DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located (Two-letter U.S. postal code). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy` • `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • `My students need hands on literacy materials to manage sensory needs!`
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*

Feature	Description
`project_essay_4`	Fourth application essay
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
`teacher_id`	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same teacher. **Example:** `2`

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [1]:

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

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
```

```
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

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

from tqdm import tqdm
import os

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

1.1 Reading Data

In [2]:

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

In [3]:

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

```
Number of data points in train data (109248, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' '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

In [5]:

```
project_data=project_data.sample(n=50000)
project_data.shape
```

Out[5]:

```
(50000, 17)
```

In [6]:

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

Out[6]:

```
1    42481
0     7519
Name: project_is_approved, dtype: int64
```

In [7]:

```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_data.columns)]


#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
project_data.drop('project_submitted_datetime', axis=1, inplace=True)
project_data.sort_values(by=['Date'], inplace=True)


# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084039
project_data = project_data[cols]


project_data.head(2)
```

Out[7]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_s
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2	

1.2 preprocessing of `project_subject_categories`

In [8]:

```
print(project_data['project_subject_categories'].head(5))
```

```
55660                       Math & Science
473                      Applied Learning
49228                   Literacy & Language
72638    Applied Learning, Music & The Arts
7176       Math & Science, Applied Learning
Name: project_subject_categories, dtype: object
```

In [9]:

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

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

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

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

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

In [10]:

```
print(project_data['clean_categories'].head(5))
```

```
55660                  Math_Science
473                AppliedLearning
49228             Literacy_Language
72638      AppliedLearning Music_Arts
7176     Math_Science AppliedLearning
Name: clean_categories, dtype: object
```

1.3 preprocessing of `project_subject_subcategories`

In [11]:

```
print(project_data['project_subject_subcategories'].head(5))
```

```
55660    Applied Sciences, Health & Life Science
473                            Early Development
49228                                    Literacy
72638               Extracurricular, Visual Arts
7176         Applied Sciences, Early Development
Name: project_subject_subcategories, dtype: object
```

In [12]:

```
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 & H
unger"]
```

```
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    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]))
```

In [13]:

```
print(project_data['clean_subcategories'].head(5))
```

```
55660    AppliedSciences Health_LifeScience
473                        EarlyDevelopment
49228                              Literacy
72638            Extracurricular VisualArts
7176       AppliedSciences EarlyDevelopment
Name: clean_subcategories, dtype: object
```

1.4 preprocessing of `school_state`

In [14]:

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

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

In [15]:

```
print(project_data['school_state'].head(5))
```

```
55660    CA
473      GA
49228    IL
72638    OH
7176     OH
Name: school_state, dtype: object
```

1.5 preprocessing of `project_grade_category`

In [16]:

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

Out[16]:

```
Grades PreK-2    20268
Grades 3-5       16955
Grades 6-8        7776
Grades 9-12       5001
Name: project_grade_category, dtype: int64
```

In [17]:

```
#https://stackoverflow.com/questions/36383821/pandas-dataframe-apply-function-to-column-strings-based-on-other-column-value
project_data['project_grade_category'] = project_data['project_grade_category'].str.replace(' ','_')
project_data['project_grade_category'] = project_data['project_grade_category'].str.replace('-','_')
project_data['project_grade_category'] = project_data['project_grade_category'].str.lower()
project_data['project_grade_category'].value_counts()
```

Out[17]:

```
grades_prek_2    20268
grades_3_5       16955
grades_6_8        7776
grades_9_12       5001
Name: project_grade_category, dtype: int64
```

In [18]:

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

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

In [19]:

```
print(sorted_grade_dict)
```

```
{'grades_9_12': 5001, 'grades_6_8': 7776, 'grades_3_5': 16955, 'grades_prek_2': 20268}
```

1.6 preprocessing of `teacher_prefix`

In [20]:

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

Out[20]:

```
Mrs.       26049
Ms.        17952
Mr.         4872
Teacher     1120
Dr.            6
Name: teacher_prefix, dtype: int64
```

In [21]:

```
project_data['teacher_prefix']=project_data['teacher_prefix'].fillna('Mrs.')
project_data['teacher_prefix'] = project_data['teacher_prefix'].str.replace('-','_')
project_data['teacher_prefix'] = project_data['teacher_prefix'].str.replace('.',' ')
project_data['teacher_prefix'] = project_data['teacher_prefix'].str.lower()
project_data['teacher_prefix'].value_counts()
```

Out[21]:

```
mrs        26050
ms         17952
mr          4872
teacher     1120
dr             6
Name: teacher_prefix, dtype: int64
```

In [22]:

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

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

In [23]:

```
print(project_data['teacher_prefix'].head(5))
```

```
55660     mrs
473       mrs
49228      ms
72638      mr
7176      mrs
Name: teacher_prefix, dtype: object
```

1.7 Text preprocessing

In [24]:

```
# 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 [25]:

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

Out[25]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_t
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	mrs	CA	2016-04-27 00:27:36	grades_prek_2	Engineer STEAM i the Prim Classro
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	mrs	GA	2016-04-27 00:53:00	grades_prek_2	Flexi Seating Flexi Learn

In [26]:

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

In [27]:

```
# 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[49999])
print("="*50)
```

I have been fortunate enough to use the Fairy Tale STEM kits in my classroom as well as the STEM j ournals, which my students really enjoyed. I would love to implement more of the Lakeshore STEM k its in my classroom for the next school year as they provide excellent and engaging STEM lessons.My students come from a variety of backgrounds, including language and socioeconomic statu s. Many of them don't have a lot of experience in science and engineering and these kits give me the materials to provide these exciting opportunities for my students.Each month I try to do several science or STEM/STEAM projects. I would use the kits and robot to help guide my science i nstruction in engaging and meaningful ways. I can adapt the kits to my current language arts paci ng guide where we already teach some of the material in the kits like tall tales (Paul Bunyan) or Johnny Appleseed. The following units will be taught in the next school year where I will implement these kits: magnets, motion, sink vs. float, robots. I often get to these units and don 't know If I am teaching the right way or using the right materials. The kits will give me additional ideas, strategies, and lessons to prepare my students in science.It is challenging to d evelop high quality science activities. These kits give me the materials I need to provide my students with science activities that will go along with the curriculum in my classroom. Although I have some things (like magnets) in my classroom, I don't know how to use them effectively. The kits will provide me with the right amount of materials and show me how to use them in an appropriate way.

==========

Clock ticking. Lights flickering. Perfume. New clothes. These daily encounters often go unnoticed by most, but these, and many other experiences can cause sensory overload for students with autism or sensory processing disorders.I have the joy of teaching 12 students in a 2nd-4th grade special education classroom setting which includes children with mild intellectual disabilities, autism, a nd other handicapping impairments. Each day provides a new learning experience not only for my stu dents, but for me as well!Sensory overload occurs when an individual has difficulty processing eve ryday sensory information. Sensory overload causes stress and anxiety, which can lead to withdrawal, challenging behaviors, and even meltdowns. Students can be over- and under-stimulated. That’s where a sensory room comes in. Sensory rooms are designed to give individuals with sensory processing disorders the opportunity to learn to relax and self-regulate through learning and prac ticing stress management techniques, and through sensory integration. A sensory room provides act ivities to calm or stimulate the body and mind through each of the senses. Research has shown that individuals who participate in sensory integration show an increase in calmness, concentration, fo cus,and alertness, as well as a decrease in aggression. My desire is to create a safe environment that will provide opportunities for my students to meet their visual, olfactory, oral-motor, proprioception, tactile, and auditory needs.This sensory room will be used by my students and the three other special education classes at our school. It will also be available for any student who needs to have their sensory needs met. By having my project supported I will be able to turn an empty classroom into a sensory room that will allow me to meet the sensory input needs of my students, which in turn will allow me to greater meet their academic needs as well. Thank you for supporting my classroom and my students!

==========

My classroom is a place where students feel like a family and we build relationships with one anot her. My students develop a love a reading through the books I incorporate within my classroom les sons. They run to the library and find books by the same author or same topic and begin their jou rney.My students are a diverse group. They come in with many different backgrounds and experiences. They are eager to learn and excited to be at school. Their academic abilities range in different areas. They are interested in Lego's, Littlest Pet Shop, and utilizing technology. Th ey are competitive with each other and enjoy playing games. My school is a large K-8 elementary s chool. We are located in a low socioeconomic area which means 70% of our students receive free or reduced lunch. Each grade level has at least three to four classrooms. Our class sizes range from 18-higher. We are located in a small mountain community where everyone knows one another. I would describe my school as a place where students are put first. We encourage parental involvement at home and at school. I would say we are a close school district.One of my main goals at the beginning of the school year is to build classroom community. We would meet on our c lassroom rug daily. The students would have a spot in the room for our novel reading where we coul d gather together instead of sitting at our desks. Being able to gather at one spot in the room w ould allow for the students to have a deeper discussion about the novel, be able to answer questio ns about main idea, cause and effect, and other literacy concepts incorporated into a novel study. The students will use the rug to partner read which would build fluency. At the end of 3rd grade they should be reading 100 wpm. The students would also use the rug for partner/group work. Thi s area would provide a place for them to work collaboratively with one another practicing math concepts such as fractions or place value. Learning through collaboration is an important skill and allows students who may not understand quite as easily through whole group instruction.A rug would improve my classroom by providing an opportunity to build classroom community. My students would be able to meet as a group to do activities, work collaboratively practicing important literacy and math concepts, and have deeper discussions. Students would build fluency, comprehension, and vocabulary skills. They would build an understanding in math concepts that lay the academic foundation for later in their academic career.

==========

I have so much to say about my students I don't even know where to begin. They love life, they bound into school with unmatched energy, and they have the best smiles around. Everyday we laugh, learn, and grow together in reading, writing, math, science, social studies, and LIFE. \r\nDespite the many hardships life throws these kids they persevere with a passion I learn from e very day. Our school is a Title 1 school and over 70% of our students qualify for free or reduce d lunch. I rarely see these kids quit and when things get tough they back up and try again.I know my third graders very well, I was lucky enough to teach them in second grade. I know their learning styles, their habits, their stressors, and what they need on a daily basis to be

```
successful.  They like to move and fidget a lot but we've got important work that needs to be done
.  \r\nAt every chance I get we engage in a brain break but while their brains are working hard I
would like to give them the opportunity to work their bodies, too by utilizing bands on their
chairs or seats.  At the same time, we need our classroom to be a safe and calming environment.  U
nfortunately, I cannot control what happens when they leave my room but when they are there I want
them to be comfortable.  I will use the light covers to create that space for them.nannan
==================================================
I teach first grade in a Title I school. Although my students may live in an impoverished
neighborhood, my students are vibrant and will brighten up your day! They can be an energetic bunc
h! They are hard workers and love to show it to anyone who walks into our classroom or whoever the
y see in the hallway! \r\n\r\nThey are an amazing group of students with different abilities and p
ersonalities. They are busting at the seams to learn.  They are excited to learn and grow.\"Love o
f beauty is taste. The creation of beauty is art.\"\r\n~Ralph Waldo Emerson\r\n\r\nMy students are
hard working First Grade students.  I want to brighten up their work by making their centers
colorful.  I will use the ink and colorful paper to create centers that will keep them engage.  I
will laminate their centers to make my centers eco-friendly.  My students will be using the ink an
d colorful paper to print their projects and papers.  My students write stories and draw their
illustrations.  The ink and paper will allow my students to use their technology skills to type th
eir stories and draw their illustrations on the computer to create their books.  This project will
connect multiple subjects and allow students of various abilities to participate and demonstrate t
heir work in a variety of ways.\r\nThis project will enhance my students' work.  They work hard
and would love to display their work around the classroom and in the school.  The colorful display
will brighten up our school.nannan
==================================================
```

In [28]:

```
# 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 [29]:

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

```
I have so much to say about my students I do not even know where to begin.  They love life, they b
ound into school with unmatched energy, and they have the best smiles around.  Everyday we laugh,
learn, and grow together in reading, writing, math, science, social studies, and LIFE.
\r\nDespite the many hardships life throws these kids they persevere with a passion I learn from e
very day.   Our school is a Title 1 school and over 70% of our students qualify for free or reduce
d lunch.  I rarely see these kids quit and when things get tough they back up and try again.I know
my third graders very well, I was lucky enough to teach them in second grade.  I know their
learning styles, their habits, their stressors, and what they need on a daily basis to be
successful.  They like to move and fidget a lot but we have got important work that needs to be do
ne.  \r\nAt every chance I get we engage in a brain break but while their brains are working hard
I would like to give them the opportunity to work their bodies, too by utilizing bands on their
chairs or seats.  At the same time, we need our classroom to be a safe and calming environment.  U
nfortunately, I cannot control what happens when they leave my room but when they are there I want
them to be comfortable.  I will use the light covers to create that space for them.nannan
==================================================
```

In [30]:

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

```
sent = sent.replace('\\n', ' ')
print(sent)
```

```
I have so much to say about my students I do not even know where to begin.  They love life, they b
ound into school with unmatched energy, and they have the best smiles around.  Everyday we laugh,
learn, and grow together in reading, writing, math, science, social studies, and LIFE.    Despite
the many hardships life throws these kids they persevere with a passion I learn from every day.
Our school is a Title 1 school and over 70% of our students qualify for free or reduced lunch.  I
rarely see these kids quit and when things get tough they back up and try again.I know my third gr
aders very well, I was lucky enough to teach them in second grade.  I know their learning styles,
their habits, their stressors, and what they need on a daily basis to be successful.  They like to
move and fidget a lot but we have got important work that needs to be done.    At every chance I g
et we engage in a brain break but while their brains are working hard I would like to give them th
e opportunity to work their bodies, too by utilizing bands on their chairs or seats.  At the same
time, we need our classroom to be a safe and calming environment.  Unfortunately, I cannot control
what happens when they leave my room but when they are there I want them to be comfortable.  I wil
l use the light covers to create that space for them.nannan
```

In [31]:

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

```
I have so much to say about my students I do not even know where to begin They love life they
bound into school with unmatched energy and they have the best smiles around Everyday we laugh lea
rn and grow together in reading writing math science social studies and LIFE Despite the many hard
ships life throws these kids they persevere with a passion I learn from every day Our school is a
Title 1 school and over 70 of our students qualify for free or reduced lunch I rarely see these ki
ds quit and when things get tough they back up and try again I know my third graders very well I w
as lucky enough to teach them in second grade I know their learning styles their habits their
stressors and what they need on a daily basis to be successful They like to move and fidget a lot
but we have got important work that needs to be done At every chance I get we engage in a brain br
eak but while their brains are working hard I would like to give them the opportunity to work thei
r bodies too by utilizing bands on their chairs or seats At the same time we need our classroom to
be a safe and calming environment Unfortunately I cannot control what happens when they leave my r
oom but when they are there I want them to be comfortable I will use the light covers to create th
at space for them nannan
```

In [32]:

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

In [33]:

```
# 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%|██████████| 50000/50000 [01:06<00:00, 755.65it/s]
```

In [34]:

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

Out[34]:

```
'i much say students i not even know begin they love life bound school unmatched energy best smiles around everyday laugh learn grow together reading writing math science social studies life despite many hardships life throws kids persevere passion i learn every day our school title 1 school 70 students qualify free reduced lunch i rarely see kids quit things get tough back try i know third graders well i lucky enough teach second grade i know learning styles habits stressors need daily basis successful they like move fidget lot got important work needs done at every chance i get engage brain break brains working hard i would like give opportunity work bodies utilizing bands chairs seats at time need classroom safe calming environment unfortunately i cannot control happens leave room i want comfortable i use light covers create space nannan'
```

1.8 Preprocessing of `project_title`

In [35]:

```
# similarly you can preprocess the titles also
preprocessed_titles = []
# tqdm is for printing the status bar
for sentence in tqdm(project_data['project_title'].values):
    sent = decontracted(sentence)
    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_titles.append(sent.lower().strip())
```

```
100%|██████████| 50000/50000 [00:03<00:00, 16607.82it/s]
```

In [36]:

```
print(45660,preprocessed_titles[45660])
```

```
45660 our world view
```

In [37]:

```
project_data['project_title']=preprocessed_titles
```

In [38]:

```
print(45660,project_data['project_title'].iloc[45660])
```

```
45660 our world view
```

Sentiment Analysis of essays

In [39]:

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

neg = []
pos = []
neu = []
compound = []

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

```
[nltk_data] Downloading package vader_lexicon to C:\Users\ABHISHEK
[nltk_data]     SINGH\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
100%|██████████████████████████████████████████████████████████████████████████| 50000/50000 [21
:48<00:00, 38.21it/s]
```

In [40]:

```
project_data["pos"] = pos
```

In [41]:

```
project_data["neg"] = neg
```

In [42]:

```
project_data["neu"] = neu
```

In [43]:

```
project_data["compound"] = compound
```

Number of Words in Title

In [44]:

```
title_word_count = []

for a in project_data["project_title"] :
    b = len(a.split())
    title_word_count.append(b)

project_data["title_word_count"] = title_word_count
```

Number of Words in Essays

In [45]:

```
essay_word_count = []

for a in project_data["essay"] :
    b = len(a.split())
    essay_word_count.append(b)

project_data["essay_word_count"] = essay_word_count
```

1.9 Preparing data for models

In [46]:

```
project_data.columns
```

Out[46]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', '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_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'pos', 'neg', 'neu',
       'compound', 'title_word_count', 'essay_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
```

Assignment 5: Logistic Regression

1. **[Task-1] Logistic Regression(either SGDClassifier with log loss, or LogisticRegression) on these feature sets**
   - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (`BOW with bi-grams` with `min_df=10` and `max_features=5000`)
   - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (`TFIDF with bi-grams` with `min_df=10` and `max_features=5000`)
   - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
   - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)
2. **Hyper paramter tuning (find best hyper parameters corresponding the algorithm that you choose)**
   - Find the best hyper parameter which will give the maximum AUC 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.

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 Logistic Regression on the below feature set Set 5 by finding the best hyper parameter as suggested in step 2 and step 3.**
5. Consider these set of features Set 5 :

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

   And apply the Logistic regression on these features by finding the best hyper paramter as suggested in step 2 and step 3

6. **Conclusion**

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

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. Logistic Regression

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

In [47]:

```
y = project_data['project_is_approved']
print(y.shape)
```

```
(50000,)
```

In [48]:

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

In [49]:

```
X=project_data
print(X.shape)
```

```
(50000, 23)
```

In [50]:

```
#train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
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]:

```
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)
print("="*100)
```

```
(22445, 23) (22445,)
(11055, 23) (11055,)
(16500, 23) (16500,)
====================================================================================================
```

Encoding of Text Data

In [52]:

```
from sklearn.feature_extraction.text import CountVectorizer
```

BOW of Essay

In [53]:

```
vectorizer = CountVectorizer(min_df=10,ngram_range=(2,2), max_features=5000)
```

In [54]:

```
vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data
```

Out[54]:

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

In [55]:

```
# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['essay'].values)
```

In [56]:

```
X_cv_essay_bow = vectorizer.transform(X_cv['essay'].values)
```

In [57]:

```
X_test_essay_bow = vectorizer.transform(X_test['essay'].values)
```

In [58]:

```
print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 5000) (22445,)
```

```
(11055, 5000) (11055,)
(16500, 5000) (16500,)
==========================================================================================================
```

BOW of Title

In [59]:

```
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
```

In [60]:

```
vectorizer.fit(X_train['project_title'].values) # fit has to happen only on train data
```

Out[60]:

```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                lowercase=True, max_df=1.0, max_features=5000, min_df=10,
                ngram_range=(1, 4), preprocessor=None, stop_words=None,
                strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, vocabulary=None)
```

In [61]:

```
# we use the fitted CountVectorizer to convert the text to vector
X_train_title_bow = vectorizer.transform(X_train['project_title'].values)
```

In [62]:

```
X_cv_title_bow = vectorizer.transform(X_cv['project_title'].values)
```

In [63]:

```
X_test_title_bow = vectorizer.transform(X_test['project_title'].values)
```

In [64]:

```
print("After vectorizations")
print(X_train_title_bow.shape, y_train.shape)
print(X_cv_title_bow.shape, y_cv.shape)
print(X_test_title_bow.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 2010) (22445,)
(11055, 2010) (11055,)
(16500, 2010) (16500,)
==========================================================================================================
```

TFIDF of Essay

In [65]:

```
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(2,2), max_features=5000)
```

In [66]:

```
vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data
```

Out[66]:

```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.float64'>, encoding='utf-8'
```

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

In [67]:

```
# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer.transform(X_train['essay'].values)
```

In [68]:

```
X_cv_essay_tfidf = vectorizer.transform(X_cv['essay'].values)
```

In [69]:

```
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values)
```

In [70]:

```
print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
====================================================================================================
```

TFIDF of Title

In [71]:

```
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
```

In [72]:

```
vectorizer.fit(X_train['project_title'].values) # fit has to happen only on train data
```

Out[72]:

```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.float64'>, encoding='utf-8',
                input='content', lowercase=True, max_df=1.0, max_features=5000,
                min_df=10, ngram_range=(1, 4), norm='l2', preprocessor=None,
                smooth_idf=True, stop_words=None, strip_accents=None,
                sublinear_tf=False, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, use_idf=True, vocabulary=None)
```

In [73]:

```
# we use the fitted CountVectorizer to convert the text to vector
X_train_title_tfidf = vectorizer.transform(X_train['project_title'].values)
```

In [74]:

```
X_cv_title_tfidf = vectorizer.transform(X_cv['project_title'].values)
```

In [75]:

```
X_test_title_tfidf = vectorizer.transform(X_test['project_title'].values)
```

```
X_test_title_tfidf = vectorizer.transform(X_test['project_title'].values)
```

In [76]:

```
print("After vectorizations")
print(X_train_title_tfidf.shape, y_train.shape)
print(X_cv_title_tfidf.shape, y_cv.shape)
print(X_test_title_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 2010) (22445,)
(11055, 2010) (11055,)
(16500, 2010) (16500,)
====================================================================================================
```

Avg W2V of Essay

In [77]:

```
# 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,encoding = "ISO-8859-1")
    glove_words =  set(model.keys())
```

In [78]:

```
# average Word2Vec
# compute average word2vec for each essay.
avg_w2v_essay_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essay_train.append(vector)

print(len(avg_w2v_essay_train))
print(len(avg_w2v_essay_train[0]))
print(type(avg_w2v_essay_train))
```

```
100%|██████████| 22445/22445 [00:21<00:00, 1027.50it/s]
```

```
22445
300
<class 'list'>
```

In [79]:

```
# average Word2Vec
# compute average word2vec for each essay.
avg_w2v_essay_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essay_cv.append(vector)
```

```
print(len(avg_w2v_essay_cv))
print(len(avg_w2v_essay_cv[0]))
print(type(avg_w2v_essay_cv))
```

```
100%|████████████████████████████████████████████████████████████| 11055/11055 [00:
11<00:00, 963.86it/s]
```

```
11055
300
<class 'list'>
```

In [80]:

```
# average Word2Vec
# compute average word2vec for each essay.
avg_w2v_essay_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essay_test.append(vector)

print(len(avg_w2v_essay_test))
print(len(avg_w2v_essay_test[0]))
print(type(avg_w2v_essay_test))
```

```
100%|████████████████████████████████████████████████████████████| 16500/16500
[00:16<00:00, 1013.56it/s]
```

```
16500
300
<class 'list'>
```

Avg W2V of Title

In [81]:

```
# average Word2Vec
# compute average word2vec for each essay.
avg_w2v_title_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_title_train.append(vector)

print(len(avg_w2v_title_train))
print(len(avg_w2v_title_train[0]))
print(type(avg_w2v_title_train))
```

```
100%|████████████████████████████████████████████████████████████| 22445/22445
[00:00<00:00, 27862.23it/s]
```

```
22445
300
<class 'list'>
```

In [82]:

```
# average Word2Vec
# compute average word2vec for each essay.
avg_w2v_title_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_title_cv.append(vector)

print(len(avg_w2v_title_cv))
print(len(avg_w2v_title_cv[0]))
print(type(avg_w2v_title_cv))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 11055/11055
[00:00<00:00, 27882.96it/s]
```

```
11055
300
<class 'list'>
```

In [83]:

```
# average Word2Vec
# compute average word2vec for each essay.
avg_w2v_title_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_title_test.append(vector)

print(len(avg_w2v_title_test))
print(len(avg_w2v_title_test[0]))
print(type(avg_w2v_title_test))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 16500/16500
[00:00<00:00, 27638.55it/s]
```

```
16500
300
<class 'list'>
```

TFIDF-W2V of Essay

In [84]:

```
tfidf_model = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
tfidf_model.fit(X_train['essay'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [85]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_train_essay = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
```

```
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_train_essay.append(vector)

print(len(tfidf_w2v_train_essay))
print(len(tfidf_w2v_train_essay[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 22445/22445 [03:
35<00:00, 104.17it/s]
```

```
22445
300
```

In [86]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_cv_essay = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_cv_essay.append(vector)

print(len(tfidf_w2v_cv_essay))
print(len(tfidf_w2v_cv_essay[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 11055/11055 [01:
48<00:00, 101.82it/s]
```

```
11055
300
```

In [87]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_test_essay = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_test_essay.append(vector)
```

```
print(len(tfidf_w2v_test_essay))
print(len(tfidf_w2v_test_essay[0]))
```

```
100%|██████████| 16500/16500 [02:40<00:00, 102.56it/s]
```

```
16500
300
```

TFIDF-W2V of Title

In [88]:

```
tfidf_model = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
tfidf_model.fit(X_train['project_title'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [89]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_train_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_train_title.append(vector)

print(len(tfidf_w2v_train_title))
print(len(tfidf_w2v_train_title[0]))
```

```
100%|██████████| 22445/22445 [00:01<00:00, 15235.06it/s]
```

```
22445
300
```

In [90]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_cv_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_cv_title.append(vector)
```

```
print(len(tfidf_w2v_cv_title))
print(len(tfidf_w2v_cv_title[0]))
```

```
100%|██████████| 11055/11055 [00:00<00:00, 15459.40it/s]
```

```
11055
300
```

In [91]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_test_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_test_title.append(vector)

print(len(tfidf_w2v_test_title))
print(len(tfidf_w2v_test_title[0]))
```

```
100%|██████████| 16500/16500 [00:01<00:00, 15380.35it/s]
```

```
16500
300
```

2.3 Make Data Model Ready: encoding eassay, and project_title

In [92]:

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
```

In [93]:

```
price_data.head(5)
```

Out[93]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21
2	p000003	298.97	4
3	p000004	1113.69	98
4	p000005	485.99	8

In [94]:

```
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 [95]:

```
X_train=X_train.fillna(0)
X_cv=X_cv.fillna(0)
X_test=X_test.fillna(0)
```

Normalizing the numerical features: Price

In [96]:

```
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.reshape(-1,1))
X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(-1,1))
print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Normalizing the numerical features: Number of previously posted projects

In [97]:

```
normalizer = Normalizer()
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_train_project_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'
].values.reshape(-1,1))
X_cv_project_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].valu
es.reshape(-1,1))
X_test_project_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].
values.reshape(-1,1))
print("After vectorizations")
print(X_train_project_norm.shape, y_train.shape)
print(X_cv_project_norm.shape, y_cv.shape)
print(X_test_project_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Normalizing the numerical features: Title word Count

In [98]:

```
normalizer = Normalizer()
normalizer.fit(X_train['title_word_count'].values.reshape(-1,1))
X_train_title_norm = normalizer.transform(X_train['title_word_count'].values.reshape(-1,1))
X_cv_title_norm = normalizer.transform(X_cv['title_word_count'].values.reshape(-1,1))
```

```
X_test_title_norm = normalizer.transform(X_test['title_word_count'].values.reshape(-1,1))
print("After vectorizations")
print(X_train_title_norm.shape, y_train.shape)
print(X_cv_title_norm.shape, y_cv.shape)
print(X_test_title_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Normalizing the numerical features: Essay word Count

In [99]:

```
normalizer = Normalizer()
normalizer.fit(X_train['essay_word_count'].values.reshape(-1,1))
X_train_essay_norm = normalizer.transform(X_train['essay_word_count'].values.reshape(-1,1))
X_cv_essay_norm = normalizer.transform(X_cv['essay_word_count'].values.reshape(-1,1))
X_test_essay_norm = normalizer.transform(X_test['essay_word_count'].values.reshape(-1,1))
print("After vectorizations")
print(X_train_essay_norm.shape, y_train.shape)
print(X_cv_essay_norm.shape, y_cv.shape)
print(X_test_essay_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Normalizing the numerical features: Essay Sentiments-Positive

In [100]:

```
normalizer = Normalizer()
normalizer.fit(X_train['pos'].values.reshape(-1,1))
essay_sent_pos_train = normalizer.transform(X_train['pos'].values.reshape(-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)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Normalizing the numerical features: Essay Sentiments-Negative

In [101]:

```
normalizer = Normalizer()
normalizer.fit(X_train['neg'].values.reshape(-1,1))
essay_sent_neg_train = normalizer.transform(X_train['neg'].values.reshape(-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)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Normalizing the numerical features: Essay Sentiments-Neutral

In [102]:

```
normalizer = Normalizer()
normalizer.fit(X_train['neu'].values.reshape(-1,1))
essay_sent_neu_train = normalizer.transform(X_train['neu'].values.reshape(-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)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Normalizing the numerical features: Essay Sentiments-Compound

In [103]:

```
normalizer = Normalizer()
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.reshape(-1,1))
essay_sent_comp_test = normalizer.transform(X_test['compound'].values.reshape(-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
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Vectorizing Categorical features

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

Vectorizing Categorical features: project grade category

In [104]:

```
from sklearn.feature_extraction.text import CountVectorizer
```

In [105]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_grade_dict.keys()), lowercase=False, binary=True)
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].values)
X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].values)

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 4) (22445,)
(11055, 4) (11055,)
(16500, 4) (16500,)
['grades_9_12', 'grades_6_8', 'grades_3_5', 'grades_prek_2']
====================================================================================================
```

Vectorizing Categorical features: teacher prefix

In [106]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_teacher_dict.keys()), lowercase=False, binary=True)
vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_cv_teacher_ohe.shape, y_cv.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 5) (22445,)
(11055, 5) (11055,)
(16500, 5) (16500,)
['dr', 'teacher', 'mr', 'ms', 'mrs']
====================================================================================================
```

Vectorizing Categorical features: school state

In [107]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_state_dict.keys()), lowercase=False, binary=True)
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)
```

```
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 51) (22445,)
(11055, 51) (11055,)
(16500, 51) (16500,)
['WY', 'VT', 'ND', 'MT', 'RI', 'NE', 'SD', 'NH', 'AK', 'DE', 'WV', 'HI', 'ME', 'DC', 'NM', 'KS', 'I
A', 'ID', 'AR', 'CO', 'OR', 'MN', 'NV', 'KY', 'MS', 'MD', 'TN', 'WI', 'AL', 'CT', 'UT', 'VA', 'AZ',
'NJ', 'OK', 'LA', 'WA', 'MA', 'OH', 'IN', 'MO', 'PA', 'MI', 'SC', 'GA', 'IL', 'NC', 'FL', 'TX', 'NY
', 'CA']
====================================================================================================
```

Vectorizing Categorical features: clean categories

In [108]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True
)
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_cat_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_cv_cat_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_cat_ohe = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_cat_ohe.shape, y_train.shape)
print(X_cv_cat_ohe.shape, y_cv.shape)
print(X_test_cat_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 9) (22445,)
(11055, 9) (11055,)
(16500, 9) (16500,)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
====================================================================================================
```

Vectorizing Categorical features: clean subcategories

In [109]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=
True)
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_sub_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
X_cv_sub_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_sub_ohe = vectorizer.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_sub_ohe.shape, y_train.shape)
print(X_cv_sub_ohe.shape, y_cv.shape)
print(X_test_sub_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 30) (22445,)
```

```
(11055, 30) (11055,)
(16500, 30) (16500,)
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'CharacterEducation', 'PerformingArts', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'EarlyDevelopment', 'Health_LifeScience', 'ESL
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
==========================================================================================
```

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

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

Applying Logistic Regression on BOW, SET 1

Creating Data Matrix

In [110]:

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

# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_bow,X_train_title_bow, X_train_state_ohe, X_train_teacher_ohe,
X_train_grade_ohe,X_train_cat_ohe,X_train_sub_ohe, X_train_price_norm,X_train_project_norm)).tocsr
()
X_cr = hstack((X_cv_essay_bow,X_cv_title_bow, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe,X_cv
_cat_ohe,X_cv_sub_ohe, X_cv_price_norm,X_cv_project_norm)).tocsr()
X_te = hstack((X_test_essay_bow,X_test_title_bow, X_test_state_ohe, X_test_teacher_ohe, X_test_grad
e_ohe,X_test_cat_ohe,X_test_sub_ohe, X_test_price_norm,X_test_project_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
(22445, 7111) (22445,)
(11055, 7111) (11055,)
(16500, 7111) (16500,)
==========================================================================================
```

Hyperparameter Tuning: Simple for loop (if you are having memory limitations use this)

In [111]:

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

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

    return y_data_pred
```

In [112]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
log_alphas=[]

parameters = {'C':[0.0001,0.0005,0.001,0.005,0.01,0.05,0.1,0.5,1,2.5,5]}

for i in tqdm(parameters['C']):
    neigh = LogisticRegression(C=i)
    neigh.fit(X_tr, y_train)

    y_train_pred = batch_predict(neigh, X_tr)
    y_cv_pred = batch_predict(neigh, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
100%|██████████| 11/11
[01:05<00:00, 13.16s/it]
```

In [113]:

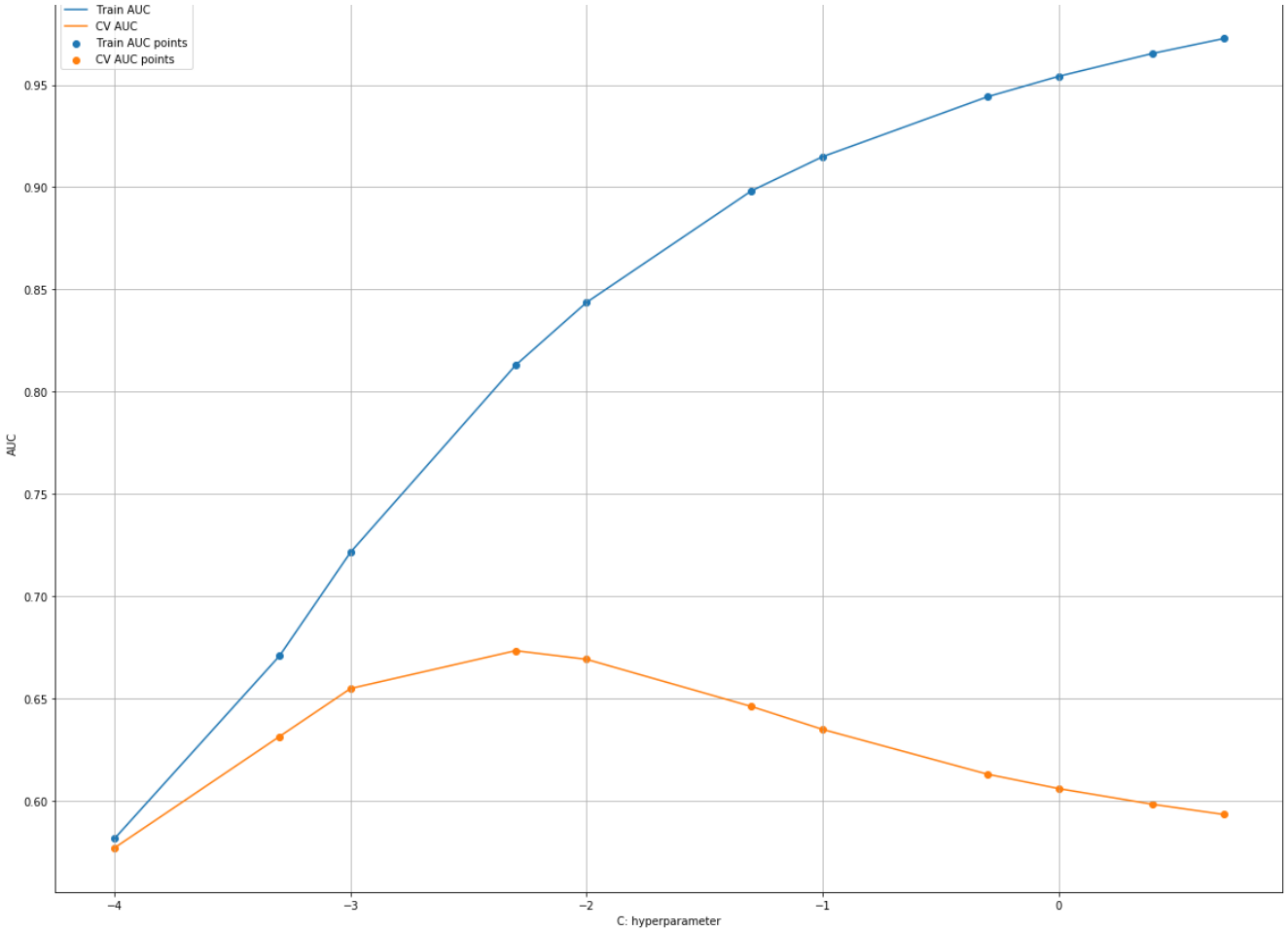
```
import math
for a in tqdm(parameters['C']):
    b = math.log10(a)
    log_alphas.append(b)
print(log_alphas)
```

```
100%|██████████| 11/11
[00:00<00:00, 10927.84it/s]
```

```
[-4.0, -3.3010299956639813, -3.0, -2.3010299956639813, -2.0, -1.3010299956639813, -1.0, -0.3010299
956639812, 0.0, 0.3979400086720376, 0.6989700043360189]
```

In [114]:

```
plt.figure(figsize=(20,15))
plt.plot(log_alphas, train_auc, label='Train AUC')
plt.plot(log_alphas, cv_auc, label='CV AUC')

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

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



In [115]:

```
best_k=0.005
```

Train The Model

In [116]:

```
from sklearn.metrics import roc_curve, auc

neigh = LogisticRegression(C=best_k)
neigh.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, 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)

x=[0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
```

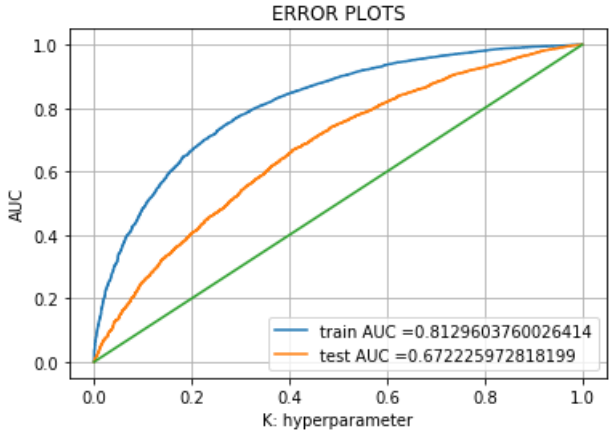
In [117]:

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

# neigh = LogisticRegression(C=best_k)
# neigh.fit(X_tr, y_train)
# # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positi
ve class
# # not the predicted outputs

# y_train_pred = neigh.predict_proba(X_tr)
# y_test_pred = neigh.predict_proba(X_te)
```

```
# train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred[:,1])
# test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred[:,1])

# x=[0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
```

In [118]:

```
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.plot(x,x)
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Confusion Matrix

In [119]:

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

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

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

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

In [120]:

```
print("Train confusion matrix")
conf_matr_df_train_2=pd.DataFrame(confusion_matrix(y_train,predict(y_train_pred,tr_thresholds,trai
n_fpr,train_fpr)),range(2),range(2))
sns.set(font_scale=1)#for label size
sns.heatmap(conf_matr_df_train_2,annot=True,annot_kws={"size":30},fmt='g')
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24999997805212623 for threshold 0.77
```

Out[120]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1881e95f898>
```

In [121]:

```
print("Test confusion matrix")
conf_matr_df_train_2=pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred,tr_thresholds,test_fp
r,test_fpr)),range(2),range(2))
sns.set(font_scale=1)#for label size
sns.heatmap(conf_matr_df_train_2,annot=True,annot_kws={"size":30},fmt='g')
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2499999593849979 for threshold 0.804
```

Out[121]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x188129c0828>
```

2.4.2 Applying Logistic Regression on TFIDF, SET 2

Creating Data Matrix

In [122]:

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


# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_tfidf,X_train_title_tfidf, X_train_state_ohe, X_train_teacher_ohe, X_t
rain_grade_ohe,X_train_cat_ohe,X_train_sub_ohe, X_train_price_norm,X_train_project_norm)).tocsr()
X_cr = hstack((X_cv_essay_tfidf,X_cv_title_tfidf, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe,
X_cv_cat_ohe,X_cv_sub_ohe, X_cv_price_norm,X_cv_project_norm)).tocsr()
X_te = hstack((X_test_essay_tfidf,X_test_title_tfidf, X_test_state_ohe, X_test_teacher_ohe,
X_test_grade_ohe,X_test_cat_ohe,X_test_sub_ohe, X_test_price_norm,X_test_project_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
(22445, 7111) (22445,)
(11055, 7111) (11055,)
(16500, 7111) (16500,)
==========================================================================================================
```

Hyperparameter Tuning: Simple for loop (if you are having memory limitations use this)

In [123]:

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

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

    return y_data_pred
```

In [124]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
log_alphas=[]
parameters = {'C':[0.0001,0.0005,0.001,0.005,0.01,0.05,0.1,0.5,1,2.5,5,7.5,10]}

for i in tqdm(parameters['C']):
    neigh = LogisticRegression(C=i)
    neigh.fit(X_tr, y_train)

    y_train_pred = batch_predict(neigh, X_tr)
    y_cv_pred = batch_predict(neigh, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
100%|██████████████████████████████████████████████████████████████████████████████████████████| 13/13
[00:43<00:00,  6.76s/it]
```

In [125]:

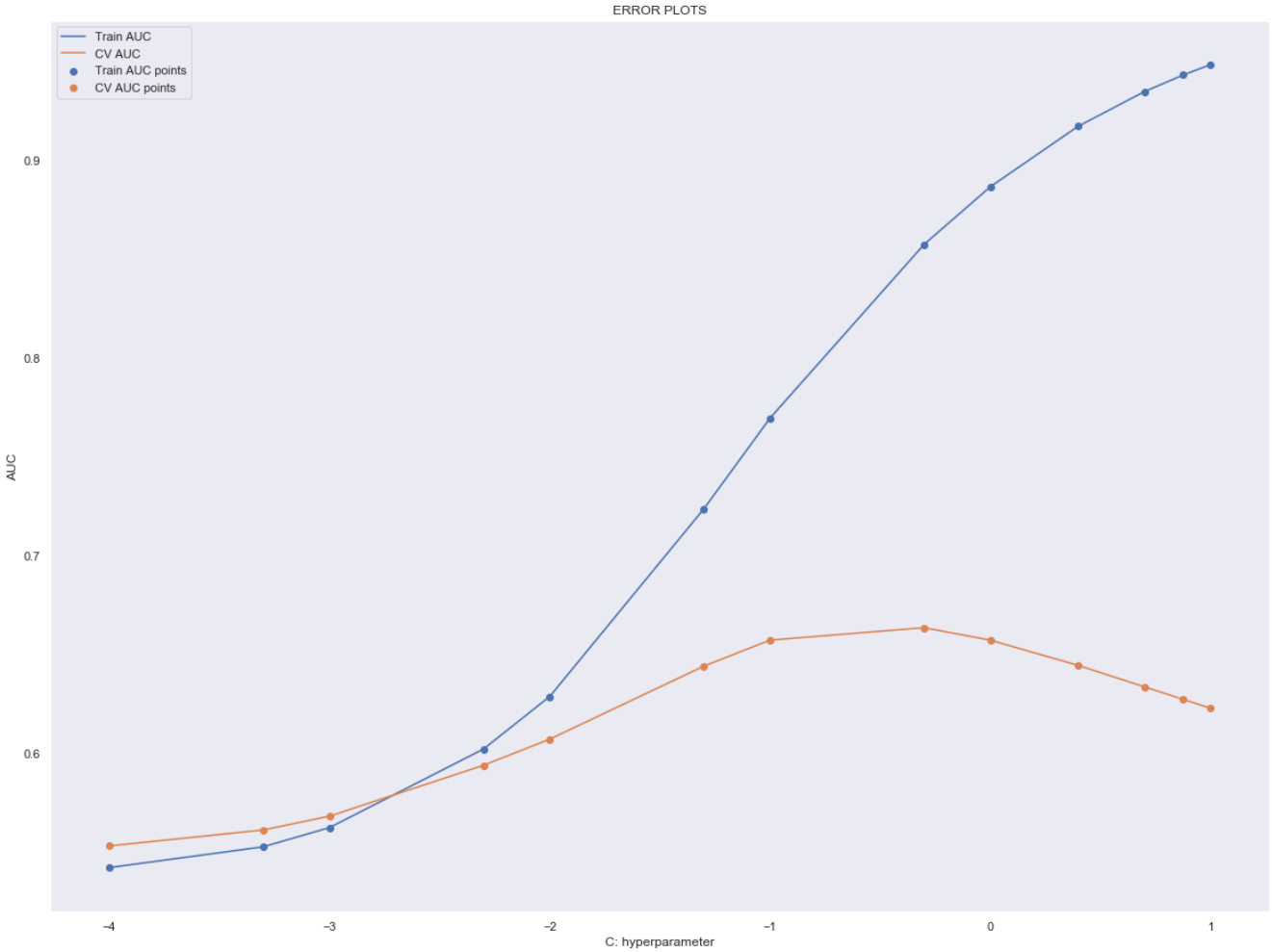
```
for a in tqdm(parameters['C']):
    b = math.log10(a)
    log_alphas.append(b)
print(log_alphas)
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 1
3/13 [00:00<?, ?it/s]
```

```
[-4.0, -3.3010299956639813, -3.0, -2.3010299956639813, -2.0, -1.3010299956639813, -1.0, -0.3010299
956639812, 0.0, 0.3979400086720376, 0.6989700043360189, 0.8750612633917001, 1.0]
```

In [126]:

```
plt.figure(figsize=(20,15))
plt.plot(log_alphas, train_auc, label='Train AUC')
plt.plot(log_alphas, cv_auc, label='CV AUC')

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

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



In [127]:

```
best_k=0.30
```

Train The Model

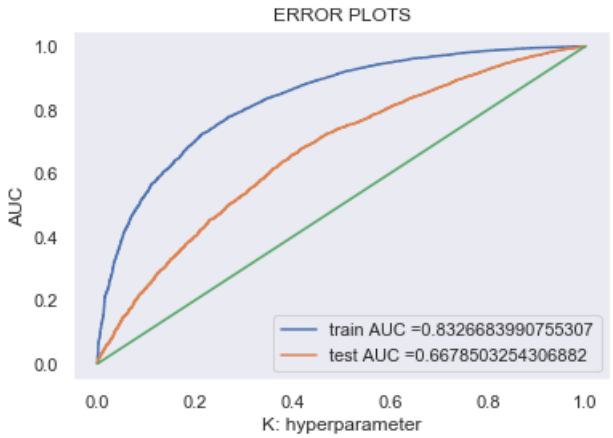
In [128]:

```
from sklearn.metrics import roc_curve, auc

neigh = LogisticRegression(C=best_k)
```

```
neigh.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, 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)

x=[0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
```

In [129]:

```
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.plot(x,x)
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Confusion Matrix

In [130]:

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

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

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

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

In [131]:

```
print("Train confusion matrix")
conf_matr_df_train_2=pd.DataFrame(confusion_matrix(y_train,predict(y_train_pred,tr_thresholds,trai
n_fpr,train_fpr)),range(2),range(2))
sns.set(font_scale=1)#for label size
sns.heatmap(conf_matr_df_train_2,annot=True,annot_kws={"size":30},fmt='g')
```

Train confusion matrix

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24999997805212623 for threshold 0.771
```

Out[131]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x188116c4a20>
```

In [132]:

```
print("Test confusion matrix")
conf_matr_df_train_2=pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred,tr_thresholds,test_fp
r,test_fpr)),range(2),range(2))
sns.set(font_scale=1)#for label size
sns.heatmap(conf_matr_df_train_2,annot=True,annot_kws={"size":30},fmt='g')
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2499999593849979 for threshold 0.805
```

Out[132]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18813a8d278>
```

2.4.3 Applying Logistic Regression on AVG W2V, SET 3

Creating Data Matrix

In [133]:

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

# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((avg_w2v_essay_train,avg_w2v_title_train, X_train_state_ohe, X_train_teacher_ohe, X_t
rain_grade_ohe,X_train_cat_ohe,X_train_sub_ohe, X_train_price_norm,X_train_project_norm)).tocsr()
X_cr = hstack((avg_w2v_essay_cv,avg_w2v_title_cv, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe,
X_cv_cat_ohe,X_cv_sub_ohe, X_cv_price_norm,X_cv_project_norm)).tocsr()
X_te = hstack((avg_w2v_essay_test,avg_w2v_title_test, X_test_state_ohe, X_test_teacher_ohe,
X_test_grade_ohe,X_test_cat_ohe,X_test_sub_ohe, X_test_price_norm,X_test_project_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
(22445, 701) (22445,)
(11055, 701) (11055,)
(16500, 701) (16500,)
====================================================================================================
```

Hyperparameter Tuning: Simple for loop (if you are having memory limitations use this)

In [134]:

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

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

    return y_data_pred
```

In [135]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
log_alphas=[]
parameters = {'C':[0.01,0.05,0.1,0.5,1,5,10,50,100,500,1000,5000,10000,50000]}

for i in tqdm(parameters['C']):
    neigh = LogisticRegression(C=i)
    neigh.fit(X_tr, y_train)

    y_train_pred = batch_predict(neigh, X_tr)
    y_cv_pred = batch_predict(neigh, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
100%|██████████| 14/14
[08:00<00:00, 53.23s/it]
```

In [136]:

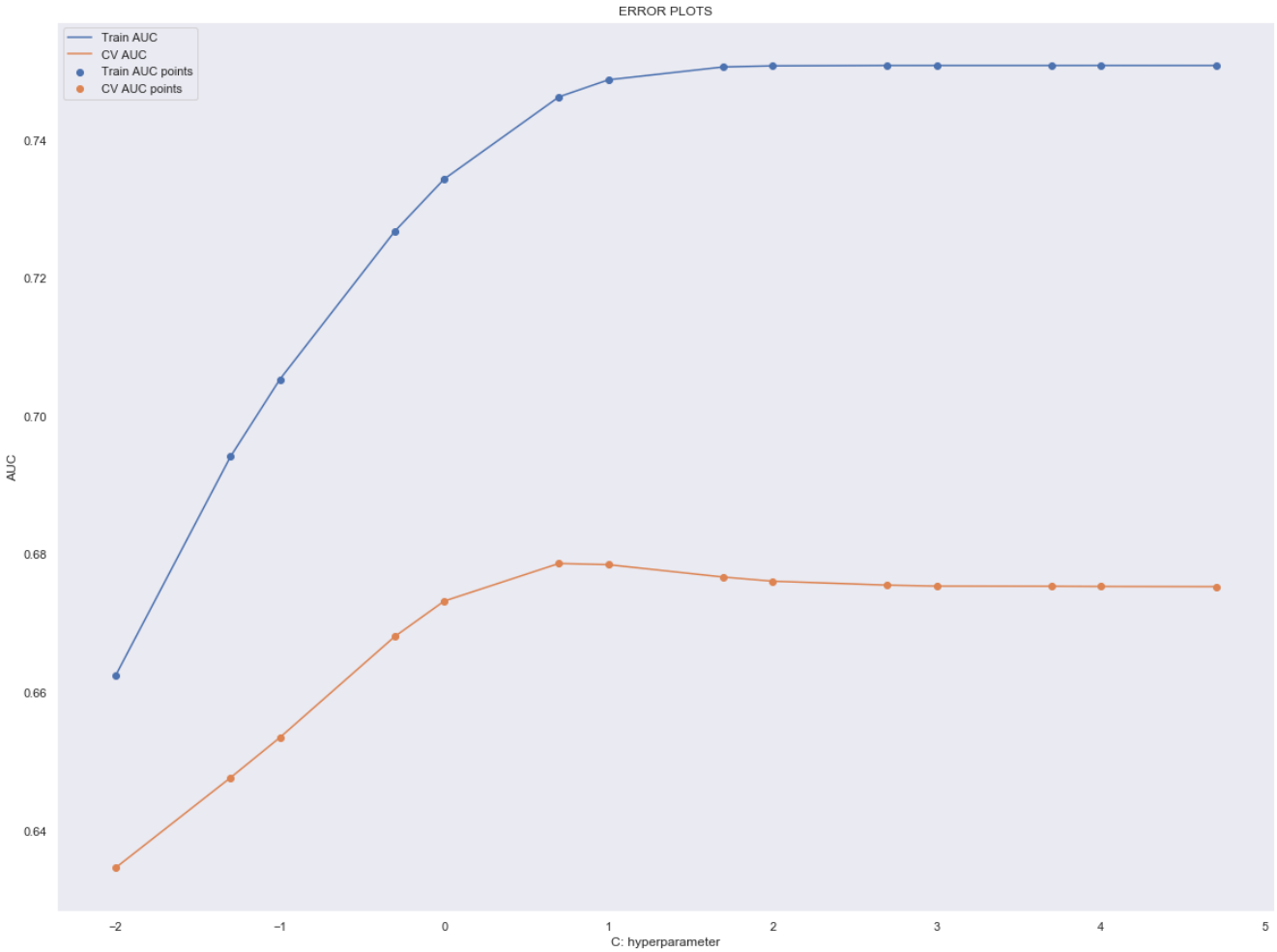
```
for a in tqdm(parameters['C']):
    b = math.log10(a)
    log_alphas.append(b)
print(log_alphas)
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 14/14
[00:00<00:00, 14051.27it/s]
```

```
[-2.0, -1.3010299956639813, -1.0, -0.3010299956639812, 0.0, 0.6989700043360189, 1.0,
1.6989700043360187, 2.0, 2.6989700043360187, 3.0, 3.6989700043360187, 4.0, 4.698970004336019]
```

In [137]:

```
plt.figure(figsize=(20,15))
plt.plot(log_alphas, train_auc, label='Train AUC')
plt.plot(log_alphas, cv_auc, label='CV AUC')

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

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



In [138]:

```
best_k=1
```

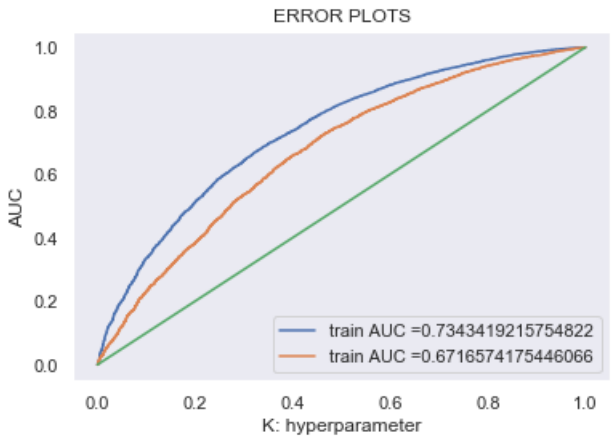
Train The Model

In [139]:

```
from sklearn.metrics import roc_curve, auc

neigh = LogisticRegression(C=best_k)
neigh.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, 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)
```

In [140]:

```
x=[0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr)))
plt.plot(x,x)
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Confusion Matrix

In [141]:

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

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

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

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

In [142]:

```
print("Train confusion matrix")
conf_matr_df_train_2=pd.DataFrame(confusion_matrix(y_train,predict(y_train_pred,tr_thresholds,trai
```

```
n_fpr,train_fpr)),range(2),range(2))
sns.set(font_scale=1)#for label size
sns.heatmap(conf_matr_df_train_2,annot=True,annot_kws={"size":30},fmt='g')
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2499999780521262 for threshold 0.791
```

Out[142]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1881e38ea58>
```

In [143]:

```
print("Test confusion matrix")
conf_matr_df_train_2=pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred,tr_thresholds,test_fp
r,test_fpr)),range(2),range(2))
sns.set(font_scale=1)#for label size
sns.heatmap(conf_matr_df_train_2,annot=True,annot_kws={"size":30},fmt='g')
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2499999593849979 for threshold 0.833
```

Out[143]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1881b578fd0>
```

2.4.4 Applying Logistic Regression on TFIDF W2V, SET 4

Creating Data Matrix

In [144]:

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

# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((tfidf_w2v_train_essay,tfidf_w2v_train_title, X_train_state_ohe, X_train_teacher_ohe
```

```
, X_train_grade_ohe,X_train_cat_ohe,X_train_sub_ohe, X_train_price_norm,X_train_project_norm)).toc
sr()
X_cr = hstack((tfidf_w2v_cv_essay,tfidf_w2v_cv_title, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_
ohe,X_cv_cat_ohe,X_cv_sub_ohe, X_cv_price_norm,X_cv_project_norm)).tocsr()
X_te = hstack((tfidf_w2v_test_essay,tfidf_w2v_test_title, X_test_state_ohe, X_test_teacher_ohe,
X_test_grade_ohe,X_test_cat_ohe,X_test_sub_ohe, X_test_price_norm,X_test_project_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
(22445, 701) (22445,)
(11055, 701) (11055,)
(16500, 701) (16500,)
====================================================================================================
```

Hyperparameter Tuning: Simple for loop (if you are having memory limitations use this)

In [145]:

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

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

    return y_data_pred
```

In [146]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
log_alphas=[]
parameters = {'C':[0.0001,0.0005,0.001,0.005,0.01,0.05,0.1,0.5,1,2.5,5,10,50,100,500]}

for i in tqdm(parameters['C']):
    neigh = LogisticRegression(C=i)
    neigh.fit(X_tr, y_train)

    y_train_pred = batch_predict(neigh, X_tr)
    y_cv_pred = batch_predict(neigh, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
```

```
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 15/15
[04:57<00:00, 41.63s/it]
```

In [147]:

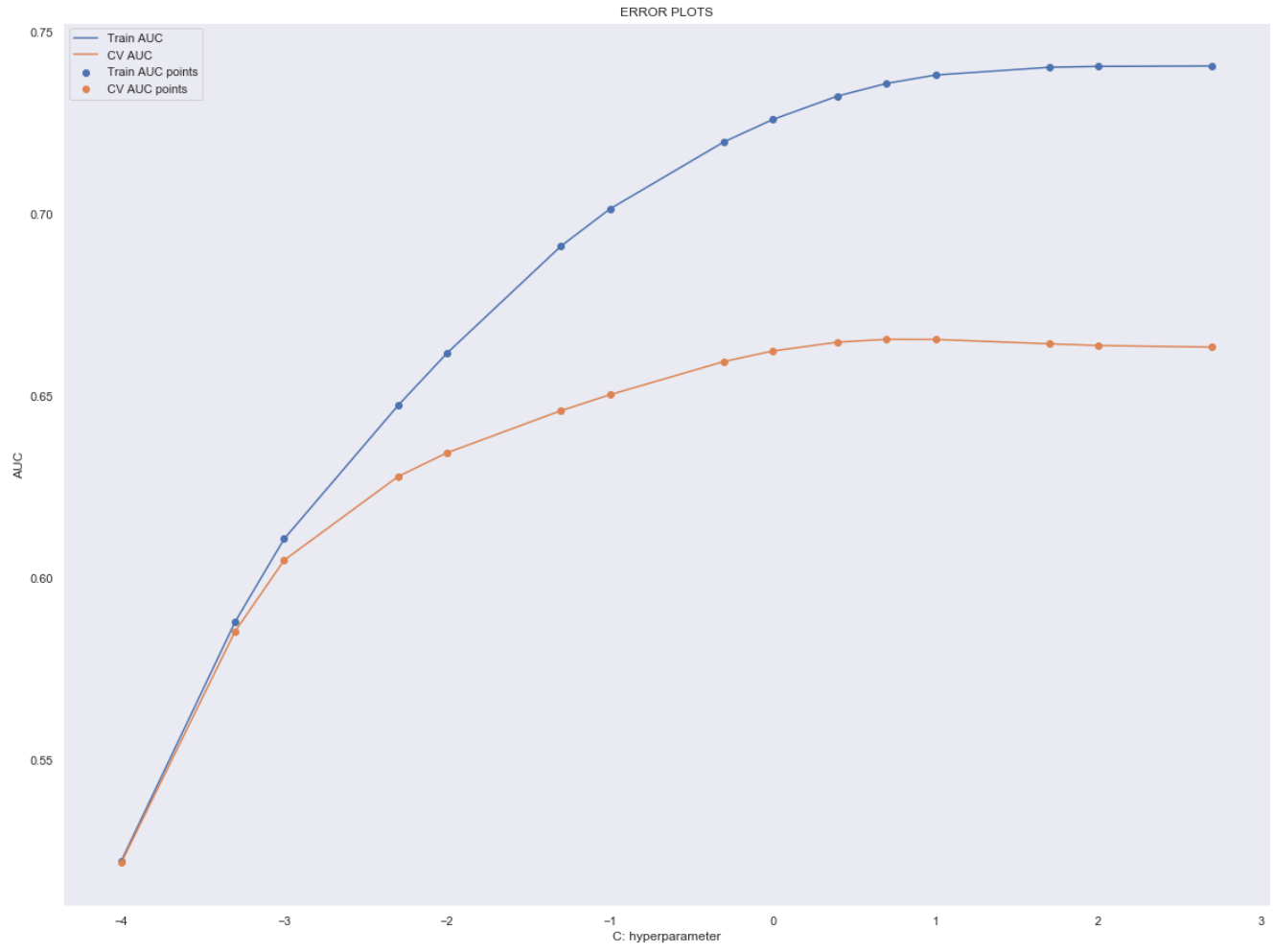
```
for a in tqdm(parameters['C']):
    b = math.log10(a)
    log_alphas.append(b)
print(log_alphas)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 1
5/15 [00:00<?, ?it/s]
```

```
[-4.0, -3.3010299956639813, -3.0, -2.3010299956639813, -2.0, -1.3010299956639813, -1.0, -0.3010299
956639812, 0.0, 0.3979400086720376, 0.6989700043360189, 1.0, 1.6989700043360187, 2.0,
2.6989700043360187]
```

In [148]:

```
plt.figure(figsize=(20,15))
plt.plot(log_alphas, train_auc, label='Train AUC')
plt.plot(log_alphas, cv_auc, label='CV AUC')

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

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



In [149]:

```
best_k=0.1
```

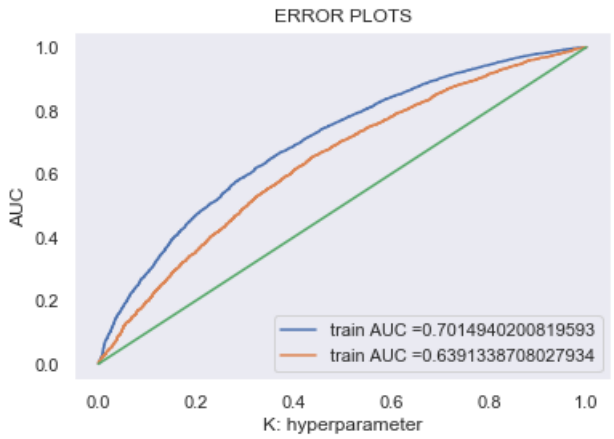
Train The Model

In [150]:

```
from sklearn.metrics import roc_curve, auc

neigh = LogisticRegression(C=best_k)
neigh.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, 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)
```

In [151]:

```
x=[0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr)))
plt.plot(x,x)
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Confusion Matrix

In [152]:

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

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

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

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

```
            predictions.append(0)
    return predictions
```

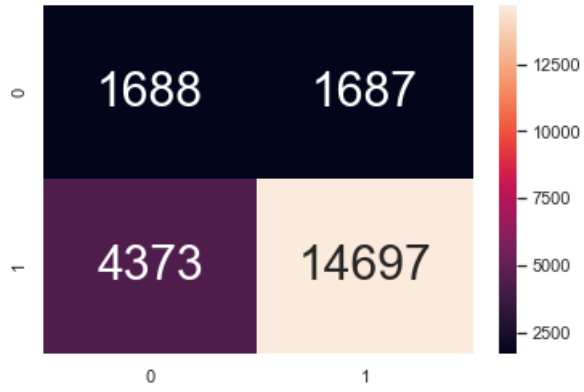
In [153]:

```
print("Train confusion matrix")
conf_matr_df_train_2=pd.DataFrame(confusion_matrix(y_train,predict(y_train_pred,tr_thresholds,trai
n_fpr,train_fpr)),range(2),range(2))
sns.set(font_scale=1)#for label size
sns.heatmap(conf_matr_df_train_2,annot=True,annot_kws={"size":30},fmt='g')
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24999997805212623 for threshold 0.815
```

Out[153]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1881f86cc18>
```



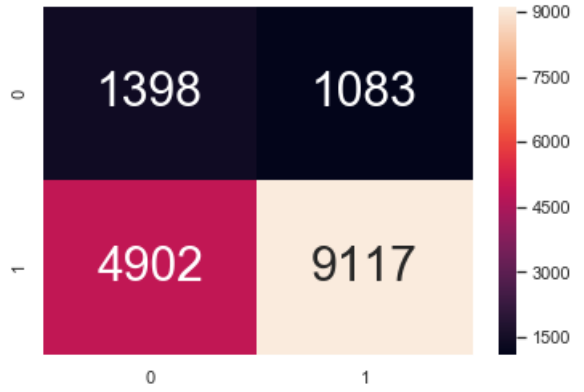
In [154]:

```
print("Test confusion matrix")
conf_matr_df_train_2=pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred,tr_thresholds,test_fp
r,test_fpr)),range(2),range(2))
sns.set(font_scale=1)#for label size
sns.heatmap(conf_matr_df_train_2,annot=True,annot_kws={"size":30},fmt='g')
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2499999593849979 for threshold 0.839
```

Out[154]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1881e38ef60>
```



2.5 Logistic Regression with added Features, SET 5

Creating Data Matrix

In [155]:

```
X_tr = hstack((X_train_state_ohe, X_train_teacher_ohe,
X_train_grade_ohe,X_train_cat_ohe,X_train_sub_ohe,
X_train_price_norm,X_train_project_norm,X_train_title_norm,X_train_essay_norm,essay_sent_pos_train
,essay_sent_neg_train,essay_sent_neu_train,essay_sent_comp_train)).tocsr()
X_cr = hstack((X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe,X_cv_cat_ohe,X_cv_sub_ohe, X_cv_pri
ce_norm,X_cv_project_norm,X_cv_title_norm,X_cv_essay_norm,essay_sent_pos_cv,essay_sent_neg_cv,essa
y_sent_neu_cv,essay_sent_comp_cv)).tocsr()
X_te = hstack((X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe,X_test_cat_ohe,X_test_sub_ohe
, X_test_price_norm,X_test_project_norm,X_test_title_norm,X_test_essay_norm,essay_sent_pos_test,es
say_sent_neg_test,essay_sent_neu_test,essay_sent_comp_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
(22445, 107) (22445,)
(11055, 107) (11055,)
(16500, 107) (16500,)
====================================================================================================
```

Hyperparameter Tuning: Simple for loop (if you are having memory limitations use this)

In [156]:

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

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

    return y_data_pred
```

In [157]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
log_alphas=[]

parameters = {'C':[0.0001,0.0005,0.001,0.005,0.01,0.05,0.1,0.5,1,2.5,5]}

for i in tqdm(parameters['C']):
    neigh = LogisticRegression(C=i)
    neigh.fit(X_tr, y_train)
```

```
    y_train_pred = batch_predict(neigh, X_tr)
    y_cv_pred = batch_predict(neigh, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
100%|██████████████████████████████████████████████████████████████████████████████████████████| 11/11
[00:02<00:00,  2.75it/s]
```

In [158]:

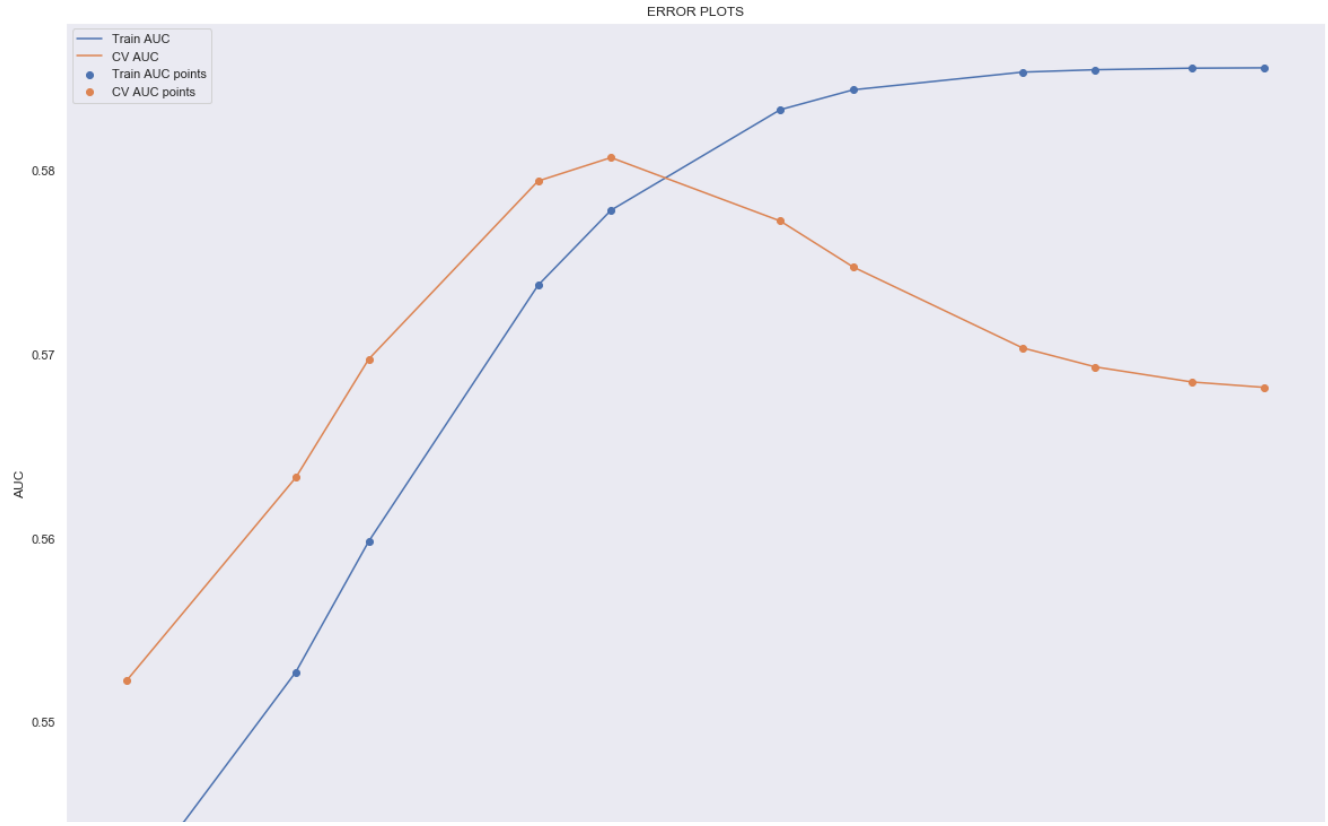
```
import math
for a in tqdm(parameters['C']):
    b = math.log10(a)
    log_alphas.append(b)
print(log_alphas)
```

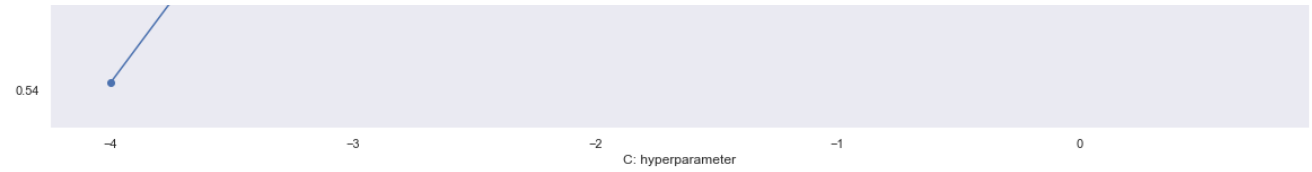
```
100%|██████████████████████████████████████████████████████████████████████████████████████████| 1
1/11 [00:00<?, ?it/s]
```

```
[-4.0, -3.3010299956639813, -3.0, -2.3010299956639813, -2.0, -1.3010299956639813, -1.0, -0.3010299
956639812, 0.0, 0.3979400086720376, 0.6989700043360189]
```

In [159]:

```
plt.figure(figsize=(20,15))
plt.plot(log_alphas, train_auc, label='Train AUC')
plt.plot(log_alphas, cv_auc, label='CV AUC')

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

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





In [160]:

```
best_k=0.01
```

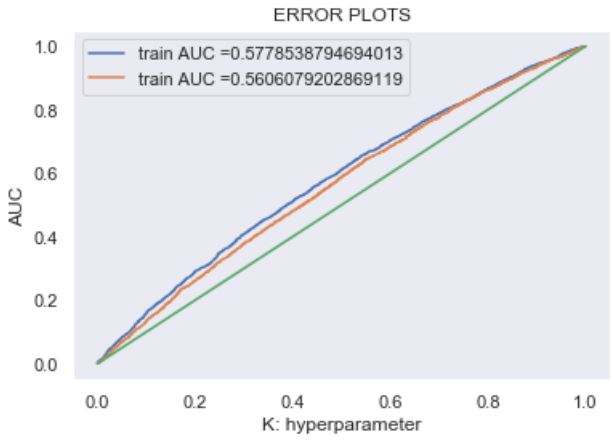
Train The Model

In [161]:

```
from sklearn.metrics import roc_curve, auc

neigh = LogisticRegression(C=best_k)
neigh.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, 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)
```

In [162]:

```
x=[0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr)))
plt.plot(x,x)
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Confusion Matrix

In [163]:

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

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

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

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

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

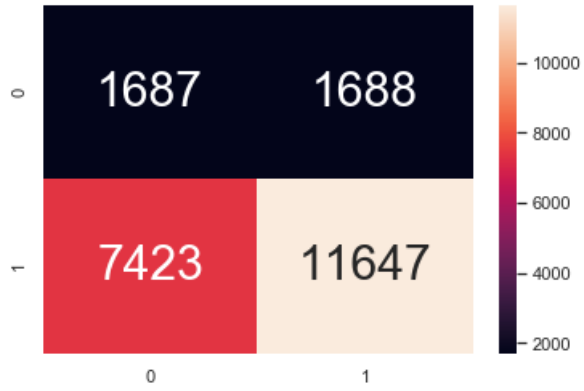
In [164]:

```
print("Train confusion matrix")
conf_matr_df_train_2=pd.DataFrame(confusion_matrix(y_train,predict(y_train_pred,tr_thresholds,train_fpr,train_fpr)),range(2),range(2))
sns.set(font_scale=1)#for label size
sns.heatmap(conf_matr_df_train_2,annot=True,annot_kws={"size":30},fmt='g')
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2499999780521262 for threshold 0.845
```

Out[164]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18813870fd0>
```



In [165]:

```
print("Test confusion matrix")
conf_matr_df_train_2=pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred,tr_thresholds,test_fpr,test_fpr)),range(2),range(2))
sns.set(font_scale=1)#for label size
sns.heatmap(conf_matr_df_train_2,annot=True,annot_kws={"size":30},fmt='g')
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2499999593849979 for threshold 0.854
```

Out[165]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1881f49a4a8>
```

3. Conclusion

In [166]:

```
# Please compare all your mod# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable
#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable
x=PrettyTable()
x.field_names=["Vectorizer","Hyper Parameter","AUC"]
x.add_row(["BOW",0.01,0.66])
x.add_row(["TFIDF",0.32,0.65])
x.add_row(["AVG W2V",1,0.69])
x.add_row(["TFIDF W2V",0.1,0.68])
x.add_row(["WITHOUT TEXT",0.01,0.57])
print(x)
```

```
+--------------+-----------------+------+
|  Vectorizer  | Hyper Parameter | AUC  |
+--------------+-----------------+------+
|     BOW      |       0.01      | 0.66 |
|    TFIDF     |       0.32      | 0.65 |
|   AVG W2V    |        1        | 0.69 |
|  TFIDF W2V   |       0.1       | 0.68 |
| WITHOUT TEXT |       0.01      | 0.57 |
+--------------+-----------------+------+
```

INFERENCE:

Here we can be observed that "Essays" and "Project Titles" play a major role in predicting the outcome of the project. So,we should not neglect them as the top four models containing them proved to have a better AUC score.

In [ ]: