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

import pickle

from tqdm import tqdm
import os

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

1.1 Reading Data

In [2]:

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

In [3]:

```
project_data.shape
```

Out[3]:

```
(109248, 17)
```

In [4]:

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

Out[4]:

```
(50000, 17)
```

In [5]:

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

Out[5]:

```
1    42447
0     7553
Name: project_is_approved, dtype: int64
```

In [6]:

```
resource_data.shape
```

Out[6]:

```
(1541272, 4)
```

In [7]:

```
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 (50000, 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 [8]:

```
# 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[8]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_s
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-27 01:10:09	Grades 3-5	Math &
79026	139722	p182545	22460c54072bd0cf958cc8349fac8b8f	Ms.	CA	2016-04-27 02:02:27	Grades 3-5	Math &

In [9]:

```
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[9]:

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

1.2 preprocessing of `project_subject_categories`

In [10]:

In [10]:

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

```
29891    Math & Science, History & Civics
79026    Math & Science, History & Civics
86551                        Math & Science
49228                Literacy & Language
7176     Math & Science, Applied Learning
Name: project_subject_categories, dtype: object
```

In [11]:

```
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 [12]:

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

```
29891     Math_Science History_Civics
79026     Math_Science History_Civics
86551                     Math_Science
49228                Literacy_Language
7176     Math_Science AppliedLearning
Name: clean_categories, dtype: object
```

1.3 preprocessing of `project_subject_subcategories`

In [13]:

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

```
29891              Mathematics, Social Sciences
79026    Applied Sciences, History & Geography
86551             Applied Sciences, Mathematics
49228                                 Literacy
7176        Applied Sciences, Early Development
Name: project_subject_subcategories, dtype: object
```

In [14]:

```
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 [15]:

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

```
29891           Mathematics SocialSciences
79026    AppliedSciences History_Geography
86551          AppliedSciences Mathematics
49228                              Literacy
7176      AppliedSciences EarlyDevelopment
Name: clean_subcategories, dtype: object
```

1.4 preprocessing of `school_state`

In [16]:

```
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 [17]:

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

```
29891    CA
79026    CA
86551    HI
49228    IL
7176     OH
Name: school_state, dtype: object
```

1.5 preprocessing of `project_grade_category`

In [18]:

```
preproc = []
# tqdm is for printing the status bar
for sent in project_data['project_grade_category']:
    sent = sent.replace('Grades ', '')
    sent = sent.replace('PreK-2', 'PreKto2')
    sent = sent.replace('3-5', '3to5')
    sent = sent.replace('6-8', '6to8')
    sent = sent.replace('9-12', '9to12')
    preproc.append(sent)
project_data['project_grade_category']=preproc
```

In [19]:

```
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 [20]:

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

```
29891       3to5
79026       3to5
86551       3to5
49228    PreKto2
7176     PreKto2
Name: project_grade_category, dtype: object
```

1.6 preprocessing of `teacher_prefix`

In [21]:

```
print(type(project_data['teacher_prefix']))
```

```
<class 'pandas.core.series.Series'>
```

In [22]:

```
project_data['teacher_prefix'] = project_data['teacher_prefix'].astype(str)
preproc = []
# tqdm is for printing the status bar
for sent in project_data['teacher_prefix']:
    sent = sent.replace('Mr.', 'Mr')
    sent = sent.replace('Mrs.', 'Mrs')
    sent = sent.replace('Dr.', 'Dr')
    sent = sent.replace('Ms.', 'Ms')
    preproc.append(sent)
project_data['teacher_prefix']=preproc
```

In [23]:

```
#['Teacher', 'Mrs.', 'Dr.', 'Mr.', 'Ms.']
project_data['teacher_prefix']=project_data['teacher_prefix'].fillna('')
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 [24]:

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

```
29891    Mrs
79026     Ms
86551    Mrs
49228     Ms
7176     Mrs
Name: teacher_prefix, dtype: object
```

1.3 Preprocessing of Essays

In [25]:

```
# 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 [26]:

```
# 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 [27]:

```
# 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 [28]:

```
# 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.lower() not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████| 50000/50000 [00:29<00:00, 1689.03it/s]
```

In [29]:

```
# after preprocesing
preprocessed_essays[2000]
```

Out[29]:

```
'project feeder elementary school music programs several elementary schools feed independence high
school excited help students elementary schools expand musical language using tone chimes students
elementary schools able use chimes play existing music create outstanding teachers feeder elementa
ry schools excited prospect equipment opportunity project designed provide much needed piece music
al equipment elementary school students east frisco tone chimes teach musical literacy
understanding also provide much needed piece equipment help students need tactile experience
better learn project aims provide one set tone chimes used elementary music teachers area rotating
basis elementary music program would designated period year would get use equipment make music
understand concepts pitch rhythm meter well dynamics articulation nannan'
```

In [30]:

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

1.4 Preprocessing of `project_title`

In [31]:

```
# 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 [00:01<00:00, 37299.57it/s]
```

In [32]:

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

Sentiment Analysis of essays

In [33]:

```
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
[nltk_data]     /home/samthekiller/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
100%|██████████| 50000/50000 [05:16<00:00, 157.80it/s]
```

In [34]:

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

In [35]:

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

In [36]:

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

In [37]:

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

Number of Words in Title

In [38]:

```
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 [39]:

```
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
```

1.5 Preparing data for models

In [40]:

```
project_data.columns
```

Out[40]:

```
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

In [41]:

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

In [42]:

```
# words = []
# for i in preprocessed_essays:
#     words.extend(i.split(' '))

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

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

# words_courpus = {}
# words_glove = set(model.keys())
```

```
# for i in words:
#     if i in words_glove:
#         words_courpus[i] = model[i]
# print("word 2 vec length", len(words_courpus))


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

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

In [43]:

```
# 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())
```

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.
    - 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-leakag, 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 [44]:

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

```
(50000,)
```

In [45]:

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

In [46]:

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

```
(50000, 23)
```

In [47]:

```
#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 eassay, and project_title

In [48]:

```
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 [49]:

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

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

BOW of Essay

In [50]:

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

In [51]:

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

Out[51]:

```
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 [52]:

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

In [53]:

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

In [54]:

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

In [55]:

```
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 [56]:

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

In [57]:

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

Out[57]:

```
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 [58]:

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

In [59]:

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

In [60]:

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

In [61]:

```
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, 1969) (22445,)
(11055, 1969) (11055,)
(16500, 1969) (16500,)
====================================================================================================
```

TFIDF of Essay

In [62]:

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

In [63]:

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

Out[63]:

```
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=(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 [64]:

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

In [65]:

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

In [66]:

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

In [67]:

```
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 [68]:

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

In [69]:

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

Out[69]:

```
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 [70]:

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

In [71]:

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

In [72]:

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

In [73]:

```
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, 1969) (22445,)
(11055, 1969) (11055,)
(16500, 1969) (16500,)
====================================================================================================
```

Avg W2V of Essay

In [74]:

```
# 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 [00:05<00:00, 3771.90it/s]
```

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

In [75]:

```
# 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 [00:04<00:00, 3883.26it/s]
```

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

In [76]:

```
# 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 [00:02<00:00, 3926.66it/s]
```

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

Avg W2V of Title

In [77]:

```
# 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, 63805.13it/s]
```

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

In [78]:

```
# 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, 69882.90it/s]
```

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

In [79]:

```
# 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, 65638.61it/s]
```

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

TFIDF-W2V of Essay

In [80]:

```
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 [81]:

```
# 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 [00:36<00:00, 612.48it/s]
```

```
22445
300
```

In [82]:

```
# 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 [00:27<00:00, 602.01it/s]
```

```
16500
300
```

In [83]:

```
# 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 [00:18<00:00, 605.17it/s]
```

```
11055
300
```

TFIDF-W2V of Title

In [84]:

```
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 [85]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_train_title = []; # the avg-w2v for each sentence/review is stored in this list
```

```
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 [00:00<00:00, 32275.77it/s]
```

```
22445
300
```

In [86]:

```
# 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 [00:00<00:00, 31829.48it/s]
```

```
16500
300
```

In [87]:

```
# 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, 30726.58it/s]
```

```
11055
300
```

2.3 Make Data Model Ready: encoding numerical and categorical features

Vectorizing Numerical features

In [88]:

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

In [89]:

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

Out[89]:

	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 [90]:

```
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 [91]:

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

Normalizing the numerical features: Price

In [92]:

```
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 [93]:

```
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 [94]:

```
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 [95]:

```
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 [96]:

```
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 [97]:

```
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 [98]:

```
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 [99]:

```
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 [100]:

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

In [101]:

```
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,)
['9to12', '6to8', '3to5', 'PreKto2']
==========================================================================================
```

Vectorizing Categorical features: teacher prefix

In [102]:

```
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, 6) (22445,)
(11055, 6) (11055,)
(16500, 6) (16500,)
['nan', 'Dr', 'Teacher', 'Mr', 'Ms', 'Mrs']
==========================================================================================
```

Vectorizing Categorical features: school state

In [103]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_state_dict.keys()), lowercase=False, binary=Tr
ue)
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)

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

Vectorizing Categorical features: clean categories

In [104]:

```
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 [105]:

```
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',
'PerformingArts', 'SocialSciences', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'History_Geography', 'Music', 'Health_LifeScience', 'ESL', 'Gym_Fitness', 'E
arlyDevelopment', '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 instructions

Applying Logistic Regression on BOW, SET 1

Applying Logistic Regression on BOW, SET 1

Creating Data Matrix

In [106]:

```
# 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, 7071) (22445,)
(11055, 7071) (11055,)
(16500, 7071) (16500,)
====================================================================================================
```

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

In [107]:

```
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 [108]:

```
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 [00:13<00:00,  2.50s/it]
```

In [109]:

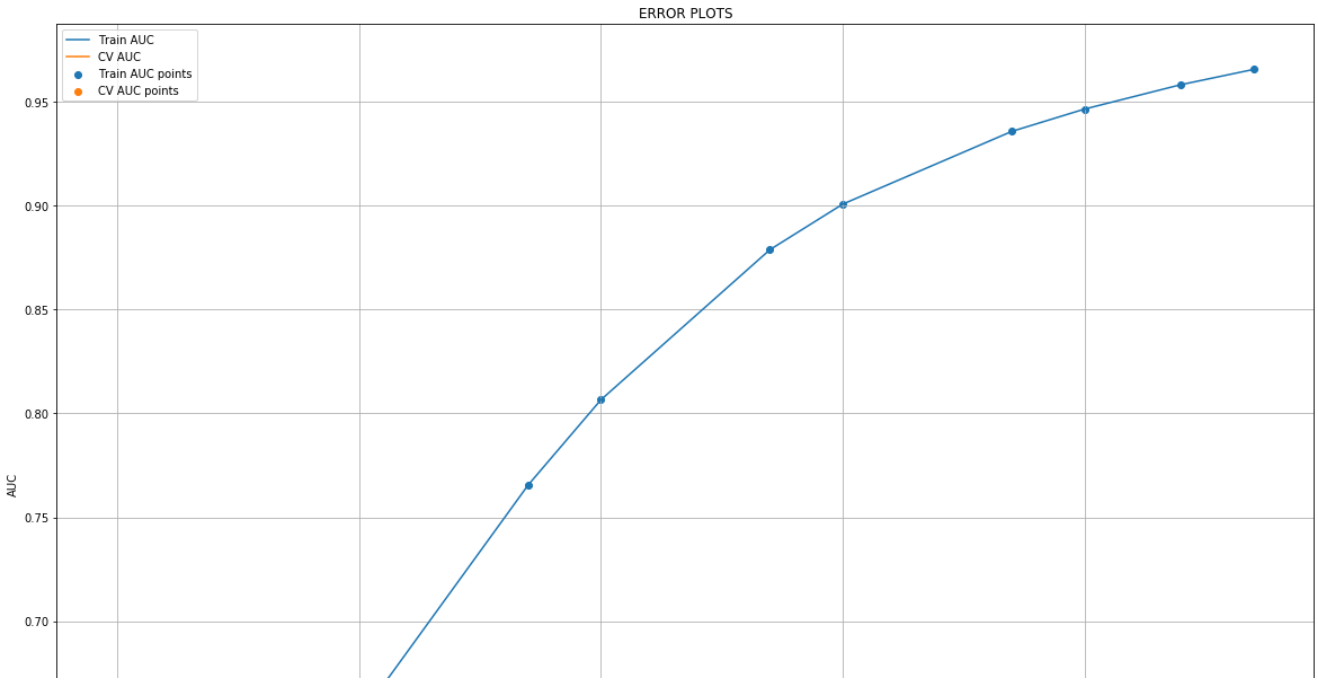
```
import math
for a in tqdm(parameters['C']):
    b = math.log10(a)
    log_alphas.append(b)
print(log_alphas)
```

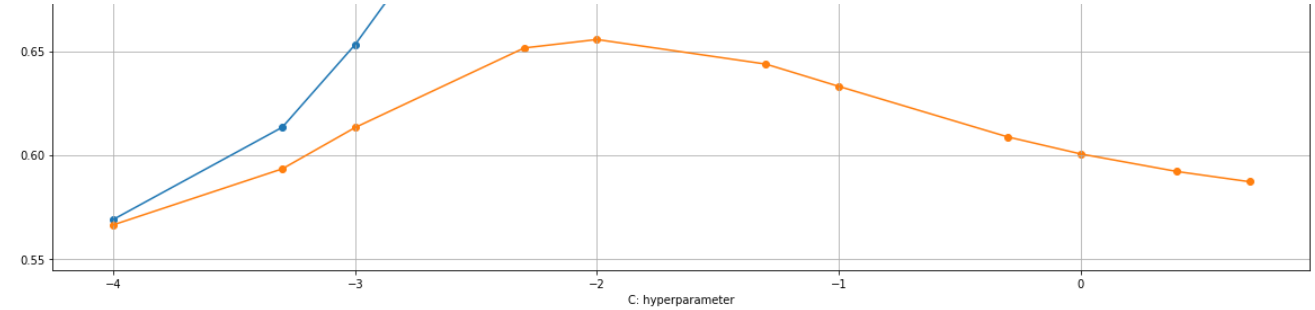
```
100%|██████████| 11/11 [00:00<00:00, 4753.98it/s]
```

```
[-4.0, -3.3010299956639813, -3.0, -2.3010299956639813, -2.0, -1.3010299956639813, -1.0, -0.3010299
956639812, 0.0, 0.39794000867203776, 0.6989700043360189]
```

In [110]:

```
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 [111]:

```
best_k=0.01
```

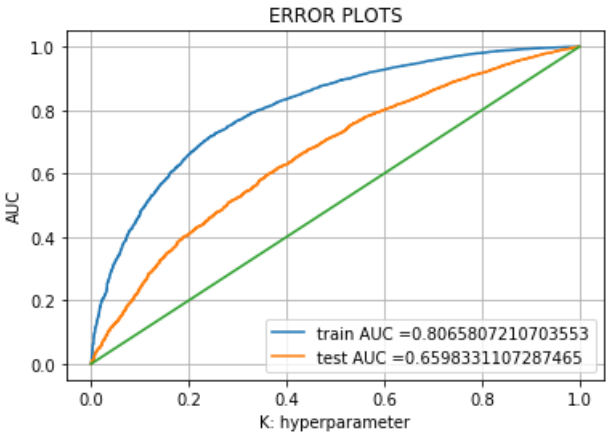
Train The Model

In [112]:

```
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 [113]:

```
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 [114]:

```
# we are writing our own function for predict, with defined threshold
```

```
# we are writing our own function for predict, with defined threshould
# 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 [115]:

```
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) 0.24999997825875406 for threshold 0.788
```

Out[115]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8474b91320>
```

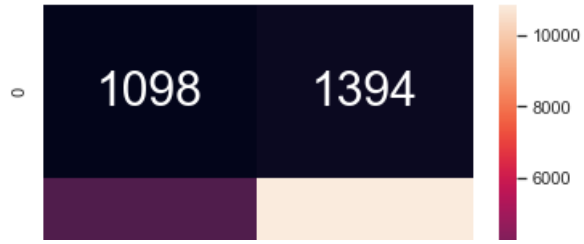
In [116]:

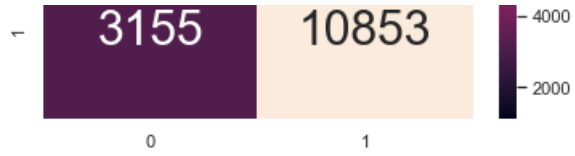
```
print("Test confusion matrix")
conf_matr_df_train_2=pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred,tr_thresholds,test_fp
r,test_fpr)),range(2),range(2))
sns.set(font_scale=1)#for label size
sns.heatmap(conf_matr_df_train_2,annot=True,annot_kws={"size":30},fmt='g')
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999983897106373 for threshold 0.812
```

Out[116]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8474c2ae80>
```





2.4.2 Applying Logistic Regression on TFIDF, SET 2

Creating Data Matrix

In [117]:

```
# 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, 7071) (22445,)
(11055, 7071) (11055,)
(16500, 7071) (16500,)
====================================================================================================
```

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

In [118]:

```
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 [119]:

```
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
```

```
# 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 [00:20<00:00,  3.21s/it]
```

In [120]:

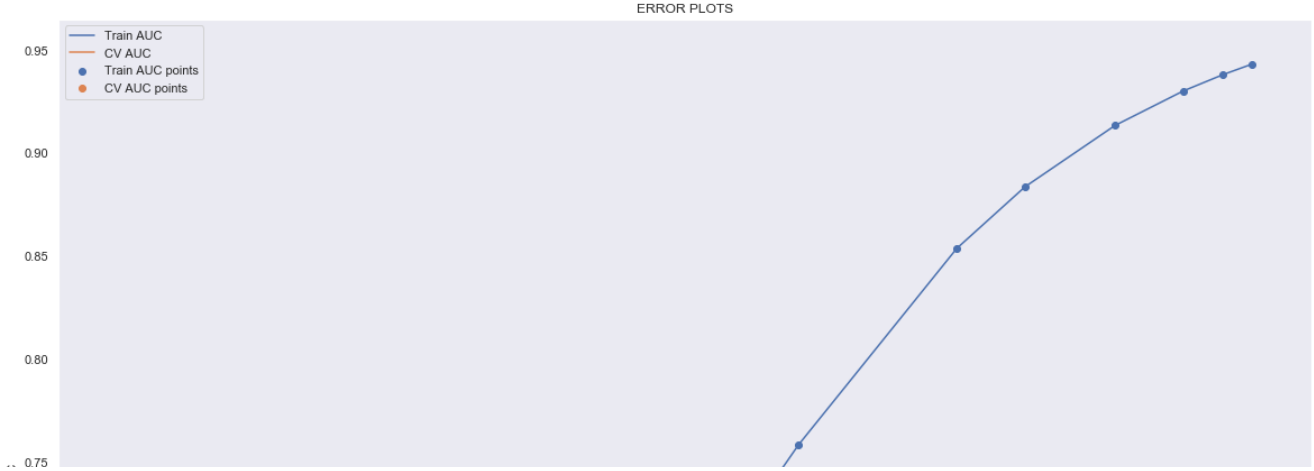
```
for a in tqdm(parameters['C']):
    b = math.log10(a)
    log_alphas.append(b)
print(log_alphas)
```

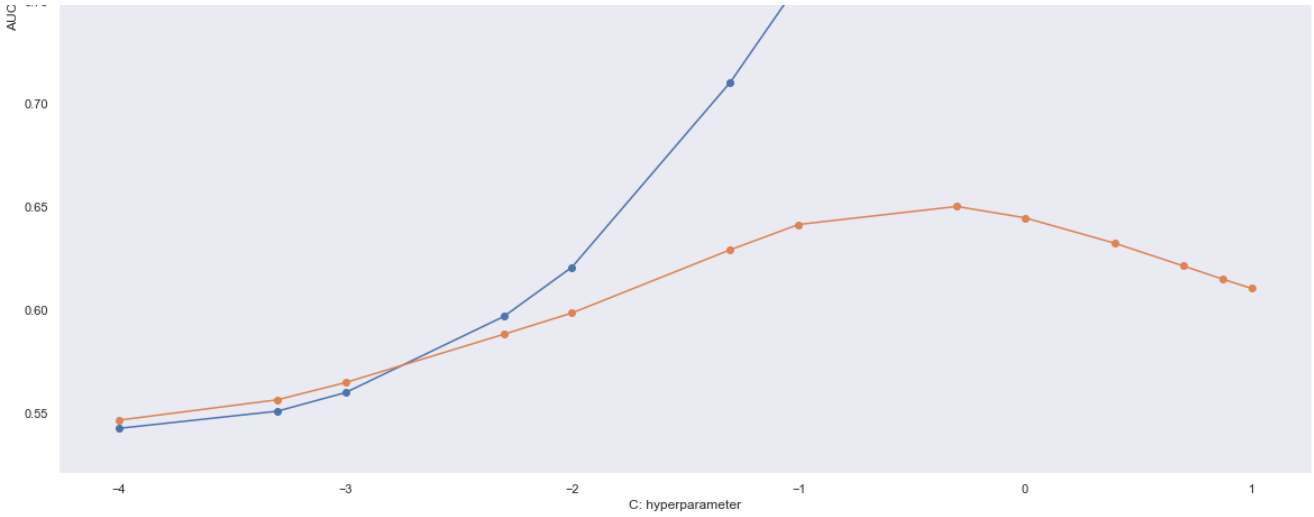
```
100%|██████████| 13/13 [00:00<00:00, 32689.42it/s]
```

```
[-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 [121]:

```
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 [122]:

```
best_k=0.32
```

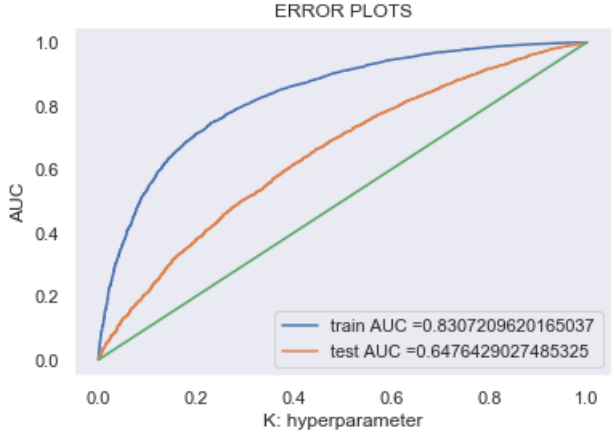
Train The Model

In [123]:

```
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 [124]:

```
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="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 [125]:

```
# 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 [126]:

```
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) 0.24999997825875406 for threshold 0.776
```

Out[126]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8474b2a5c0>
```

In [127]:

```
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) 0.25 for threshold 0.805
```

Out[127]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f84794a9e48>
```

2.4.3 Applying Logistic Regression on AVG W2V, SET 3

Creating Data Matrix

In [128]:

```
# 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, 702) (22445,)
(11055, 702) (11055,)
(16500, 702) (16500,)
====================================================================================================
```

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

In [129]:

```
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 [136]:

```
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 [02:56<00:00, 17.38s/it]
```

In [137]:

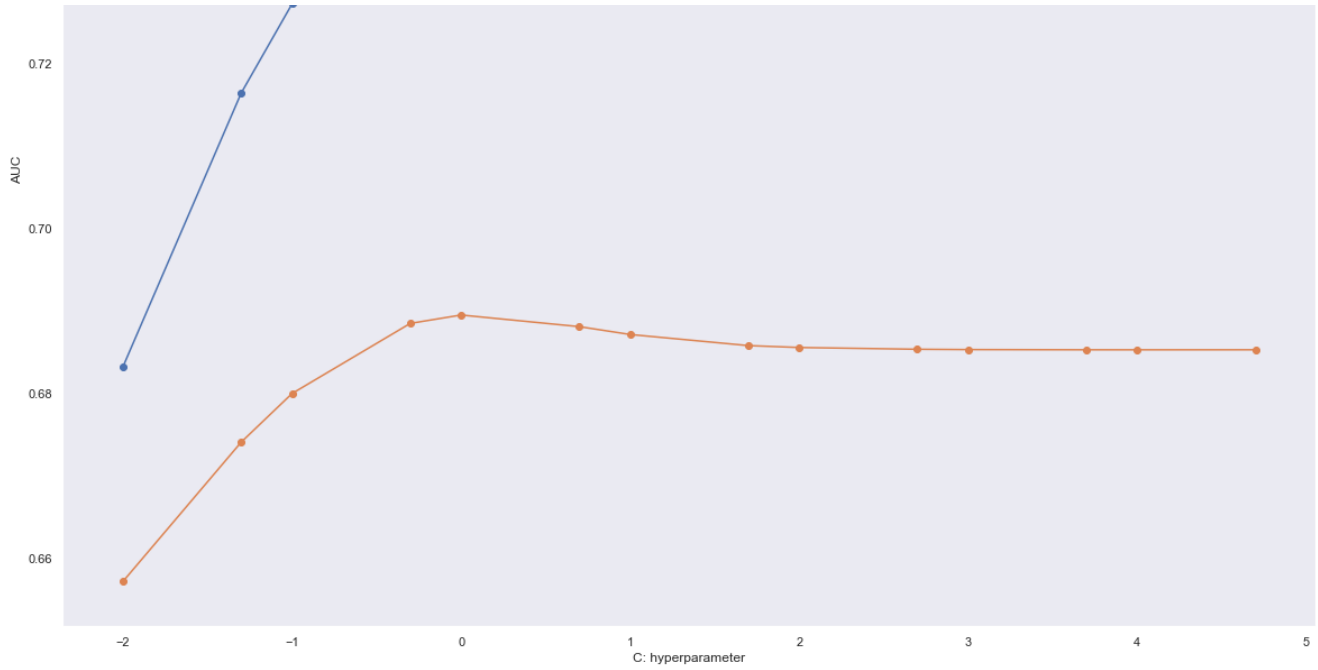
```
for a in tqdm(parameters['C']):
    b = math.log10(a)
    log_alphas.append(b)
print(log_alphas)
```

```
100%|██████████| 14/14 [00:00<00:00, 14118.84it/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 [138]:

```
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 [139]:

```
best_k=1
```

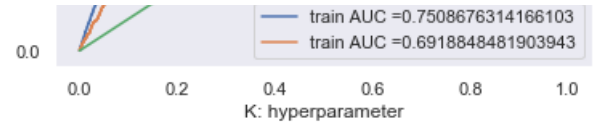
Train The Model

In [140]:

```
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 [141]:

```
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 [142]:

```
# 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 [143]:

```
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) 0.24999997825875406 for threshold 0.778
```

Out[143]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8474cb4908>
```

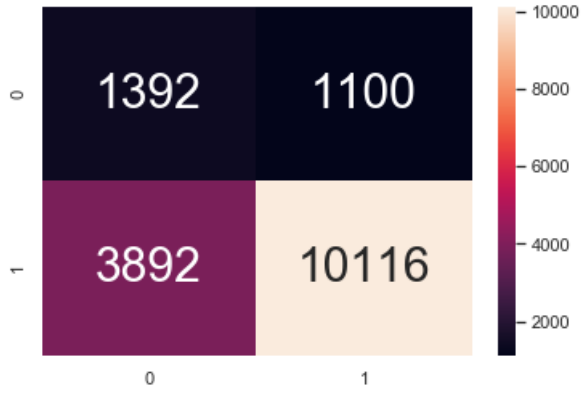
In [144]:

```
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) 0.24999983897106373 for threshold 0.826
```

Out[144]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8474a98b38>
```



2.4.4 Applying Logistic Regression on TFIDF W2V, SET 4

Creating Data Matrix

In [145]:

```
# 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, 702) (22445,)
(11055, 702) (11055,)
(16500, 702) (16500,)
====================================================================================================
```

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

In [146]:

```
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 [151]:

```
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 [01:41<00:00, 13.58s/it]
```

In [152]:

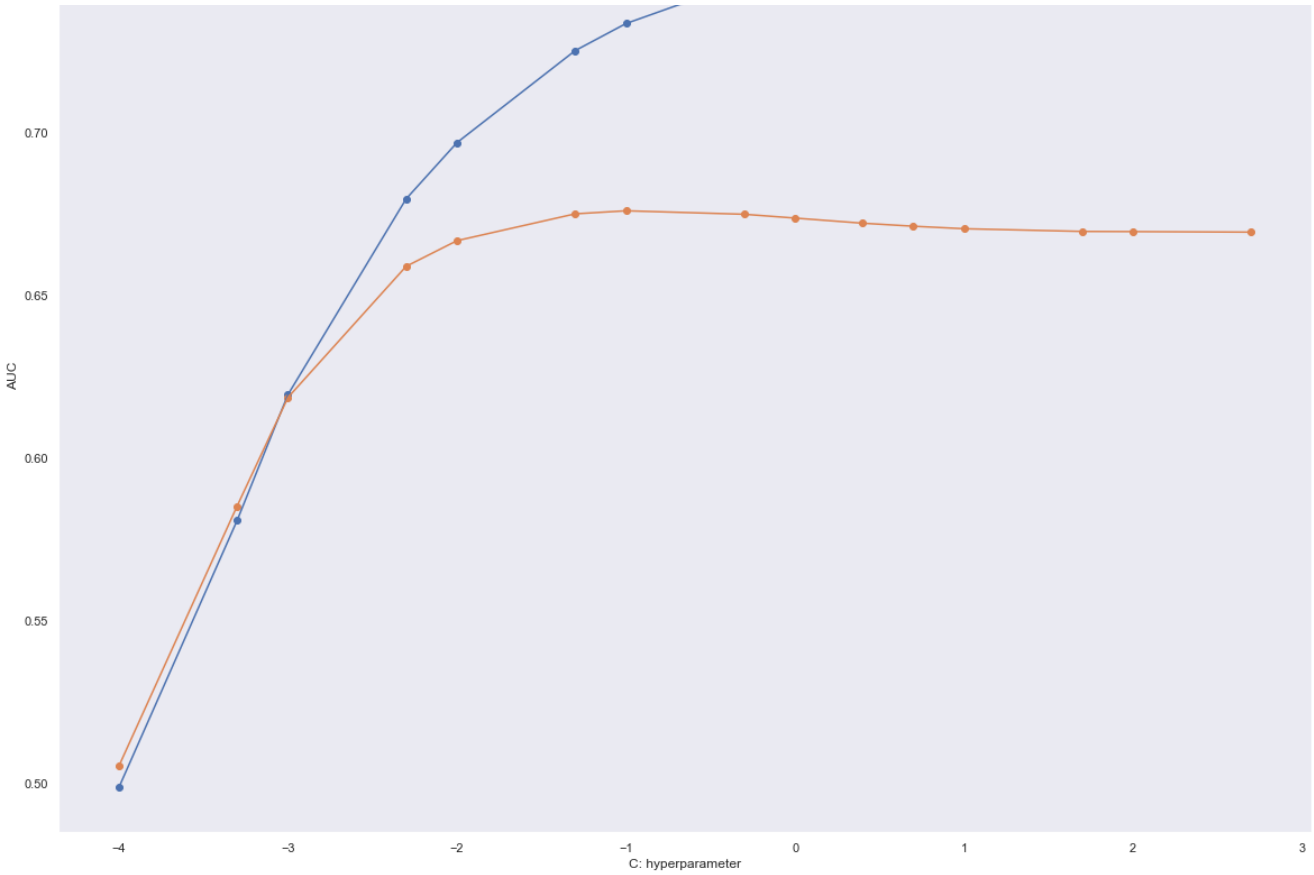
```
for a in tqdm(parameters['C']):
    b = math.log10(a)
    log_alphas.append(b)
print(log_alphas)
```

```
100%|██████████| 15/15 [00:00<00:00, 24413.88it/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 [153]:

```
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=0.1
```

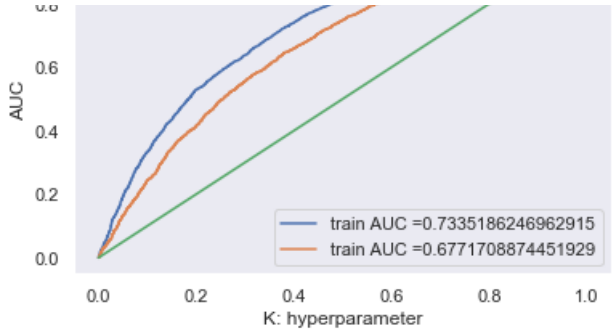
Train The Model

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

    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,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) 0.24999997825875406 for threshold 0.794
```

Out[164]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8479490198>
```

In [165]:

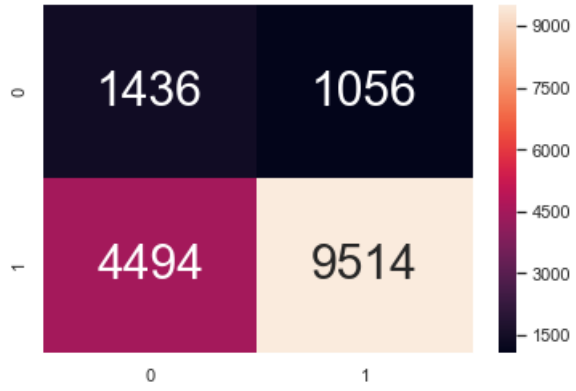
```
print("Test confusion matrix")
conf_matr_df_train_2=pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred,tr_thresholds,test_fp
r,test_fpr)),range(2),range(2))
sns.set(font_scale=1)#for label size
```

```
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) 0.25 for threshold 0.832
```

Out[165]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8474a6d358>
```



Set 5 : Logistic Regression on Categorical features, Numerical features & Essay Sentiments

Creating Data Matrix

In [166]:

```
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, 108) (22445,)
(11055, 108) (11055,)
(16500, 108) (16500,)
====================================================================================================
```

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

In [167]:

```
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
```

```
    # 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 [176]:

```
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 [00:07<00:00,  1.01it/s]
```

In [177]:

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

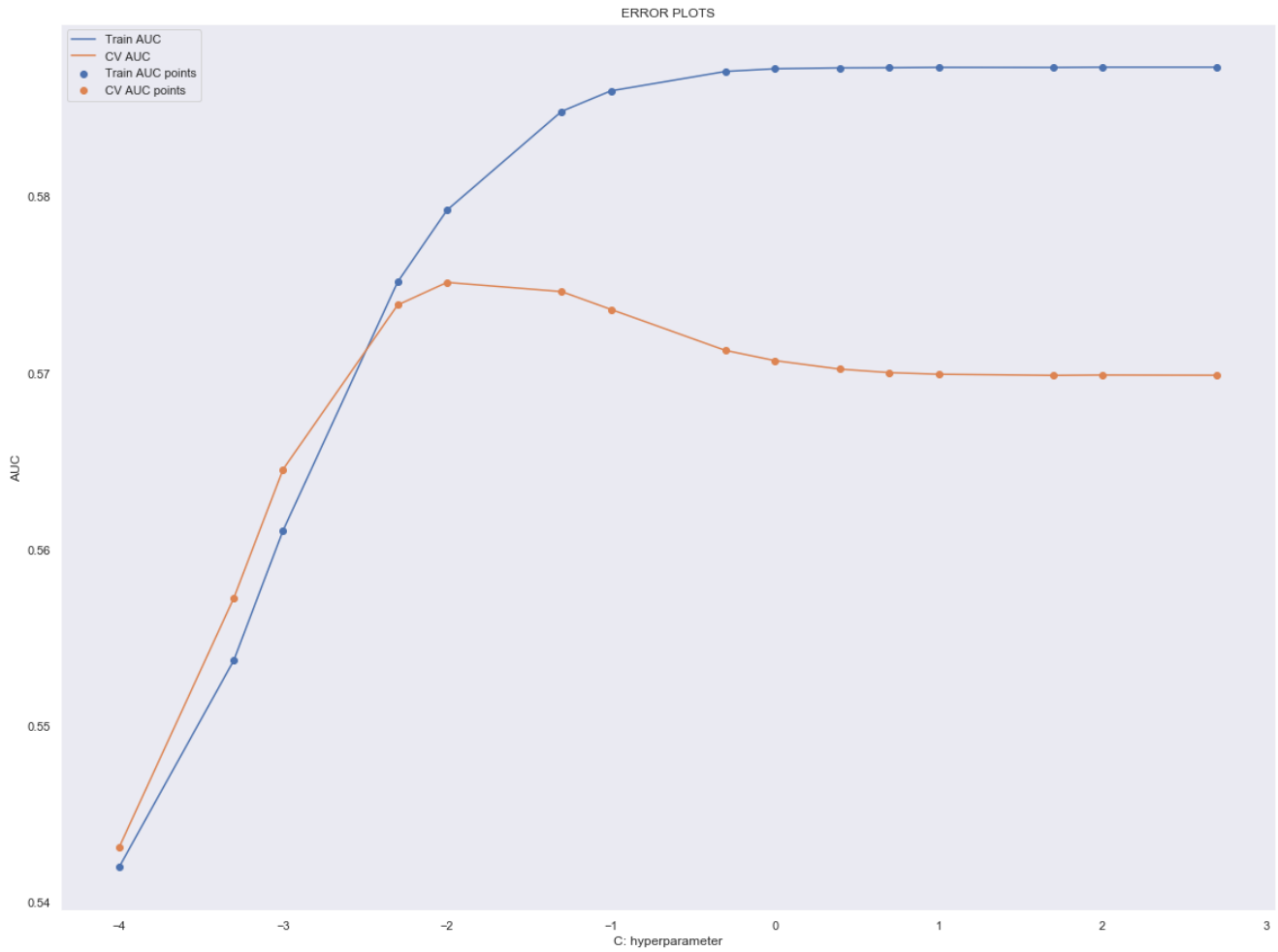
```
100%|██████████| 15/15 [00:00<00:00, 37560.93it/s]
```

```
[-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 [178]:

```
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 [179]:

```
best_k=0.01
```

Train The Model

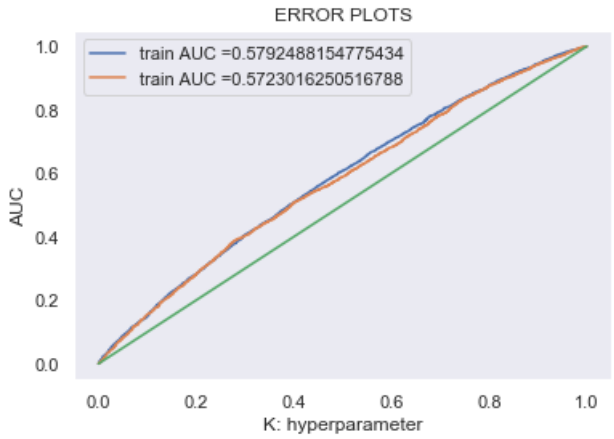
In [180]:

```
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 [181]:

```
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 [182]:

```
# 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 [183]:

```
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) 0.24999997825875406 for threshold 0.845
```

Out[183]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f84748eb5f8>
```

In [184]:

```
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) 0.25 for threshold 0.854
```

Out[184]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f84748d4978>
```

3. Conclusions

In [185]:

```
# 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:

It can be observed that "Essays" and "Project Titles" play a major role in predicting the outcome of the project. So, they cannot be neglected as the top four models containing them proved to have a better AUC score.

In [ ]: