state_vectorizer.transform(X_cv['sch
ool_state'].values)

print("Shape of train matrix after one hot encodig ",X_Train_school_sta
te_one_hot.shape)
print("Shape of test matrix after one hot encodig ",X_Test_school_state
_one_hot.shape)
print("Shape of cv matrix after one hot encodig ",X_CV_school_state_one
_hot.shape)

print("the type of count vectorizer ",type(X_Train_school_state_one_hot
))
print("the type of count vectorizer ",type(X_Test_school_state_one_hot
))
print("the type of count vectorizer ",type(X_CV_school_state_one_hot))
```

```
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'H
I', 'IA', 'ID', 'IL', 'IN', 'KS', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI',
'MN', 'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM', 'NV', 'NY',
'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT',
'WA', 'WI', 'WV', 'WY']
Shape of train matrix after one hot encodig  (49041, 51)
Shape of test matrix after one hot encodig  (36052, 51)
Shape of cv matrix after one hot encodig  (24155, 51)
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
```

In [54]:

```
# Vectorizing teacher prefix on train , test and cv

project_data["teacher_prefix"].fillna("No_Prefix", inplace = True)

teacher_prefix_vectorizer = CountVectorizer(lowercase=False, binary=Tru
e)
teacher_prefix_vectorizer.fit(X_train['teacher_prefix'].values)
```

```
print(teacher_prefix_vectorizer.get_feature_names())

X_Train_teacher_prefix_one_hot = teacher_prefix_vectorizer.transform(X_
train['teacher_prefix'].values)
X_Test_teacher_prefix_one_hot = teacher_prefix_vectorizer.transform(X_t
est['teacher_prefix'].values)
X_CV_teacher_prefix_one_hot = teacher_prefix_vectorizer.transform(X_cv[
'teacher_prefix'].values)

print("Shape of train matrix after one hot encodig ",X_Train_teacher_pr
efix_one_hot.shape)
print("Shape of test matrix after one hot encodig ",X_Test_teacher_pref
ix_one_hot.shape)
print("Shape of cv matrix after one hot encodig ",X_CV_teacher_prefix_o
ne_hot.shape)
```

```
['Dr', 'Mr', 'Mrs', 'Ms', 'No_Prefix', 'Teacher']
Shape of train matrix after one hot encodig  (49041, 6)
Shape of test matrix after one hot encodig  (36052, 6)
Shape of cv matrix after one hot encodig  (24155, 6)
```

In [55]:

```
# Vectorizing grade category on train , test and cv

my_grade_counter = Counter()

for project_grade in project_data['project_grade_category'].values:

    if (' ' in project_grade):

        project_grade = project_grade.replace(" ", "~")

    my_grade_counter.update(project_grade.split())

project_grade_cat_dict = dict(my_grade_counter)
sorted_project_grade_cat_dict = dict(sorted(project_grade_cat_dict.item
s(), key=lambda kv: kv[1]))

grade_cat_vectorizer = CountVectorizer(vocabulary=list(sorted_project_g
rade_cat_dict.keys()), lowercase=False, binary=True)
```

```
grade_cat_vectorizer.fit(X_train['project_grade_category'].values)
print(grade_cat_vectorizer.get_feature_names())

X_Train_grade_cat_one_hot = grade_cat_vectorizer.transform(X_train['pro
ject_grade_category'].values)
X_Test_grade_cat_one_hot = grade_cat_vectorizer.transform(X_test['proje
ct_grade_category'].values)
X_CV_grade_cat_one_hot = grade_cat_vectorizer.transform(X_cv['project_g
rade_category'].values)

print("Shape of train matrix after one hot encodig ",X_Train_grade_cat_
one_hot.shape)
print("Shape of test matrix after one hot encodig ",X_Test_grade_cat_on
e_hot.shape)
print("Shape of cv matrix after one hot encodig ",X_CV_grade_cat_one_ho
t.shape)
```

```
['Grades~9-12', 'Grades~6-8', 'Grades~3-5', 'Grades~PreK-2']
Shape of train matrix after one hot encodig  (49041, 4)
Shape of test matrix after one hot encodig  (36052, 4)
Shape of cv matrix after one hot encodig  (24155, 4)
```

2.3 Make Data Model Ready: encoding eassay, and project_title

In [56]:
```
# merge two column text dataframe:
X_train["essay"] = X_train["project_essay_1"].map(str) +\
                        X_train["project_essay_2"].map(str) + \
                        X_train["project_essay_3"].map(str) + \
                        X_train["project_essay_4"].map(str)
```

In [57]:
```
# preprocessing essay train data
from tqdm import tqdm
X_Train_preprocessed_essays = []
# tqdm is for printing the status bar
for sentence in tqdm(X_train['essay'].values):
    X_Train_essay_sent = decontracted(sentance)
```

```
    X_Train_essay_sent = X_Train_essay_sent.replace('\\r', ' ')
    X_Train_essay_sent = X_Train_essay_sent.replace('\\"', ' ')
    X_Train_essay_sent = X_Train_essay_sent.replace('\\n', ' ')
    X_Train_essay_sent = re.sub('[^A-Za-z0-9]+', ' ', X_Train_essay_sen
t)

    X_Train_essay_sent = ' '.join(e for e in X_Train_essay_sent.split()
 if e.lower() not in stopwords)
    X_Train_preprocessed_essays.append(X_Train_essay_sent.lower().strip
())
```

```
100%|████████████████████████████████████████████████████████████████
████████| 49041/49041 [01:01<00:00, 794.60it/s]
```

In [58]:
```
# preprocessing essay test data
from tqdm import tqdm
X_Test_preprocessed_essays = []
# tqdm is for printing the status bar
for sentence in tqdm(X_test['essay'].values):
    X_Test_essay_sent = decontracted(sentence)
    X_Test_essay_sent = X_Test_essay_sent.replace('\\r', ' ')
    X_Test_essay_sent = X_Test_essay_sent.replace('\\"', ' ')
    X_Test_essay_sent = X_Test_essay_sent.replace('\\n', ' ')
    X_Test_essay_sent = re.sub('[^A-Za-z0-9]+', ' ', X_Test_essay_sent)

    X_Test_essay_sent = ' '.join(e for e in X_Test_essay_sent.split() i
f e.lower() not in stopwords)
    X_Test_preprocessed_essays.append(X_Test_essay_sent.lower().strip
())
```

```
100%|████████████████████████████████████████████████████████████████
████████| 36052/36052 [00:43<00:00, 824.25it/s]
```

In [59]:
```
# preprocessing essay cv data
from tqdm import tqdm
X_CV_preprocessed_essays = []
# tqdm is for printing the status bar
for sentence in tqdm(X_cv['essay'].values):
    X_CV_essay_sent = decontracted(sentence)
```

```
    X_CV_essay_sent = X_CV_essay_sent.replace('\\r', ' ')
    X_CV_essay_sent = X_CV_essay_sent.replace('\\"', ' ')
    X_CV_essay_sent = X_CV_essay_sent.replace('\\n', ' ')
    X_CV_essay_sent = re.sub('[^A-Za-z0-9]+', ' ', X_CV_essay_sent)

    X_CV_essay_sent = ' '.join(e for e in X_CV_essay_sent.split() if e.
lower() not in stopwords)
    X_CV_preprocessed_essays.append(X_CV_essay_sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████
████████| 24155/24155 [00:28<00:00, 835.77it/s]
```

In [60]:

```
# Standardizing price train test and cv data

from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

normalizer.fit(X_train['essay_word_count'].values.reshape(-1,1))

essay_word_count_train = normalizer.transform(X_train['essay_word_coun
t'].values.reshape(-1,1))
essay_word_count_cv = normalizer.transform(X_cv['essay_word_count'].val
ues.reshape(-1,1))
essay_word_count_test = normalizer.transform(X_test['essay_word_count']
.values.reshape(-1,1))

print("After vectorizations")
print(essay_word_count_train.shape, y_train.shape)
print(essay_word_count_cv.shape, y_cv.shape)
print(essay_word_count_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
====================================================================
============================
```

In [61]:
```
normalizer = Normalizer()

normalizer.fit(X_train['pos'].values.reshape(-1,1))

essay_sent_pos_train = normalizer.transform(X_train['pos'].values.resha
pe(-1,1))
essay_sent_pos_cv = normalizer.transform(X_cv['pos'].values.reshape(-1,
1))
essay_sent_pos_test = normalizer.transform(X_test['pos'].values.reshape
(-1,1))

print("After vectorizations")
print(essay_sent_pos_train.shape, y_train.shape)
print(essay_sent_pos_cv.shape, y_cv.shape)
print(essay_sent_pos_test.shape, y_test.shape)
print("="*100)

normalizer.fit(X_train['neg'].values.reshape(-1,1))

essay_sent_neg_train = normalizer.transform(X_train['neg'].values.resha
pe(-1,1))
essay_sent_neg_cv = normalizer.transform(X_cv['neg'].values.reshape(-1,
1))
essay_sent_neg_test = normalizer.transform(X_test['neg'].values.reshape
(-1,1))

print("After vectorizations")
print(essay_sent_neg_train.shape, y_train.shape)
print(essay_sent_neg_cv.shape, y_cv.shape)
print(essay_sent_neg_test.shape, y_test.shape)
print("="*100)

normalizer.fit(X_train['neu'].values.reshape(-1,1))

essay_sent_neu_train = normalizer.transform(X_train['neu'].values.resha
pe(-1,1))
essay_sent_neu_cv = normalizer.transform(X_cv['neu'].values.reshape(-1,
1))
essay_sent_neu_test = normalizer.transform(X_test['neu'].values.reshape
```

```
(-1,1))

print("After vectorizations")
print(essay_sent_neu_train.shape, y_train.shape)
print(essay_sent_neu_cv.shape, y_cv.shape)
print(essay_sent_neu_test.shape, y_test.shape)
print("="*100)

normalizer.fit(X_train['compound'].values.reshape(-1,1))

essay_sent_comp_train = normalizer.transform(X_train['compound'].values
.reshape(-1,1))
essay_sent_comp_cv = normalizer.transform(X_cv['compound'].values.resha
pe(-1,1))
essay_sent_comp_test = normalizer.transform(X_test['compound'].values.r
eshape(-1,1))

print("After vectorizations")
print(essay_sent_comp_train.shape, y_train.shape)
print(essay_sent_comp_cv.shape, y_cv.shape)
print(essay_sent_comp_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
====================================================================
============================
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
====================================================================
============================
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
====================================================================
```

```
==============================
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
==================================================================================
==============================
```

In [62]:

```
# preprocessing project title train data
X_Train_preprocessed_titles = []

for dataset in tqdm(X_train['project_title'].values):
    data = decontracted(dataset)  # Replacing some specific and general
 short form into proper word/stopword.
    data = re.sub(r"it's", "it is", data) # Replacing it's with it is a
s it is not part of function decontracted
    data = data.replace('\\r', ' ') # Replacing \r with space
    data = data.replace('\\"', ' ') # Replacing \ with space
    data = data.replace('\\n', ' ') # Replacing \n with space
    data = re.sub('[^A-Za-z0-9]+', ' ', data) # Replacing special chara
cters with space
    data = re.sub("\S*\d\S*", "", data).strip() # Trimming numbers cont
aining digits

    data = ' '.join(e for e in data.split() if e not in stopwords) # Re
moving stopwords
    X_Train_preprocessed_titles.append(data.lower().strip()) # Creating
 array in all the lower cases.
```

```
100%|████████████████████████████████████████████████████████████████
████████| 49041/49041 [00:03<00:00, 15284.86it/s]
```

In [63]:

```
# preprocessing project title test data
X_Test_preprocessed_titles = []

for dataset in tqdm(X_test['project_title'].values):
    data = decontracted(dataset)  # Replacing some specific and general
 short form into proper word/stopword.
    data = re.sub(r"it's", "it is", data) # Replacing it's with it is a
```

```
s it is not part of function decontracted
    data = data.replace('\\r', ' ') # Replacing \r with space
    data = data.replace('\\"', ' ') # Replacing \ with space
    data = data.replace('\\n', ' ') # Replacing \n with space
    data = re.sub('[^A-Za-z0-9]+', ' ', data) # Replacing special chara
cters with space
    data = re.sub("\S*\d\S*", "", data).strip() # Trimming numbers cont
aining digits

    data = ' '.join(e for e in data.split() if e not in stopwords) # Re
moving stopwords
    X_Test_preprocessed_titles.append(data.lower().strip()) # Creating
 array in all the lower cases.
```

```
100%|████████████████████████████████████████████████████████████████
████████| 36052/36052 [00:02<00:00, 15377.47it/s]
```

In [64]:

```
# preprocessing project title cv data
X_CV_preprocessed_titles = []

for dataset in tqdm(X_cv['project_title'].values):
    data = decontracted(dataset)  # Replacing some specific and general
 short form into proper word/stopword.
    data = re.sub(r"it's", "it is", data) # Replacing it's with it is a
s it is not part of function decontracted
    data = data.replace('\\r', ' ') # Replacing \r with space
    data = data.replace('\\"', ' ') # Replacing \ with space
    data = data.replace('\\n', ' ') # Replacing \n with space
    data = re.sub('[^A-Za-z0-9]+', ' ', data) # Replacing special chara
cters with space
    data = re.sub("\S*\d\S*", "", data).strip() # Trimming numbers cont
aining digits

    data = ' '.join(e for e in data.split() if e not in stopwords) # Re
moving stopwords
    X_CV_preprocessed_titles.append(data.lower().strip()) # Creating ar
ray in all the lower cases.
```

```
100%|████████████████████████████████████████████████████████████████
████████| 24155/24155 [00:01<00:00, 15913.00it/s]
```

In [65]:
```
normalizer = Normalizer()

normalizer.fit(X_train['title_word_count'].values.reshape(-1,1))

title_word_count_train = normalizer.transform(X_train['title_word_coun
t'].values.reshape(-1,1))
title_word_count_cv = normalizer.transform(X_cv['title_word_count'].val
ues.reshape(-1,1))
title_word_count_test = normalizer.transform(X_test['title_word_count']
.values.reshape(-1,1))

print("After vectorizations")
print(title_word_count_train.shape, y_train.shape)
print(title_word_count_cv.shape, y_cv.shape)
print(title_word_count_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
====================================================================
============================
```

In [66]:
```
# BOW Essay train, test and cv data
# We are considering only the words which appeared in at least 10 docum
ents(rows or projects).
vectorizer = CountVectorizer(ngram_range=(2,2), min_df=10, max_features
 = 5000)
vectorizer.fit(X_Train_preprocessed_essays)

X_Train_essay_bow = vectorizer.transform(X_Train_preprocessed_essays)
X_Test_essay_bow = vectorizer.transform(X_Test_preprocessed_essays)
X_CV_essay_bow = vectorizer.transform(X_CV_preprocessed_essays)

print("Shape of train matrix after one hot encodig ",X_Train_essay_bow.
shape)
print("Shape of test matrix after one hot encodig ",X_Test_essay_bow.sh
```

```
ape)
print("Shape of CV matrix after one hot encodig ",X_CV_essay_bow.shape)
```

```
Shape of train matrix after one hot encodig  (49041, 147)
Shape of test matrix after one hot encodig  (36052, 147)
Shape of CV matrix after one hot encodig  (24155, 147)
```

In [67]:
```
# BOW title train,test and cv data

titles_vectorizer = CountVectorizer(ngram_range=(2,2), min_df=10, max_f
eatures = 5000)
titles_vectorizer.fit(X_Train_preprocessed_titles)

X_Train_titles_bow = titles_vectorizer.transform(X_Train_preprocessed_t
itles)
X_Test_titles_bow = titles_vectorizer.transform(X_Test_preprocessed_tit
les)
X_CV_titles_bow = titles_vectorizer.transform(X_CV_preprocessed_titles)

print("some sample features(unique words in the corpus)",titles_vectori
zer.get_feature_names()[0:10])
print("Shape of train matrix after one hot encodig ",X_Train_titles_bow
.shape)
print("Shape of test matrix after one hot encodig ",X_Test_titles_bow.s
hape)
print("Shape of CV matrix after one hot encodig ",X_CV_titles_bow.shape
)
```

```
some sample features(unique words in the corpus) ['about it', 'about ou
r', 'about reading', 'about science', 'about that', 'about the', 'acade
mic success', 'access technology', 'achievement gap', 'active bodies']
Shape of train matrix after one hot encodig  (49041, 1667)
Shape of test matrix after one hot encodig  (36052, 1667)
Shape of CV matrix after one hot encodig  (24155, 1667)
```

In [68]:
```
#TFIDF essay train,test and cv data

vectorizer = TfidfVectorizer(ngram_range=(2,2), min_df=10, max_features
```

```
 = 5000)
vectorizer.fit(X_Train_preprocessed_essays)

X_Train_essay_tfidf = vectorizer.transform(X_Train_preprocessed_essays)
X_Test_essay_tfidf = vectorizer.transform(X_Test_preprocessed_essays)
X_CV_essay_tfidf = vectorizer.transform(X_CV_preprocessed_essays)

print("Shape of train matrix after one hot encodig ",X_Train_essay_tfid
f.shape)
print("Shape of test matrix after one hot encodig ",X_Test_essay_tfidf.
shape)
print("Shape of CV matrix after one hot encodig ",X_CV_essay_tfidf.shap
e)
```

```
Shape of train matrix after one hot encodig  (49041, 147)
Shape of test matrix after one hot encodig  (36052, 147)
Shape of CV matrix after one hot encodig  (24155, 147)
```

In [69]:

```
# TFIDF on project titles train,test and cv data

titles_tfidf_vectorizer = TfidfVectorizer(ngram_range=(2,2), min_df=10,
 max_features = 5000)
titles_tfidf_vectorizer.fit(X_Train_preprocessed_titles)

X_Train_titles_tfidf = titles_vectorizer.transform(X_Train_preprocessed
_titles)
X_Test_titles_tfidf = titles_vectorizer.transform(X_Test_preprocessed_t
itles)
X_CV_titles_tfidf = titles_vectorizer.transform(X_CV_preprocessed_title
s)

print("Shape of train matrix after one hot encodig ",X_Train_titles_tfi
df.shape)
print("Shape of test matrix after one hot encodig ",X_Test_titles_tfidf
.shape)
print("Shape of CV matrix after one hot encodig ",X_CV_titles_tfidf.sha
pe)
```

```
Shape of train matrix after one hot encodig  (49041, 1667)
Shape of test matrix after one hot encodig  (36052, 1667)
```

```
Shape of CV matrix after one hot encodig  (24155, 1667)
```

In [70]:
```
# average Word2Vec essay on train
# compute average word2vec for each review.
X_Train_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is
 stored in this list
for sentence in tqdm(X_Train_preprocessed_essays): # for each review/se
ntence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    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
    X_Train_avg_w2v_vectors.append(vector)

print(len(X_Train_avg_w2v_vectors))
print(len(X_Train_avg_w2v_vectors[0]))
```

```
100%|████████████████████████████████████████████████████████████████
████████| 49041/49041 [00:25<00:00, 1914.26it/s]

49041
300
```

In [71]:
```
# average Word2Vec essay on test
# compute average word2vec for each review.
X_Test_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is
 stored in this list
for sentence in tqdm(X_Test_preprocessed_essays): # for each review/sen
tence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    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
    X_Test_avg_w2v_vectors.append(vector)

print(len(X_Test_avg_w2v_vectors))
print(len(X_Test_avg_w2v_vectors[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████| 36052/36052 [00:19<00:00, 1838.63it/s]
```

```
36052
300
```

In [72]:

```
# average Word2Vec essay on cv
# compute average word2vec for each review.
X_CV_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is st
ored in this list
for sentence in tqdm(X_CV_preprocessed_essays): # for each review/sente
nce
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    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
    X_CV_avg_w2v_vectors.append(vector)

print(len(X_CV_avg_w2v_vectors))
print(len(X_CV_avg_w2v_vectors[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████| 24155/24155 [00:13<00:00, 1832.79it/s]
```

```
24155
300
```

In [73]:
```
# AVG W2V on project title train
X_Train_avg_w2v_titles_vectors = [];

for sentence in tqdm(X_Train_preprocessed_titles):

    vector_titles = np.zeros(300)
    cnt_words_titles = 0;

    for word in sentence.split():

        if word in glove_words:

            vector += model[word]
            cnt_words_titles += 1

    if cnt_words_titles != 0:

        vector_titles /= cnt_words_titles

    X_Train_avg_w2v_titles_vectors.append(vector_titles)

print(len(X_Train_avg_w2v_titles_vectors))
print(len(X_Train_avg_w2v_titles_vectors[0]))
```

```
100%|██████████| 49041/49041 [00:01<00:00, 34438.24it/s]
```

```
49041
300
```

In [74]:
```
# AVG W2V on project title test
X_Test_avg_w2v_titles_vectors = [];

for sentence in tqdm(X_Test_preprocessed_titles):

    vector_titles = np.zeros(300)
    cnt_words_titles = 0;

    for word in sentence.split():
```

```
        if word in glove_words:

            vector += model[word]
            cnt_words_titles += 1

    if cnt_words_titles != 0:

        vector_titles /= cnt_words_titles

    X_Test_avg_w2v_titles_vectors.append(vector_titles)

print(len(X_Test_avg_w2v_titles_vectors))
print(len(X_Test_avg_w2v_titles_vectors[0]))
```

```
100%|██████████| 36052/36052 [00:00<00:00, 36087.15it/s]
```

```
36052
300
```

In [75]:

```
# AVG W2V on project title cv
X_CV_avg_w2v_titles_vectors = [];

for sentence in tqdm(X_CV_preprocessed_titles):

    vector_titles = np.zeros(300)
    cnt_words_titles = 0;

    for word in sentence.split():

        if word in glove_words:

            vector += model[word]
            cnt_words_titles += 1

    if cnt_words_titles != 0:

        vector_titles /= cnt_words_titles
```

```
    X_CV_avg_w2v_titles_vectors.append(vector_titles)

print(len(X_CV_avg_w2v_titles_vectors))
print(len(X_CV_avg_w2v_titles_vectors[0]))
```

```
100%|██████████| 24155/24155 [00:00<00:00, 37808.03it/s]
```

```
24155
300
```

In [76]:
```
# TFIDF W2V
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_Train_preprocessed_essays)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model
.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [77]:
```
# TFIDF w2v essay train
# compute average word2vec for each review.
X_Train_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review
 is stored in this list
for sentence in tqdm(X_Train_preprocessed_essays): # for each review/se
ntence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentenc
e/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 t
he tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentenc
e.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
```

```
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_Train_tfidf_w2v_vectors.append(vector)

print(len(X_Train_tfidf_w2v_vectors))
print(len(X_Train_tfidf_w2v_vectors[0]))
```

```
100%|██████████| 49041/49041 [03:10<00:00, 257.66it/s]
```

```
49041
300
```

In [78]:

```
# TFIDF w2v essay test
# compute average word2vec for each review.
X_Test_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review i
s stored in this list
for sentence in tqdm(X_Test_preprocessed_essays): # for each review/sen
tence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentenc
e/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 t
he tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentenc
e.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_Test_tfidf_w2v_vectors.append(vector)

print(len(X_Test_tfidf_w2v_vectors))
print(len(X_Test_tfidf_w2v_vectors[0]))
```

```
100%|██████████| 36052/36052 [00:30<00:00, 1167.77it/s]
```

```
36052

300
```

In [79]:
```
# TFIDF w2v essay cv
# compute average word2vec for each review.
X_CV_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is
 stored in this list
for sentence in tqdm(X_CV_preprocessed_essays): # for each review/sente
nce
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentenc
e/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 t
he tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentenc
e.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_CV_tfidf_w2v_vectors.append(vector)

print(len(X_CV_tfidf_w2v_vectors))
print(len(X_CV_tfidf_w2v_vectors[0]))
```

```
100%|████████████████████████████████████████████████████████████████
████████| 24155/24155 [00:20<00:00, 1175.67it/s]

24155
300
```

In [80]:
```
# TFIDF weighted W2V on project_title
titles_tfidf_model = TfidfVectorizer()
titles_tfidf_model.fit(X_Train_preprocessed_titles)
titles_dictionary = dict(zip(titles_tfidf_model.get_feature_names(), li
```

```
st(titles_tfidf_model.idf_)))
titles_tfidf_words = set(titles_tfidf_model.get_feature_names())
```

In [81]:
```
# TFIDF w2v title train
X_Train_titles_tfidf_w2v_vectors = [];

for titles_sentence in tqdm(X_Train_preprocessed_titles):

    titles_vector = np.zeros(300)
    titles_tfidf_weight = 0;

    for word in titles_sentence.split():

        if (word in glove_words) and (word in titles_tfidf_words):

            titles_vec = model[word]

            titles_tf_idf = titles_dictionary[word]*(titles_sentence.co
unt(word)/len(titles_sentence.split()))
            titles_vector += (titles_vec * titles_tf_idf)
            titles_tfidf_weight += titles_tf_idf

    if titles_tfidf_weight != 0:

        titles_vector /= titles_tfidf_weight

    X_Train_titles_tfidf_w2v_vectors.append(titles_vector)

print(len(X_Train_titles_tfidf_w2v_vectors))
print(len(X_Train_titles_tfidf_w2v_vectors[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 49041/49041 [00:02<00:00, 17139.62it/s]
```

```
49041
300
```

In [82]:
```
# TFIDF w2v title train
X_Test_titles_tfidf_w2v_vectors = [];
```

```
for titles_sentence in tqdm(X_Test_preprocessed_titles):

    titles_vector = np.zeros(300)
    titles_tfidf_weight = 0;

    for word in titles_sentence.split():

        if (word in glove_words) and (word in titles_tfidf_words):

            titles_vec = model[word]

            titles_tf_idf = titles_dictionary[word]*(titles_sentence.co
unt(word)/len(titles_sentence.split()))
            titles_vector += (titles_vec * titles_tf_idf)
            titles_tfidf_weight += titles_tf_idf

    if titles_tfidf_weight != 0:

        titles_vector /= titles_tfidf_weight

    X_Test_titles_tfidf_w2v_vectors.append(titles_vector)

print(len(X_Test_titles_tfidf_w2v_vectors))
print(len(X_Test_titles_tfidf_w2v_vectors[0]))
```

```
100%|████████████████████████████████████████████████████████████
████████| 36052/36052 [00:02<00:00, 16600.56it/s]
```

```
36052
300
```

In [83]:

```
# TFIDF w2v title cv
X_CV_titles_tfidf_w2v_vectors = [];

for titles_sentence in tqdm(X_CV_preprocessed_titles):

    titles_vector = np.zeros(300)
    titles_tfidf_weight = 0;
```

```
    for word in titles_sentence.split():

        if (word in glove_words) and (word in titles_tfidf_words):

            titles_vec = model[word]

            titles_tf_idf = titles_dictionary[word]*(titles_sentence.co
unt(word)/len(titles_sentence.split()))
            titles_vector += (titles_vec * titles_tf_idf)
            titles_tfidf_weight += titles_tf_idf

    if titles_tfidf_weight != 0:

        titles_vector /= titles_tfidf_weight

    X_CV_titles_tfidf_w2v_vectors.append(titles_vector)

print(len(X_CV_titles_tfidf_w2v_vectors))
print(len(X_CV_titles_tfidf_w2v_vectors[0]))
```

```
100%|████████████████████████████████████████████████████████████| 24155/24155 [00:01<00:00, 17339.35it/s]
```

```
24155
300
```

In [86]:
```
# Vectorizing numerical feature

# Merging price data with train, test and cv
X_train = pd.merge(X_train, price_data, on='id', how='left')
X_test = pd.merge(X_test, price_data, on='id', how='left')
X_cv = pd.merge(X_cv, price_data, on='id', how='left')
```

In [87]:
```
# Standardizing price train test and cv data

from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
```

```
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['price'].values.reshape(-1,1))

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

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

```
After vectorizations
(49041, 1) (49041,)
(36052, 1) (36052,)
(24155, 1) (24155,)
====================================================================
============================
```

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

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

Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (BOW)

In [129]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/40840
```

```
39
from scipy.sparse import hstack

# Train data stack
X_tr = hstack((X_Train_categories_one_hot,X_Train_sub_categories_one_ho
t,X_Train_school_state_one_hot,
               X_Train_teacher_prefix_one_hot,X_Train_grade_cat_one_hot
,X_Train_essay_bow,X_Train_titles_bow,
               X_train_price_norm)).tocsr()

# CV data Stack
X_cr = hstack((X_CV_categories_one_hot,X_CV_sub_categories_one_hot,X_CV
_school_state_one_hot,
               X_CV_teacher_prefix_one_hot,X_CV_grade_cat_one_hot,X_CV_
essay_bow,X_CV_titles_bow,
               X_cv_price_norm)).tocsr()

# Test Data Stack
X_te = hstack((X_Test_categories_one_hot,X_Test_sub_categories_one_hot,
X_Test_school_state_one_hot,
               X_Test_teacher_prefix_one_hot,X_Test_grade_cat_one_hot,X
_Test_essay_bow,X_Test_titles_bow,
               X_test_price_norm)).tocsr()

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(49041, 1915) (49041,)
(24155, 1915) (24155,)
(36052, 1915) (36052,)
====================================================================
============================
```

In [130]:
```
def batch_predict(clf, data):
```

```
    y_data_pred = []

    # Changing the shape of predicted data in the multiple of 1000
    tr_loop = data.shape[0] - data.shape[0]%1000

    # Running the loop for each 1000th data
    for i in range(0, tr_loop, 1000):

        # Predicting probability
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])

    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [131]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics.classification import accuracy_score, log_loss

alpha = [10 ** x for x in range(-4, 5)]

reg = ['l1', 'l2']

for i in reg:

    cv_log_error_array = []

    for j in alpha:

        print("for C =", j )
        print ("for regularization = ", i)

        clf = SGDClassifier( class_weight='balanced', alpha=j, penalty=
i, loss='hinge', random_state=0)
        clf.fit(X_tr, y_train)

        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(X_tr, y_train)
```

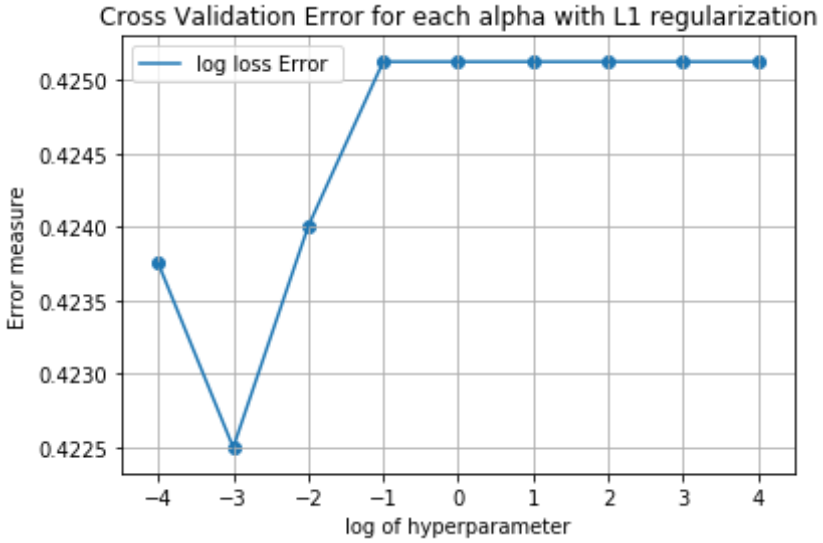
```
        sig_clf_probs = sig_clf.predict_proba(X_cr)


        cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=
clf.classes_, eps=1e-15))

        print("Log Loss :",log_loss(y_cv, sig_clf_probs))


    if (i == 'l1'):

        l1_cv_log_error = cv_log_error_array

    if (i == 'l2'):

        l2_cv_log_error = cv_log_error_array
```

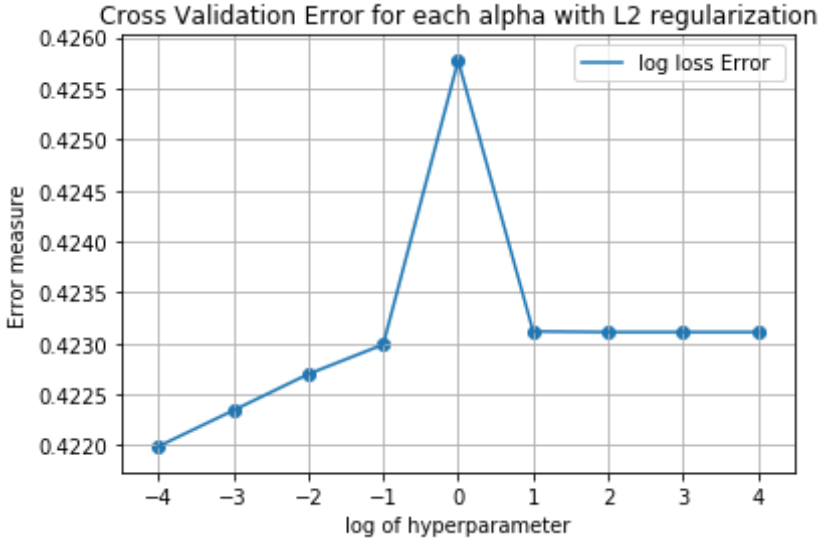
```
for C = 0.0001
for regularization =  l1
Log Loss : 0.423753079236
for C = 0.001
for regularization =  l1
Log Loss : 0.422500487331
for C = 0.01
for regularization =  l1
Log Loss : 0.424004925041
for C = 0.1
for regularization =  l1
Log Loss : 0.425125084428
for C = 1
for regularization =  l1
Log Loss : 0.425125084428
for C = 10
for regularization =  l1
Log Loss : 0.425125084428
for C = 100
for regularization =  l1
Log Loss : 0.425125084428
```

```
for C = 1000
for regularization =  l1
Log Loss : 0.425125084428
for C = 10000
for regularization =  l1
Log Loss : 0.425125084428
for C = 0.0001
for regularization =  l2
Log Loss : 0.421989821454
for C = 0.001
for regularization =  l2
Log Loss : 0.422342963142
for C = 0.01
for regularization =  l2
Log Loss : 0.42269884247
for C = 0.1
for regularization =  l2
Log Loss : 0.422988969574
for C = 1
for regularization =  l2
Log Loss : 0.425772315897
for C = 10
for regularization =  l2
Log Loss : 0.42311772471
for C = 100
for regularization =  l2
Log Loss : 0.423111347777
for C = 1000
for regularization =  l2
Log Loss : 0.423111347778
for C = 10000
for regularization =  l2
Log Loss : 0.423111347787
```

In [132]:
```
import math


# plotting obtained values
```

```
r = [10 ** x for x in range(-4, 5)]

x1 = [math.log10(i) for i in r]

plt.plot(x1,l1_cv_log_error,label='log loss Error ')
plt.scatter(x1,l1_cv_log_error)
plt.legend()
plt.xlabel('log of hyperparameter')
plt.ylabel('Error measure')
plt.title('Cross Validation Error for each alpha with L1 regularizatio
n')
plt.grid()
plt.show()


x1 = [math.log10(i) for i in r]

plt.plot(x1,l2_cv_log_error,label='log loss Error ')
plt.scatter(x1,l2_cv_log_error)
plt.legend()
plt.xlabel('log of hyperparameter')
plt.ylabel('Error measure')
plt.title('Cross Validation Error for each alpha with L2 regularizatio
n')
plt.grid()
plt.show()
```





In [133]:
```
l1_best_alpha = np.argmin(l1_cv_log_error)
print("Best alpha with L1 regularization : ", alpha[l1_best_alpha])
print("Best L1 regularization Value: ", l1_cv_log_error[l1_best_alpha])
```

```
l2_best_alpha = np.argmin(l2_cv_log_error)
print("Best alpha with L2 regularization : ", alpha[l2_best_alpha])
print("Best L2 regularization Value: ", l2_cv_log_error[l2_best_alpha])
```

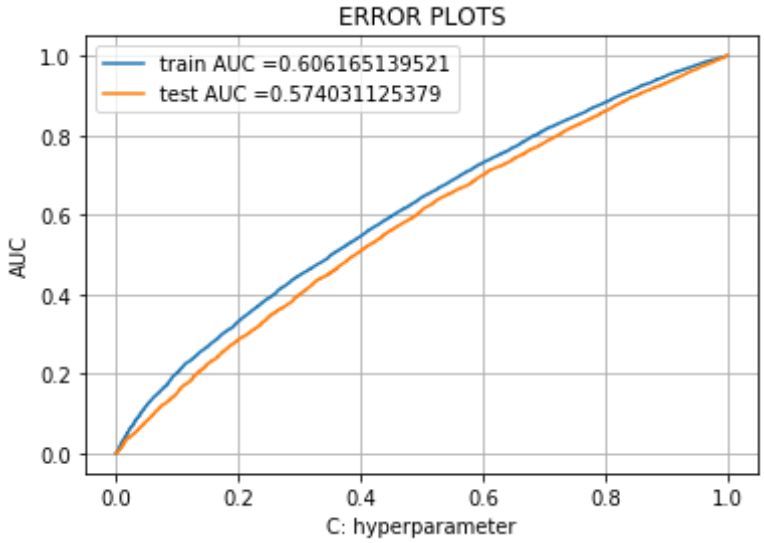
```
Best alpha with L1 regularization :  0.001
Best L1 regularization Value:  0.422500487331
Best alpha with L2 regularization :  0.0001
Best L2 regularization Value:  0.421989821454
```

In [134]:
```
# Since L2 has better regularization value so taken L2 penalty and its
 best alpha

clf = SGDClassifier(class_weight='balanced', alpha=alpha[l2_best_alpha
], penalty='l2', loss='hinge', random_state=0)
clf.fit(X_tr, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_tr, y_train)

predict_y = sig_clf.predict_proba(X_tr)
print('For values of best alpha = ', alpha[l2_best_alpha], "The train l
og loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-1
5))
predict_y = sig_clf.predict_proba(X_cr)
print('For values of best alpha = ', alpha[l2_best_alpha], "The cross v
alidation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_,
eps=1e-15))
predict_y = sig_clf.predict_proba(X_te)
print('For values of best alpha = ', alpha[l2_best_alpha], "The test lo
g loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15
))
```

```
For values of best alpha =  0.0001 The train log loss is: 0.41809514029
3
For values of best alpha =  0.0001 The cross validation log loss is: 0.
421989821454
For values of best alpha =  0.0001 The test log loss is: 0.421455938124
```

In [135]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc
```

```
_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

neigh = SGDClassifier(class_weight='balanced', alpha=alpha[l2_best_alph
a], penalty='l2', loss='hinge', random_state=0)
#neigh = LogisticRegression(C=0.5)
neigh.fit(X_tr, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(sig_clf, X_tr)
y_test_pred = batch_predict(sig_clf, X_te)

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, t
rain_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()
```



In [136]:
```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(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 th
reshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

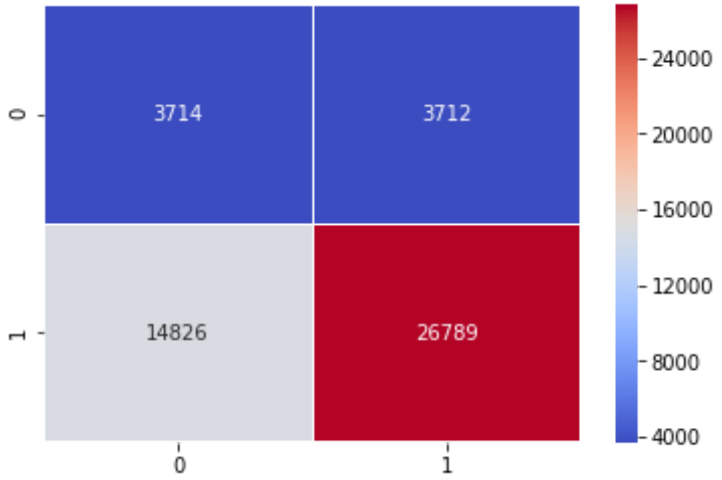
In [137]:
```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, tr
ain_fpr, train_fpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test
_fpr, test_fpr)))
```

```
====================================================================
============================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.249999981866 for threshold 0.842
[[ 3714  3712]
 [14826 26789]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.249999991611 for threshold 0.849
[[ 2550  2909]
 [10928 19665]]
```

In [138]:
```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html
# https://stackoverflow.com/questions/29647749/seaborn-showing-scientif
ic-notation-in-heatmap-for-3-digit-numbers

# Train Confusion Matrix Heatmap
train_confusion_matrix = confusion_matrix(y_train, predict(y_train_pred
, tr_thresholds, train_fpr, train_fpr))

print("Train Confusion Matrix")
sns.heatmap(train_confusion_matrix,annot=True,linewidth = 0.1, cmap='co
olwarm', fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.249999981866 for threshold 0.842
Train Confusion Matrix
```

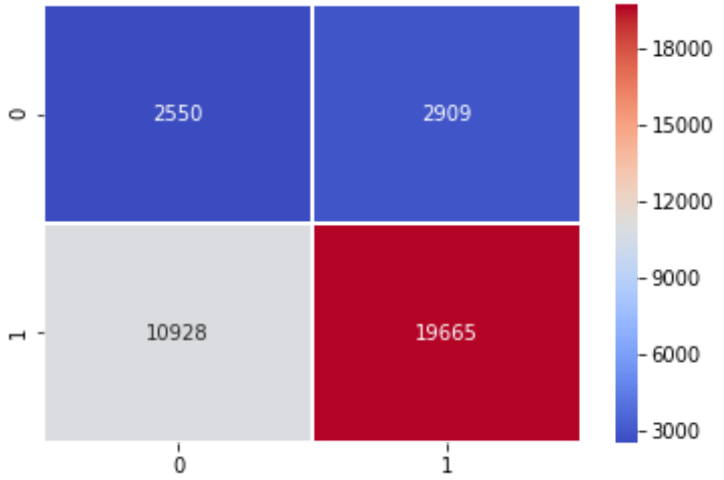
Out[138]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x16396cf8780>
```



In [139]:
```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html

# Test Confusion Matrix Heatmap

test_confusion_matrix = confusion_matrix(y_test, predict(y_test_pred, t
r_thresholds, test_fpr, test_fpr))

print("Test Confusion Matrix")
sns.heatmap(test_confusion_matrix,annot=True,linewidth = 0.5, cmap='coo
lwarm', fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.249999991611 for threshold 0.849
Test Confusion Matrix
```

Out[139]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x16396c85ba8>
```



Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)

In [140]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

# Train data stack
X_tr = hstack((X_Train_categories_one_hot,X_Train_sub_categories_one_hot,X_Train_school_state_one_hot,
               X_Train_teacher_prefix_one_hot,X_Train_grade_cat_one_hot,X_Train_essay_tfidf,X_Train_titles_tfidf,
               X_train_price_norm)).tocsr()

# CV data Stack
X_cr = hstack((X_CV_categories_one_hot,X_CV_sub_categories_one_hot,X_CV_school_state_one_hot,
               X_CV_teacher_prefix_one_hot,X_CV_grade_cat_one_hot,X_CV_essay_tfidf,X_CV_titles_tfidf,
               X_cv_price_norm)).tocsr()
```

```
# Test Data Stack
X_te = hstack((X_Test_categories_one_hot,X_Test_sub_categories_one_hot,
X_Test_school_state_one_hot,
               X_Test_teacher_prefix_one_hot,X_Test_grade_cat_one_hot,X
_Test_essay_tfidf,X_Test_titles_tfidf,
               X_test_price_norm)).tocsr()

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(49041, 1915) (49041,)
(24155, 1915) (24155,)
(36052, 1915) (36052,)
====================================================================
==============================
```

In [141]:

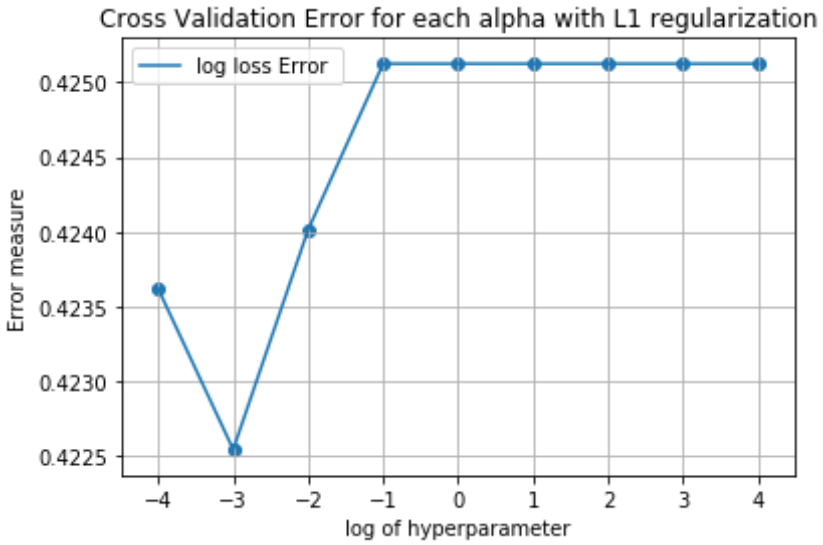
```
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics.classification import accuracy_score, log_loss

alpha = [10 ** x for x in range(-4, 5)]

reg = ['l1', 'l2']

for i in reg:

    cv_log_error_array = []

    for j in alpha:

        print("for C =", j )
        print ("for regularization = ", i)

        clf = SGDClassifier( class_weight='balanced', alpha=j, penalty=
i, loss='hinge', random_state=0)
```

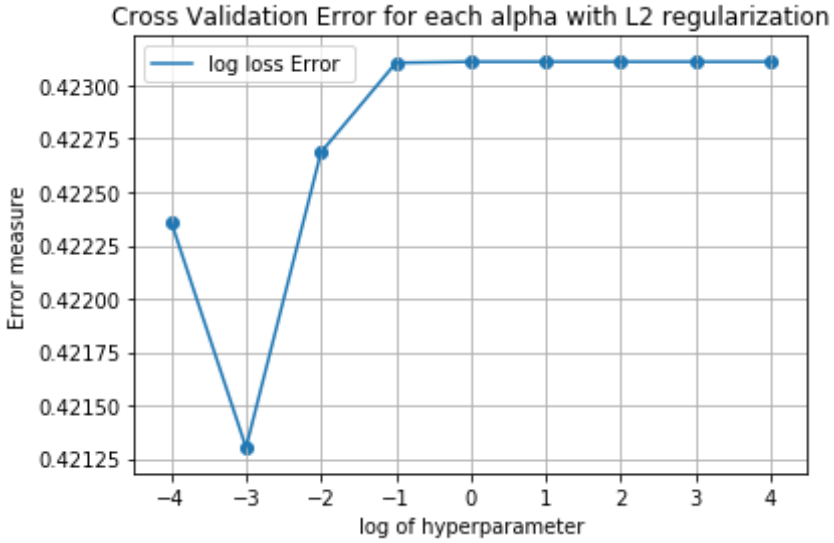
```
        clf.fit(X_tr, y_train)

        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(X_tr, y_train)

        sig_clf_probs = sig_clf.predict_proba(X_cr)


        cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=
clf.classes_, eps=1e-15))

        print("Log Loss :",log_loss(y_cv, sig_clf_probs))


    if (i == 'l1'):

        l1_cv_log_error = cv_log_error_array

    if (i == 'l2'):

        l2_cv_log_error = cv_log_error_array
```

```
for C = 0.0001
for regularization =  l1
Log Loss : 0.423618093402
for C = 0.001
for regularization =  l1
Log Loss : 0.422543769689
for C = 0.01
for regularization =  l1
Log Loss : 0.424015463556
for C = 0.1
for regularization =  l1
Log Loss : 0.425125084428
for C = 1
for regularization =  l1
Log Loss : 0.425125084428
for C = 10
```

```
for regularization =  l1
Log Loss : 0.425125084428
for C = 100
for regularization =  l1
Log Loss : 0.425125084428
for C = 1000
for regularization =  l1
Log Loss : 0.425125084428
for C = 10000
for regularization =  l1
Log Loss : 0.425125084428
for C = 0.0001
for regularization =  l2
Log Loss : 0.422362104214
for C = 0.001
for regularization =  l2
Log Loss : 0.421303156182
for C = 0.01
for regularization =  l2
Log Loss : 0.422685912836
for C = 0.1
for regularization =  l2
Log Loss : 0.423105341476
for C = 1
for regularization =  l2
Log Loss : 0.423110689051
for C = 10
for regularization =  l2
Log Loss : 0.423110730765
for C = 100
for regularization =  l2
Log Loss : 0.423110730765
for C = 1000
for regularization =  l2
Log Loss : 0.423110730766
for C = 10000
for regularization =  l2
Log Loss : 0.423110730776
```

In [142]:
```
r = [10 ** x for x in range(-4, 5)]

x1 = [math.log10(i) for i in r]

plt.plot(x1,l1_cv_log_error,label='log loss Error ')
plt.scatter(x1,l1_cv_log_error)
plt.legend()
plt.xlabel('log of hyperparameter')
plt.ylabel('Error measure')
plt.title('Cross Validation Error for each alpha with L1 regularizatio
n')
plt.grid()
plt.show()


x1 = [math.log10(i) for i in r]

plt.plot(x1,l2_cv_log_error,label='log loss Error ')
plt.scatter(x1,l2_cv_log_error)
plt.legend()
plt.xlabel('log of hyperparameter')
plt.ylabel('Error measure')
plt.title('Cross Validation Error for each alpha with L2 regularizatio
n')
plt.grid()
plt.show()
```





In [143]:
```
l1_best_alpha = np.argmin(l1_cv_log_error)
print("Best alpha with L1 regularization : ", alpha[l1_best_alpha])
```

```
print("Best L1 regularization Value: ", l1_cv_log_error[l1_best_alpha])

l2_best_alpha = np.argmin(l2_cv_log_error)
print("Best alpha with L2 regularization : ", alpha[l2_best_alpha])
print("Best L2 regularization Value: ", l2_cv_log_error[l2_best_alpha])
```

```
Best alpha with L1 regularization :  0.001
Best L1 regularization Value:  0.422543769689
Best alpha with L2 regularization :  0.001
Best L2 regularization Value:  0.421303156182
```

In [144]:

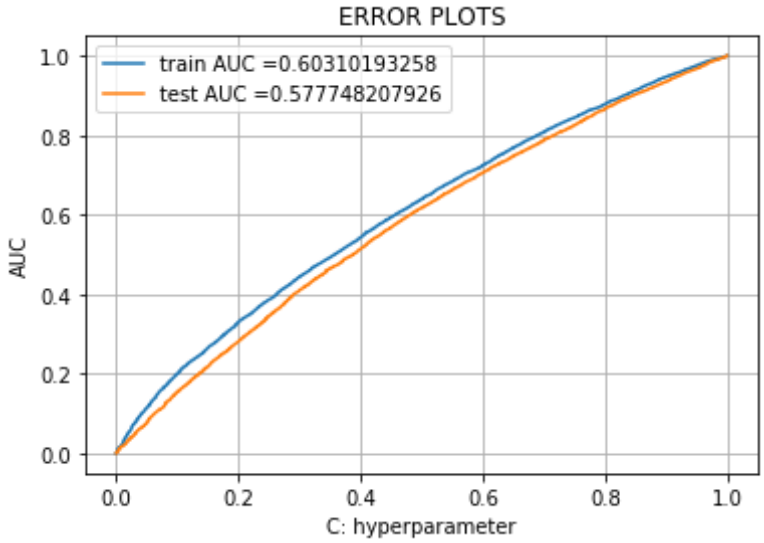
```
# Since L2 has better regularization value so taken L2 penalty and its
 best alpha

clf = SGDClassifier(class_weight='balanced', alpha=alpha[l2_best_alpha
], penalty='l2', loss='hinge', random_state=0)
clf.fit(X_tr, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_tr, y_train)

predict_y = sig_clf.predict_proba(X_tr)
print('For values of best alpha = ', alpha[l2_best_alpha], "The train l
og loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-1
5))
predict_y = sig_clf.predict_proba(X_cr)
print('For values of best alpha = ', alpha[l2_best_alpha], "The cross v
alidation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_,
eps=1e-15))
predict_y = sig_clf.predict_proba(X_te)
print('For values of best alpha = ', alpha[l2_best_alpha], "The test lo
g loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15
))
```

```
For values of best alpha =  0.001 The train log loss is: 0.417997846108
For values of best alpha =  0.001 The cross validation log loss is: 0.4
21303156182
For values of best alpha =  0.001 The test log loss is: 0.420728163891
```

In [145]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc
```

```
_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

neigh = SGDClassifier(class_weight='balanced', alpha=alpha[l2_best_alph
a], penalty='l2', loss='hinge', random_state=0)
#neigh = LogisticRegression(C=0.5)
neigh.fit(X_tr, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(sig_clf, X_tr)
y_test_pred = batch_predict(sig_clf, X_te)

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, t
rain_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()
```



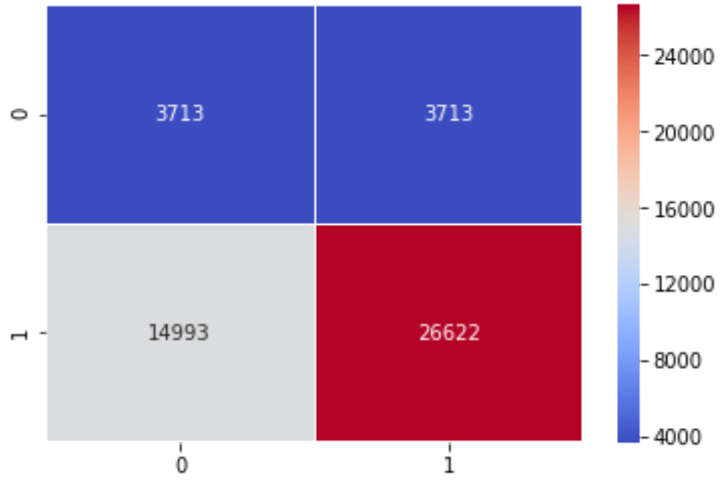
In [146]:
```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, tr
ain_fpr, train_fpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test
_fpr, test_fpr)))
```

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.841
[[ 3713  3713]
 [14993 26622]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.249999991611 for threshold 0.851
[[ 2972  2487]
 [13003 17590]]
```

In [147]:
```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html
# https://stackoverflow.com/questions/29647749/seaborn-showing-scientific-notation-in-heatmap-for-3-digit-numbers

# Train Confusion Matrix Heatmap
train_confusion_matrix = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))

print("Train Confusion Matrix")
sns.heatmap(train_confusion_matrix,annot=True,linewidth = 0.1, cmap='coolwarm', fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.841
Train Confusion Matrix
```

Out[147]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x163982c0c88>
```

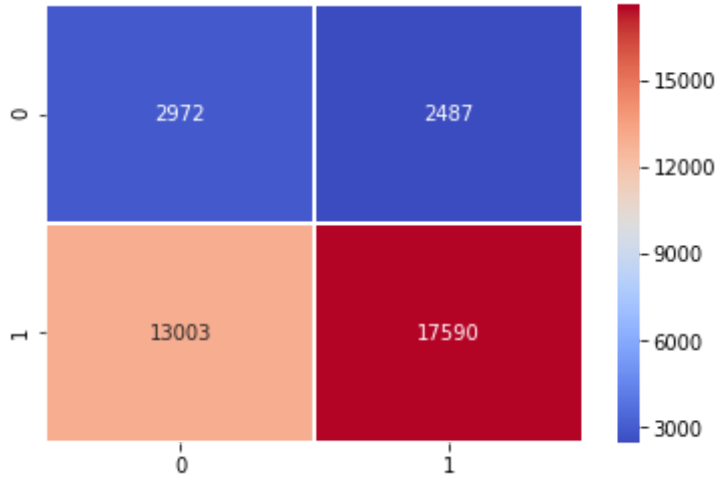


In [148]:
```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html

# Test Confusion Matrix Heatmap

test_confusion_matrix = confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr))
```

```
print("Test Confusion Matrix")
sns.heatmap(test_confusion_matrix,annot=True,linewidth = 0.5, cmap='coo
lwarm', fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.249999991611 for threshold 0.851
Test Confusion Matrix
```

Out[148]: <matplotlib.axes._subplots.AxesSubplot at 0x1639643a160>



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

In [149]:
```
# Train data stack
X_tr = hstack((X_Train_categories_one_hot,X_Train_sub_categories_one_ho
t,X_Train_school_state_one_hot,
               X_Train_teacher_prefix_one_hot,X_Train_grade_cat_one_hot
,X_Train_avg_w2v_vectors,X_Train_avg_w2v_titles_vectors,
               X_train_price_norm)).tocsr()

# CV data Stack
X_cr = hstack((X_CV_categories_one_hot,X_CV_sub_categories_one_hot,X_CV
_school_state_one_hot,
               X_CV_teacher_prefix_one_hot,X_CV_grade_cat_one_hot,X_CV_
```

```
avg_w2v_vectors,X_CV_avg_w2v_titles_vectors,
               X_cv_price_norm)).tocsr()

# Test Data Stack
X_te = hstack((X_Test_categories_one_hot,X_Test_sub_categories_one_hot,
X_Test_school_state_one_hot,
               X_Test_teacher_prefix_one_hot,X_Test_grade_cat_one_hot,X
_Test_avg_w2v_vectors,X_Test_avg_w2v_titles_vectors,
               X_test_price_norm)).tocsr()

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(49041, 701) (49041,)
(24155, 701) (24155,)
(36052, 701) (36052,)
====================================================================
============================
```

In [150]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics.classification import accuracy_score, log_loss

alpha = [10 ** x for x in range(-4, 5)]

reg = ['l1', 'l2']

for i in reg:

    cv_log_error_array = []

    for j in alpha:

        print("for C =", j )
        print ("for regularization = ", i)
```

```
        clf = SGDClassifier( class_weight='balanced', alpha=j, penalty=
i, loss='hinge', random_state=0)
        clf.fit(X_tr, y_train)

        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(X_tr, y_train)

        sig_clf_probs = sig_clf.predict_proba(X_cr)


        cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=
clf.classes_, eps=1e-15))

        print("Log Loss :",log_loss(y_cv, sig_clf_probs))


    if (i == 'l1'):

        l1_cv_log_error = cv_log_error_array

    if (i == 'l2'):

        l2_cv_log_error = cv_log_error_array
```

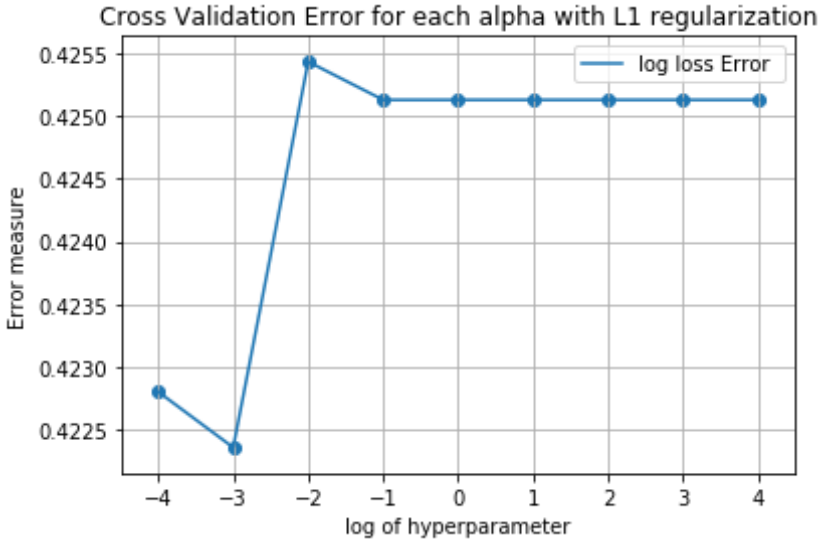
```
for C = 0.0001
for regularization =  l1
Log Loss : 0.42280452684
for C = 0.001
for regularization =  l1
Log Loss : 0.422362248267
for C = 0.01
for regularization =  l1
Log Loss : 0.425428545476
for C = 0.1
for regularization =  l1
Log Loss : 0.425125084428
for C = 1
```

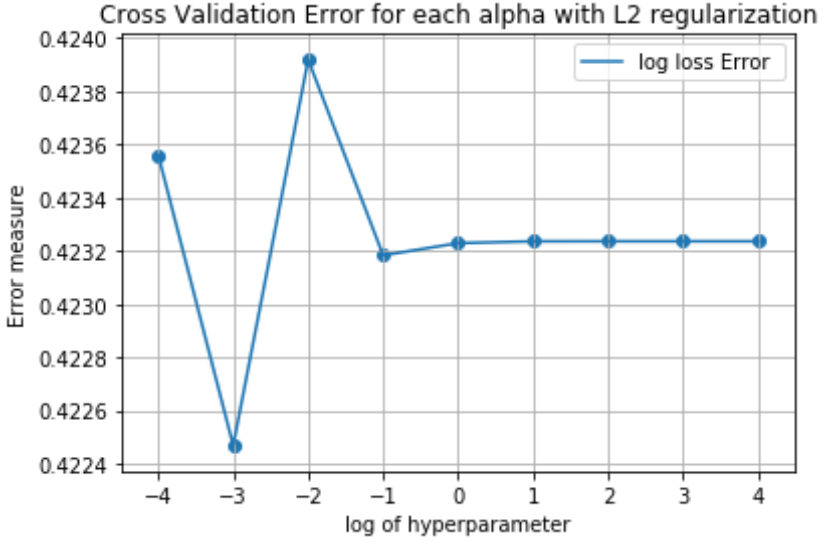
```
for regularization =  l1
Log Loss : 0.425125084428
for C = 10
for regularization =  l1
Log Loss : 0.425125084428
for C = 100
for regularization =  l1
Log Loss : 0.425125084428
for C = 1000
for regularization =  l1
Log Loss : 0.425125084428
for C = 10000
for regularization =  l1
Log Loss : 0.425125084428
for C = 0.0001
for regularization =  l2
Log Loss : 0.423560604079
for C = 0.001
for regularization =  l2
Log Loss : 0.422468333661
for C = 0.01
for regularization =  l2
Log Loss : 0.423918712218
for C = 0.1
for regularization =  l2
Log Loss : 0.423183219987
for C = 1
for regularization =  l2
Log Loss : 0.423228959245
for C = 10
for regularization =  l2
Log Loss : 0.423235962232
for C = 100
for regularization =  l2
Log Loss : 0.423236095742
for C = 1000
for regularization =  l2
Log Loss : 0.423236095744
for C = 10000
```

```
for regularization =  l2
Log Loss : 0.423236095753
```

In [151]:

```
r = [10 ** x for x in range(-4, 5)]

x1 = [math.log10(i) for i in r]

plt.plot(x1,l1_cv_log_error,label='log loss Error ')
plt.scatter(x1,l1_cv_log_error)
plt.legend()
plt.xlabel('log of hyperparameter')
plt.ylabel('Error measure')
plt.title('Cross Validation Error for each alpha with L1 regularizatio
n')
plt.grid()
plt.show()


x1 = [math.log10(i) for i in r]

plt.plot(x1,l2_cv_log_error,label='log loss Error ')
plt.scatter(x1,l2_cv_log_error)
plt.legend()
plt.xlabel('log of hyperparameter')
plt.ylabel('Error measure')
plt.title('Cross Validation Error for each alpha with L2 regularizatio
n')
plt.grid()
plt.show()
```





In [152]:
```
l1_best_alpha = np.argmin(l1_cv_log_error)
print("Best alpha with L1 regularization : ", alpha[l1_best_alpha])
print("Best L1 regularization Value: ", l1_cv_log_error[l1_best_alpha])
```

```
l2_best_alpha = np.argmin(l2_cv_log_error)
print("Best alpha with L2 regularization : ", alpha[l2_best_alpha])
print("Best L2 regularization Value: ", l2_cv_log_error[l2_best_alpha])
```

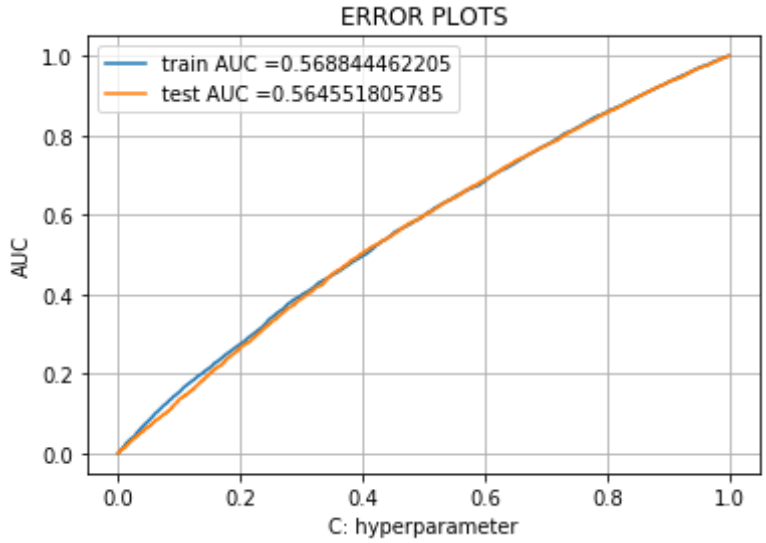
```
Best alpha with L1 regularization :  0.001
Best L1 regularization Value:  0.422362248267
Best alpha with L2 regularization :  0.001
Best L2 regularization Value:  0.422468333661
```

In [153]:
```
# Since L2 has better regularization value so taken L2 penalty and its
 best alpha

clf = SGDClassifier(class_weight='balanced', alpha=alpha[l2_best_alpha
], penalty='l2', loss='hinge', random_state=0)
clf.fit(X_tr, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_tr, y_train)

predict_y = sig_clf.predict_proba(X_tr)
print('For values of best alpha = ', alpha[l2_best_alpha], "The train l
og loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-1
5))
predict_y = sig_clf.predict_proba(X_cr)
print('For values of best alpha = ', alpha[l2_best_alpha], "The cross v
alidation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_,
eps=1e-15))
predict_y = sig_clf.predict_proba(X_te)
print('For values of best alpha = ', alpha[l2_best_alpha], "The test lo
g loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15
))
```

```
For values of best alpha =  0.001 The train log loss is: 0.421695318206
For values of best alpha =  0.001 The cross validation log loss is: 0.4
22468333661
For values of best alpha =  0.001 The test log loss is: 0.422080060527
```

In [154]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc
_curve.html#sklearn.metrics.roc_curve
```

```
from sklearn.metrics import roc_curve, auc

neigh = SGDClassifier(class_weight='balanced', alpha=alpha[l2_best_alph
a], penalty='l2', loss='hinge', random_state=0)
#neigh = LogisticRegression(C=0.5)
neigh.fit(X_tr, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(sig_clf, X_tr)
y_test_pred = batch_predict(sig_clf, X_te)

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, t
rain_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()
```



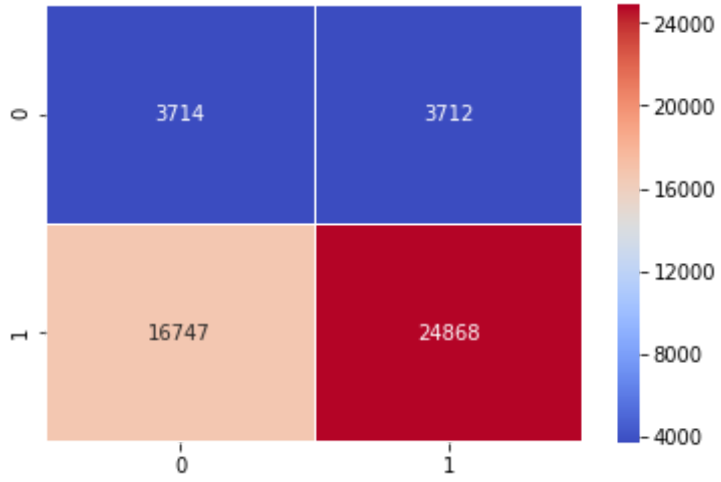
In [155]:
```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, tr
ain_fpr, train_fpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test
_fpr, test_fpr)))
```

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.249999981866 for threshold 0.845
[[ 3714  3712]
 [16747 24868]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.249999991611 for threshold 0.838
[[ 2151  3308]
 [ 9403 21190]]
```

In [156]:
```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html
# https://stackoverflow.com/questions/29647749/seaborn-showing-scientific-notation-in-heatmap-for-3-digit-numbers

# Train Confusion Matrix Heatmap
train_confusion_matrix = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))

print("Train Confusion Matrix")
sns.heatmap(train_confusion_matrix,annot=True,linewidth = 0.1, cmap='coolwarm', fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.249999981866 for threshold 0.845
Train Confusion Matrix
```

Out[156]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x163964b5cc0>
```

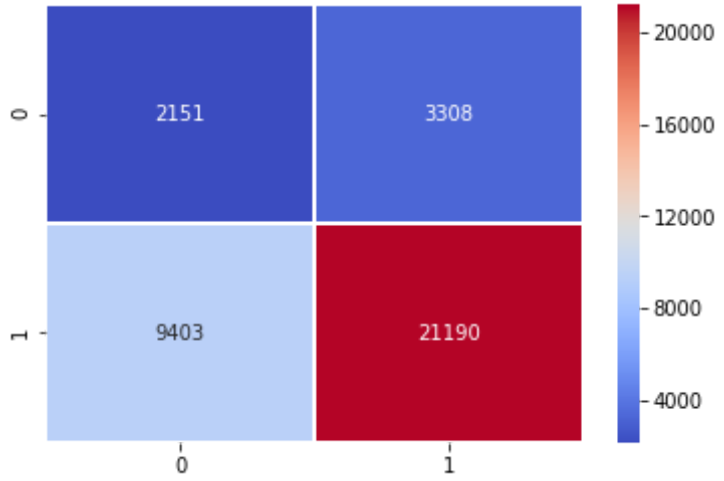


In [157]:
```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html

# Test Confusion Matrix Heatmap

test_confusion_matrix = confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr))
```

```
print("Test Confusion Matrix")
sns.heatmap(test_confusion_matrix,annot=True,linewidth = 0.5, cmap='coo
lwarm', fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.249999991611 for threshold 0.838
Test Confusion Matrix
```

Out[157]: <matplotlib.axes._subplots.AxesSubplot at 0x163964777b8>



Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)

In [158]:
```
# Train data stack
X_tr = hstack((X_Train_categories_one_hot,X_Train_sub_categories_one_ho
t,X_Train_school_state_one_hot,
               X_Train_teacher_prefix_one_hot,X_Train_grade_cat_one_hot
,X_Train_tfidf_w2v_vectors,X_Train_titles_tfidf_w2v_vectors,
               X_train_price_norm)).tocsr()

# CV data Stack
X_cr = hstack((X_CV_categories_one_hot,X_CV_sub_categories_one_hot,X_CV
_school_state_one_hot,
               X_CV_teacher_prefix_one_hot,X_CV_grade_cat_one_hot,X_CV_
```

```
tfidf_w2v_vectors,X_CV_titles_tfidf_w2v_vectors,
              X_cv_price_norm)).tocsr()

# Test Data Stack
X_te = hstack((X_Test_categories_one_hot,X_Test_sub_categories_one_hot,
X_Test_school_state_one_hot,
              X_Test_teacher_prefix_one_hot,X_Test_grade_cat_one_hot,X
_Test_tfidf_w2v_vectors,X_Test_titles_tfidf_w2v_vectors,
              X_test_price_norm)).tocsr()

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(49041, 701) (49041,)
(24155, 701) (24155,)
(36052, 701) (36052,)
====================================================================
============================
```

In [159]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics.classification import accuracy_score, log_loss

alpha = [10 ** x for x in range(-4, 5)]

reg = ['l1', 'l2']

for i in reg:

    cv_log_error_array = []

    for j in alpha:

        print("for C =", j )
        print ("for regularization = ", i)
```

```
        clf = SGDClassifier( class_weight='balanced', alpha=j, penalty=
i, loss='hinge', random_state=0)
        clf.fit(X_tr, y_train)

        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(X_tr, y_train)

        sig_clf_probs = sig_clf.predict_proba(X_cr)


        cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=
clf.classes_, eps=1e-15))

        print("Log Loss :",log_loss(y_cv, sig_clf_probs))


    if (i == 'l1'):

        l1_cv_log_error = cv_log_error_array

    if (i == 'l2'):

        l2_cv_log_error = cv_log_error_array
```

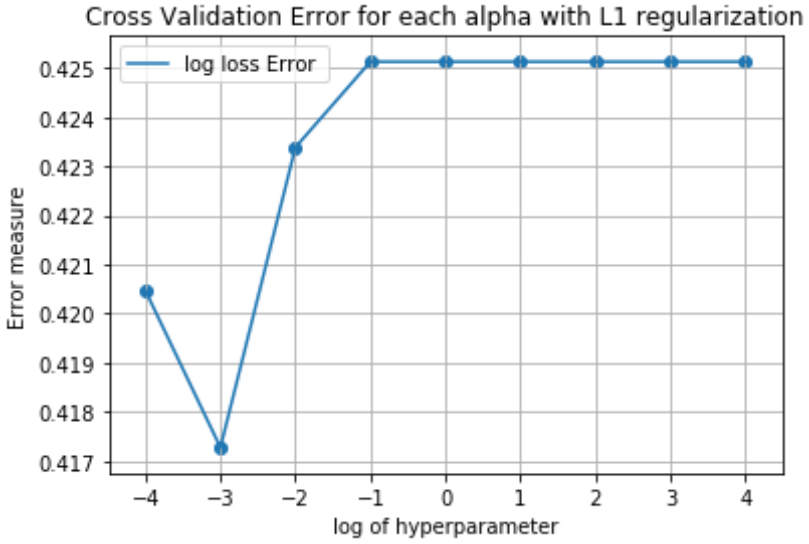
```
for C = 0.0001
for regularization =  l1
Log Loss : 0.420467039306
for C = 0.001
for regularization =  l1
Log Loss : 0.417277913841
for C = 0.01
for regularization =  l1
Log Loss : 0.423382775988
for C = 0.1
for regularization =  l1
Log Loss : 0.425125084428
for C = 1
```

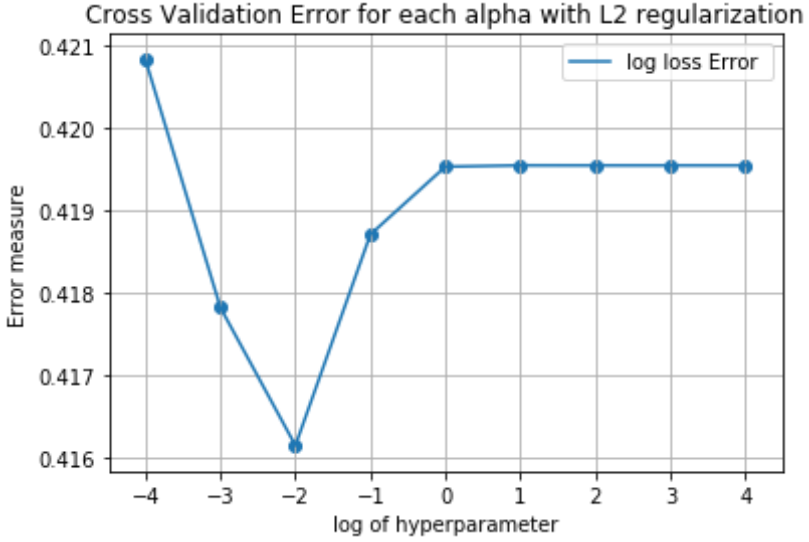
```
for regularization =  l1
Log Loss : 0.425125084428
for C = 10
for regularization =  l1
Log Loss : 0.425125084428
for C = 100
for regularization =  l1
Log Loss : 0.425125084428
for C = 1000
for regularization =  l1
Log Loss : 0.425125084428
for C = 10000
for regularization =  l1
Log Loss : 0.425125084428
for C = 0.0001
for regularization =  l2
Log Loss : 0.420825632387
for C = 0.001
for regularization =  l2
Log Loss : 0.41782210487
for C = 0.01
for regularization =  l2
Log Loss : 0.416152451382
for C = 0.1
for regularization =  l2
Log Loss : 0.418708886842
for C = 1
for regularization =  l2
Log Loss : 0.419529569292
for C = 10
for regularization =  l2
Log Loss : 0.419545631035
for C = 100
for regularization =  l2
Log Loss : 0.419545098667
for C = 1000
for regularization =  l2
Log Loss : 0.419545098667
for C = 10000
```

```
for regularization =  l2
Log Loss : 0.419545098671
```

In [160]:

```
r = [10 ** x for x in range(-4, 5)]

x1 = [math.log10(i) for i in r]

plt.plot(x1,l1_cv_log_error,label='log loss Error ')
plt.scatter(x1,l1_cv_log_error)
plt.legend()
plt.xlabel('log of hyperparameter')
plt.ylabel('Error measure')
plt.title('Cross Validation Error for each alpha with L1 regularizatio
n')
plt.grid()
plt.show()


x1 = [math.log10(i) for i in r]

plt.plot(x1,l2_cv_log_error,label='log loss Error ')
plt.scatter(x1,l2_cv_log_error)
plt.legend()
plt.xlabel('log of hyperparameter')
plt.ylabel('Error measure')
plt.title('Cross Validation Error for each alpha with L2 regularizatio
n')
plt.grid()
plt.show()
```





In [161]:
```
l1_best_alpha = np.argmin(l1_cv_log_error)
print("Best alpha with L1 regularization : ", alpha[l1_best_alpha])
print("Best L1 regularization Value: ", l1_cv_log_error[l1_best_alpha])
```

```
l2_best_alpha = np.argmin(l2_cv_log_error)
print("Best alpha with L2 regularization : ", alpha[l2_best_alpha])
print("Best L2 regularization Value: ", l2_cv_log_error[l2_best_alpha])
```

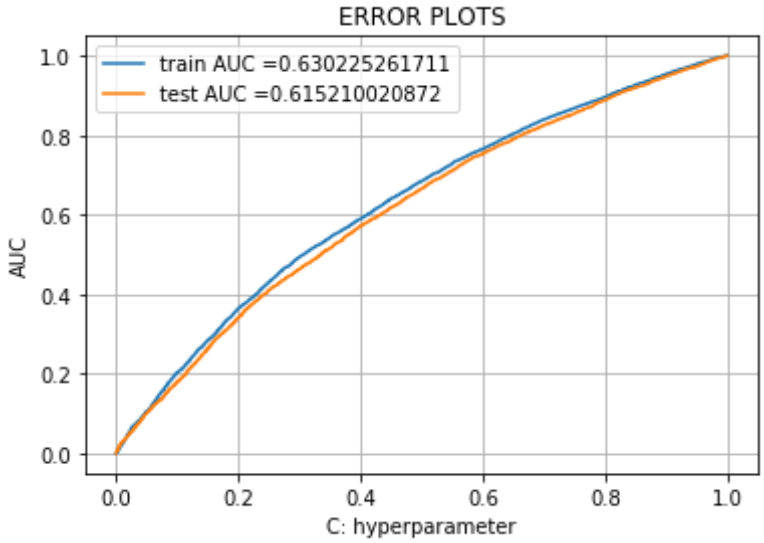
```
Best alpha with L1 regularization :  0.001
Best L1 regularization Value:  0.417277913841
Best alpha with L2 regularization :  0.01
Best L2 regularization Value:  0.416152451382
```

In [162]:
```
# Since L2 has better regularization value so taken L2 penalty and its
 best alpha

clf = SGDClassifier(class_weight='balanced', alpha=alpha[l2_best_alpha
], penalty='l2', loss='hinge', random_state=0)
clf.fit(X_tr, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_tr, y_train)

predict_y = sig_clf.predict_proba(X_tr)
print('For values of best alpha = ', alpha[l2_best_alpha], "The train l
og loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-1
5))
predict_y = sig_clf.predict_proba(X_cr)
print('For values of best alpha = ', alpha[l2_best_alpha], "The cross v
alidation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_,
eps=1e-15))
predict_y = sig_clf.predict_proba(X_te)
print('For values of best alpha = ', alpha[l2_best_alpha], "The test lo
g loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15
))
```

```
For values of best alpha =  0.01 The train log loss is: 0.412639664228
For values of best alpha =  0.01 The cross validation log loss is: 0.41
6152451382
For values of best alpha =  0.01 The test log loss is: 0.415147216083
```

In [163]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc
_curve.html#sklearn.metrics.roc_curve
```

```
from sklearn.metrics import roc_curve, auc

neigh = SGDClassifier(class_weight='balanced', alpha=alpha[l2_best_alph
a], penalty='l2', loss='hinge', random_state=0)
#neigh = LogisticRegression(C=0.5)
neigh.fit(X_tr, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(sig_clf, X_tr)
y_test_pred = batch_predict(sig_clf, X_te)

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, t
rain_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()
```



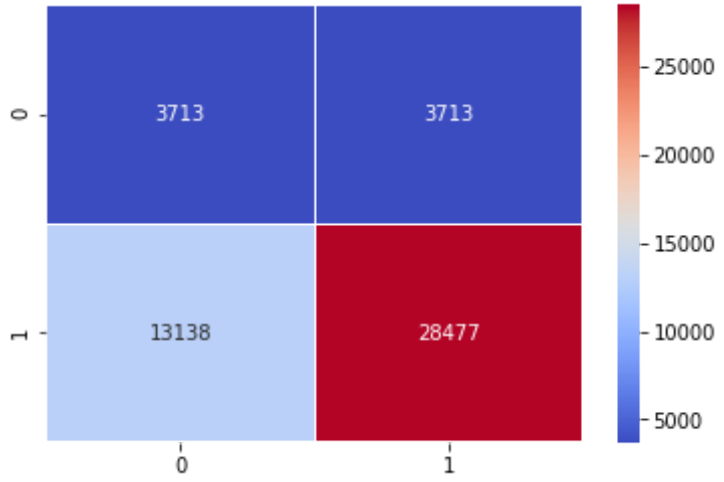
In [164]:
```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, tr
ain_fpr, train_fpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test
_fpr, test_fpr)))
```

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.834
[[ 3713  3713]
 [13138 28477]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.249999991611 for threshold 0.853
[[ 3226  2233]
 [12852 17741]]
```

In [165]:
```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html
# https://stackoverflow.com/questions/29647749/seaborn-showing-scientific-notation-in-heatmap-for-3-digit-numbers

# Train Confusion Matrix Heatmap
train_confusion_matrix = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))

print("Train Confusion Matrix")
sns.heatmap(train_confusion_matrix,annot=True,linewidth = 0.1, cmap='coolwarm', fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.834
Train Confusion Matrix
```

Out[165]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x163969fd940>
```

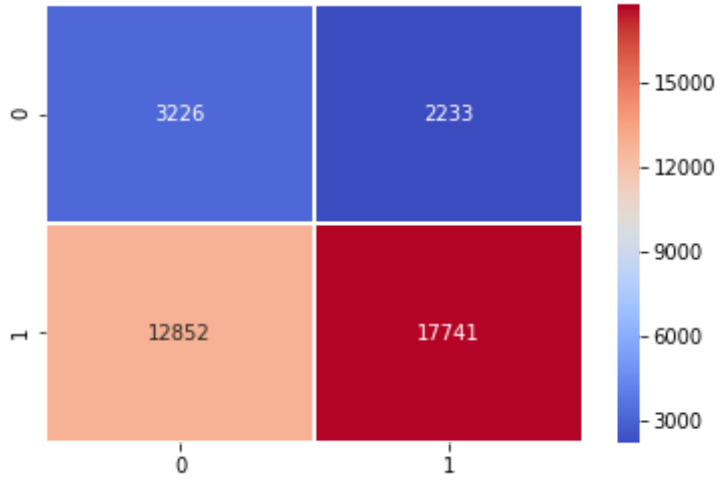


In [166]:
```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html

# Test Confusion Matrix Heatmap

test_confusion_matrix = confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr))
```

```
print("Test Confusion Matrix")
sns.heatmap(test_confusion_matrix,annot=True,linewidth = 0.5, cmap='coo
lwarm', fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.249999991611 for threshold 0.853
Test Confusion Matrix
```

Out[166]: <matplotlib.axes._subplots.AxesSubplot at 0x16396aa3438>



2.5 Support Vector Machines with added Features `Set 5`

In [167]:
```
X_Train_essay_tfidf.shape
```

Out[167]: (49041, 147)

In [169]:
```
#https://scikit-learn.org/stable/modules/generated/sklearn.decompositio
n.TruncatedSVD.html
from sklearn.decomposition import TruncatedSVD

svd = TruncatedSVD(n_components= 130, n_iter=7, random_state=42)
svd.fit(X_Train_essay_tfidf)
svd_train = svd.transform(X_Train_essay_tfidf)
```

```
svd_test = svd.transform(X_Test_essay_tfidf)
svd_cv = svd.transform(X_CV_essay_tfidf)

print("Shape of train matrix after dimension reduction ",svd_train.shape)
print("Shape of test matrix after dimension reduction ",svd_test.shape)
print("Shape of cv matrix after dimension reduction ",svd_cv.shape)
```

```
Shape of train matrix after dimension reduction  (49041, 130)
Shape of test matrix after dimension reduction  (36052, 130)
Shape of cv matrix after dimension reduction  (24155, 130)
```

In [170]:

```
# Train data stack
X_tr = hstack((X_Train_categories_one_hot,X_Train_sub_categories_one_hot,X_Train_school_state_one_hot,
               X_Train_teacher_prefix_one_hot,X_Train_grade_cat_one_hot,title_word_count_train,
               essay_word_count_train, essay_sent_pos_train, essay_sent_neg_train, essay_sent_neu_train,
               essay_sent_comp_train, svd_train)).tocsr()

# CV data Stack
X_cr = hstack((X_CV_categories_one_hot,X_CV_sub_categories_one_hot,X_CV_school_state_one_hot,
               X_CV_teacher_prefix_one_hot,X_CV_grade_cat_one_hot,title_word_count_cv,
               essay_word_count_cv, essay_sent_pos_cv, essay_sent_neg_cv, essay_sent_neu_cv,
               essay_sent_comp_cv, svd_cv)).tocsr()

# Test Data Stack
X_te = hstack((X_Test_categories_one_hot,X_Test_sub_categories_one_hot,X_Test_school_state_one_hot,
               X_Test_teacher_prefix_one_hot,X_Test_grade_cat_one_hot,title_word_count_test,
               essay_word_count_test, essay_sent_pos_test, essay_sent_neg_test, essay_sent_neu_test,
               essay_sent_comp_test, svd_test)).tocsr()
```

```
print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(49041, 236) (49041,)
(24155, 236) (24155,)
(36052, 236) (36052,)
====================================================================================================
```

In [171]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics.classification import accuracy_score, log_loss

alpha = [10 ** x for x in range(-4, 5)]

reg = ['l1', 'l2']

for i in reg:

    cv_log_error_array = []

    for j in alpha:

        print("for C =", j )
        print ("for regularization = ", i)

        clf = SGDClassifier( class_weight='balanced', alpha=j, penalty=
i, loss='hinge', random_state=0)
        clf.fit(X_tr, y_train)

        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(X_tr, y_train)

        sig_clf_probs = sig_clf.predict_proba(X_cr)
```

```
        cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=
clf.classes_, eps=1e-15))

        print("Log Loss :",log_loss(y_cv, sig_clf_probs))


    if (i == 'l1'):

        l1_cv_log_error = cv_log_error_array

    if (i == 'l2'):

        l2_cv_log_error = cv_log_error_array
```

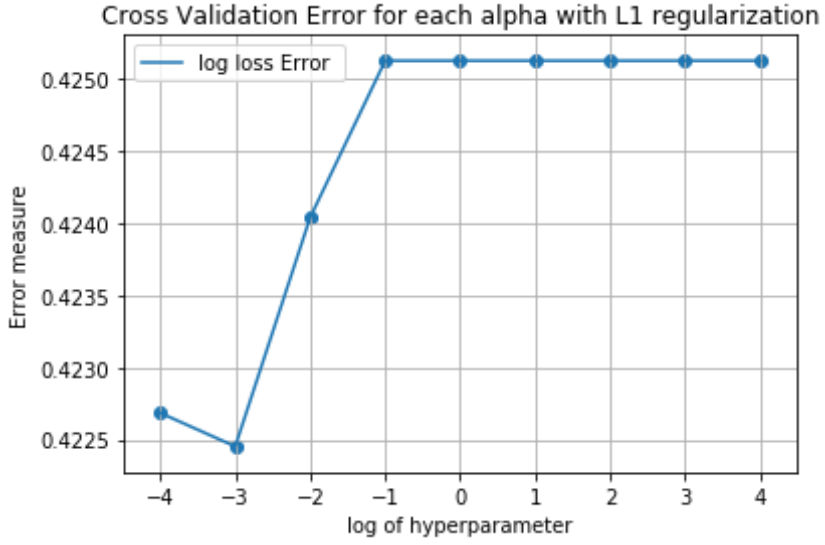
```
for C = 0.0001
for regularization =  l1
Log Loss : 0.422684378404
for C = 0.001
for regularization =  l1
Log Loss : 0.422451980342
for C = 0.01
for regularization =  l1
Log Loss : 0.424039163182
for C = 0.1
for regularization =  l1
Log Loss : 0.425125084428
for C = 1
for regularization =  l1
Log Loss : 0.425125084428
for C = 10
for regularization =  l1
Log Loss : 0.425125084428
for C = 100
for regularization =  l1
Log Loss : 0.425125084428
for C = 1000
for regularization =  l1
```

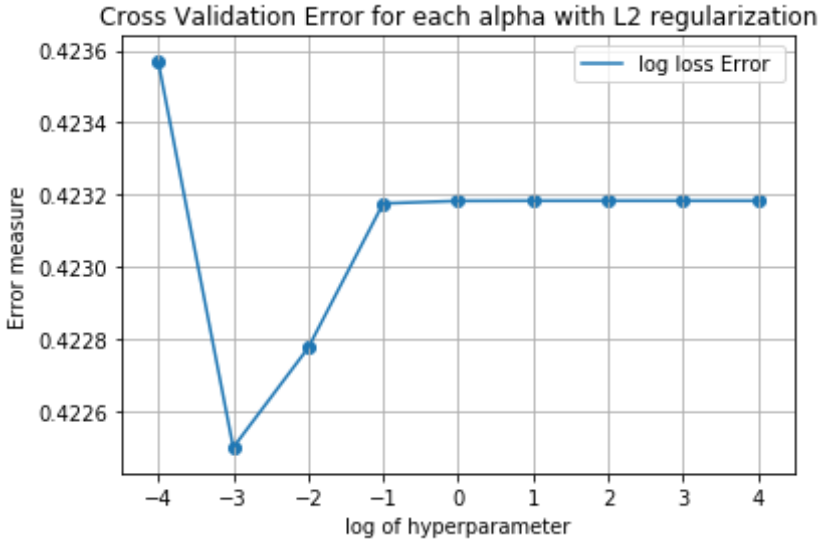
```
Log Loss : 0.425125084428
for C = 10000
for regularization =  l1
Log Loss : 0.425125084428
for C = 0.0001
for regularization =  l2
Log Loss : 0.423569155226
for C = 0.001
for regularization =  l2
Log Loss : 0.422499235601
for C = 0.01
for regularization =  l2
Log Loss : 0.422775679303
for C = 0.1
for regularization =  l2
Log Loss : 0.423176049184
for C = 1
for regularization =  l2
Log Loss : 0.423183007346
for C = 10
for regularization =  l2
Log Loss : 0.423183453646
for C = 100
for regularization =  l2
Log Loss : 0.423183453646
for C = 1000
for regularization =  l2
Log Loss : 0.423183417774
for C = 10000
for regularization =  l2
Log Loss : 0.423183417783
```

In [172]:

```
r = [10 ** x for x in range(-4, 5)]

x1 = [math.log10(i) for i in r]

plt.plot(x1,l1_cv_log_error,label='log loss Error ')
plt.scatter(x1,l1_cv_log_error)
plt.legend()
```

```
plt.xlabel('log of hyperparameter')
plt.ylabel('Error measure')
plt.title('Cross Validation Error for each alpha with L1 regularizatio
n')
plt.grid()
plt.show()


x1 = [math.log10(i) for i in r]

plt.plot(x1,l2_cv_log_error,label='log loss Error ')
plt.scatter(x1,l2_cv_log_error)
plt.legend()
plt.xlabel('log of hyperparameter')
plt.ylabel('Error measure')
plt.title('Cross Validation Error for each alpha with L2 regularizatio
n')
plt.grid()
plt.show()
```





In [173]:
```
l1_best_alpha = np.argmin(l1_cv_log_error)
print("Best alpha with L1 regularization : ", alpha[l1_best_alpha])
print("Best L1 regularization Value: ", l1_cv_log_error[l1_best_alpha])

l2_best_alpha = np.argmin(l2_cv_log_error)
print("Best alpha with L2 regularization : ", alpha[l2_best_alpha])
print("Best L2 regularization Value: ", l2_cv_log_error[l2_best_alpha])
```

```
Best alpha with L1 regularization :  0.001
Best L1 regularization Value:  0.422451980342
Best alpha with L2 regularization :  0.001
Best L2 regularization Value:  0.422499235601
```

In [174]:
```
# Since L2 has better regularization value so taken L2 penalty and its
 best alpha

clf = SGDClassifier(class_weight='balanced', alpha=alpha[l2_best_alpha
], penalty='l2', loss='hinge', random_state=0)
clf.fit(X_tr, y_train)
```

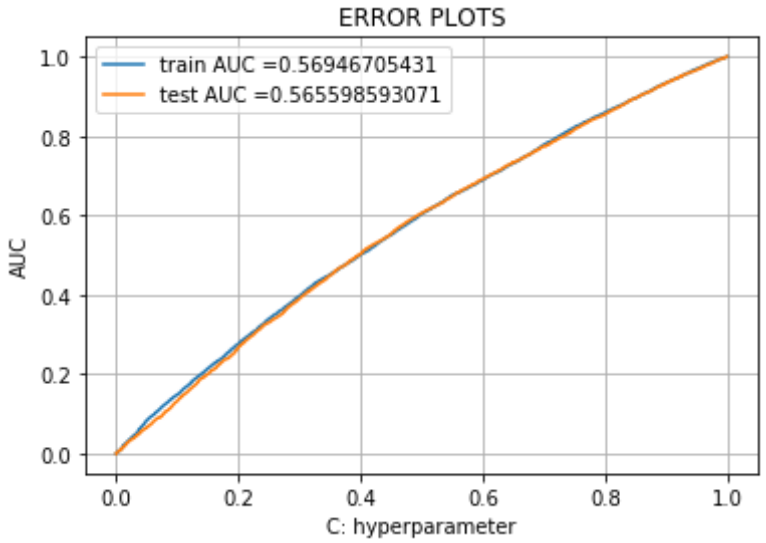
```
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_tr, y_train)

predict_y = sig_clf.predict_proba(X_tr)
print('For values of best alpha = ', alpha[l2_best_alpha], "The train l
og loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-1
5))
predict_y = sig_clf.predict_proba(X_cr)
print('For values of best alpha = ', alpha[l2_best_alpha], "The cross v
alidation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_,
eps=1e-15))
predict_y = sig_clf.predict_proba(X_te)
print('For values of best alpha = ', alpha[l2_best_alpha], "The test lo
g loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15
))
```

```
For values of best alpha =  0.001 The train log loss is: 0.421614158998
For values of best alpha =  0.001 The cross validation log loss is: 0.4
22499235601
For values of best alpha =  0.001 The test log loss is: 0.42202988579
```

In [175]:

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

neigh = SGDClassifier(class_weight='balanced', alpha=alpha[l2_best_alph
a], penalty='l2', loss='hinge', random_state=0)
#neigh = LogisticRegression(C=0.5)
neigh.fit(X_tr, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(sig_clf, X_tr)
y_test_pred = batch_predict(sig_clf, X_te)

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, t
rain_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()
```



In [176]:
```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, tr
ain_fpr, train_fpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test
_fpr, test_fpr)))
```

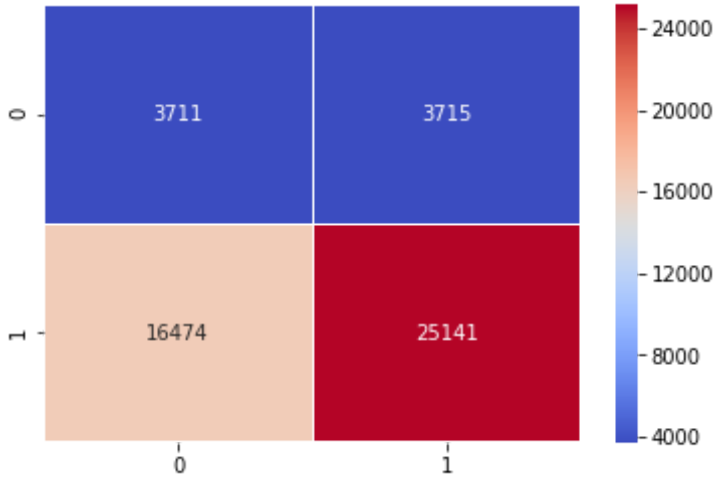
```
====================================================================================================
```

```
==============================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.249999927465 for threshold 0.845
[[ 3711  3715]
 [16474 25141]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.249999991611 for threshold 0.843
[[ 2285  3174]
 [ 9929 20664]]
```

In [177]:
```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html
# https://stackoverflow.com/questions/29647749/seaborn-showing-scientific-notation-in-heatmap-for-3-digit-numbers

# Train Confusion Matrix Heatmap
train_confusion_matrix = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))

print("Train Confusion Matrix")
sns.heatmap(train_confusion_matrix,annot=True,linewidth = 0.1, cmap='coolwarm', fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.249999927465 for threshold 0.845
Train Confusion Matrix
```

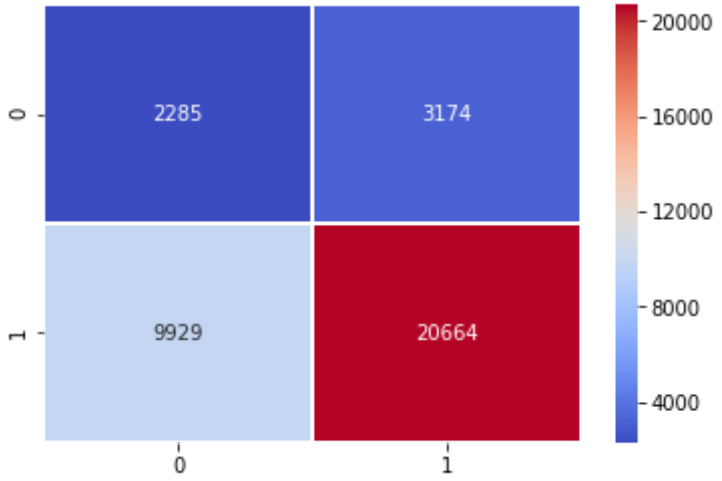
Out[177]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x16398242860>
```



In [178]:
```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html

# Test Confusion Matrix Heatmap

test_confusion_matrix = confusion_matrix(y_test, predict(y_test_pred, t
r_thresholds, test_fpr, test_fpr))

print("Test Confusion Matrix")
sns.heatmap(test_confusion_matrix,annot=True,linewidth = 0.5, cmap='coo
lwarm', fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.249999991611 for threshold 0.843
Test Confusion Matrix
```

Out[178]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x16397290a90>
```



Conclusion

In [179]:
```
# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Feature", "Best alpha", "Train AUC", "Test AUC"]

x.add_row(["Linear SVM on BOW", 1000, 0.61, 0.57])
x.add_row(["Linear SVM on TFIDF", 0.001, 0.60, 0.58])
x.add_row(["Linear SVM on AVG W2V", 0.001, 0.57, 0.56])
x.add_row(["Linear SVM on TFIDF W2V", 0.01, 0.63, 0.62])
x.add_row(["Linear SVM on Cat, Num, Essay Sentiments & truncated svd es
say", 0.001, 0.57, 0.56])

print(x)
```

```
+-------------------------------------------------------------------+-----
-------+----------+----------+
```

```
|                                Feature                                | Best
alpha | Train AUC | Test AUC |
+-----------------------------------------------------------------------+-----
-------+-----------+----------+
|                          Linear SVM on BOW                            |    1
000    |    0.61   |   0.57   |
|                          Linear SVM on TFIDF                          |   0.
001    |    0.6    |   0.58   |
|                         Linear SVM on AVG W2V                         |   0.
001    |    0.57   |   0.56   |
|                        Linear SVM on TFIDF W2V                        |
0.01    |    0.63   |   0.62   |
| Linear SVM on Cat, Num, Essay Sentiments & truncated svd essay |   0.
001    |    0.57   |   0.56   |
+-----------------------------------------------------------------------+-----
-------+-----------+----------+
```