DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

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

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

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [1]:

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

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

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

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

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

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

from tqdm import tqdm
import os


from chart_studio.plotly import plot, iplot
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]:

```
#Due to computational constraints,I'm only considering 20k rows
project_data = pd.read_csv('train_data.csv',nrows=20000)
resource_data = pd.read_csv('resources.csv')
```

In [3]:

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

```
Number of data points in train data (20000, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [4]:

```
project_data.columns
```

Out[4]:

```
Index(['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'],
      dtype='object')
```

In [5]:

```
# 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[5]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_s
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2	
7176	79341	p091436	bb2599c4a114d211b3381abe9f899bf8	Mrs.	OH	2016-04-27 07:24:47	Grades PreK-2	Math

In [6]:

```
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[6]:

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

In [7]:

```
np.unique(project_data["project_grade_category"].values)
```

Out[7]:

```
array(['Grades 3-5', 'Grades 6-8', 'Grades 9-12', 'Grades PreK-2'], dtype=object)
```

In [8]:

```
# We need to get rid of The spaces between the text and the hyphens because they're special charac
ters.
#Rmoving multiple characters from a string in Python
#https://stackoverflow.com/questions/3411771/multiple-character-replace-with-python

project_grade_category = []

for i in range(len(project_data)):
    a = project_data["project_grade_category"][i].replace(" ", "_").replace("-", "_")
    project_grade_category.append(a)
```

In [9]:

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

```
project_data["project_grade_category"] = project_grade_category
print("After removing the special characters ,Column values:
",np.unique(project_data["project_grade_category"].values))
```

```
After removing the special characters ,Column values:   ['Grades_3_5' 'Grades_6_8' 'Grades_9_12'
'Grades_PreK_2']
```

1.2 preprocessing of `project_subject_categories`

In [10]:

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

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

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [11]:

```
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]))
```

1.3 Text preprocessing

In [12]:

```
# 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 [13]:

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

Out[13]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_title	project_essay_1	pro
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Flexible Seating for Flexible Learning	I recently read an article about giving studen...	I t in
7176	79341	p091436	bb2599c4a114d211b3381abe9f899bf8	Mrs.	OH	2016-04-27 07:24:47	Robots are Taking over 2nd Grade	Computer coding and robotics, my second grader...	W Nor

In [14]:

```
# 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 [15]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
```

```
\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "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 [16]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentence in tqdm(project_data['essay'].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.lower() not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████| 20000/20000 [00:08<00:00, 2387.47it/s]
```

In [17]:

```
# after preprocesing
preprocessed_essays[10000]
```

Out[17]:

```
'school located outside tampa florida high poverty rate students eclectic artistic preteens full a
mbition emotions serve one hundred fifty sixth graders science arts inclusion lots kids english se
cond language incorporating arts science allows students make meaningful connections learning fun
learning difficult content engage artistic expressions keen eye real science content envision stud
ents using glitter glow dark acrylic paints special project universe students challenged create sp
ecial painting showing many objects universe painting done encourage reading research science voca
bulary really fun exciting way learn also use creative thinking students today must learn creative
innovators future many students stuck learning rote memory creativity leads discovery innovation w
orkforce demands students ever provided opportunities practice creative students challenged create
painting looks different light compared darkness glow paint comes play mix blend texturize paint m
eet challenge not wait find donate project helping 150 middle school students vocabulary research
reading science art creativity would appreciate support making project come life nannan'
```

In [18]:

```
#creating a new column with the preprocessed essays and replacing it with the original columns
project_data['preprocessed_essays'] = preprocessed_essays
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
```

```
project_data.drop(['project_essay_4'], axis=1, inplace=True)
```

In [19]:

```
#NaN values in techer prefix will create a problem while encoding,so we replace NaN values with th
e mode of that particular column

mode_of_teacher_prefix = project_data['teacher_prefix'].value_counts().index[0]


project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna(mode_of_teacher_prefix)
project_data['teacher_prefix']
```

Out[19]:

```
473      Mrs.
7176     Mrs.
5145     Mrs.
2521      Ms.
5364      Mr.
10985    Mrs.
15560    Mrs.
16710     Ms.
3673      Ms.
2768      Ms.
8336     Mrs.
4202      Ms.
3715      Ms.
15241     Ms.
430       Mr.
1292     Mrs.
3235     Mrs.
10294     Ms.
7147      Ms.
4996      Ms.
9367     Mrs.
17163    Mrs.
8404     Mrs.
18374     Ms.
8049      Ms.
14625     Mr.
6516     Mrs.
8784      Ms.
4178     Mrs.
17094    Mrs.
         ...
18583     Ms.
10201     Ms.
4130     Mrs.
15498    Mrs.
15500     Ms.
16088    Mrs.
9473     Mrs.
8913     Mrs.
10256    Mrs.
14226    Mrs.
7807     Mrs.
1205     Mrs.
6802      Ms.
6108     Mrs.
770       Mr.
15501     Ms.
3259      Ms.
11152    Mrs.
17044    Mrs.
16862     Ms.
3400      Ms.
13593     Ms.
11563     Mr.
12918     Ms.
6095      Ms.
15825     Ms.
5403     Mrs.
18892    Mrs.
11368    Mrs.
14678     Ms.
```

```
14678      Ms.
Name: teacher_prefix, Length: 20000, dtype: object
```

1.4 Preprocessing of `project_title`

In [20]:

```
# 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%|██████████| 20000/20000 [00:00<00:00, 52826.62it/s]
```

In [21]:

```
#creating a new column with the preprocessed titles,useful for analysis
project_data['preprocessed_titles'] = preprocessed_titles
```

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

In [22]:

```
from sklearn.model_selection import train_test_split
#https://scikit-
learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html#sklearn.model_sele
n.train_test_split
project_data_train, project_data_test, y_train, y_test = train_test_split(project_data, project_dat
a['project_is_approved'], test_size=0.33, stratify = project_data['project_is_approved'])
project_data_train, project_data_cv, y_train, y_cv = train_test_split(project_data_train, y_train,
test_size=0.33, stratify=y_train)
```

In [23]:

```
print("Split ratio")
print('-'*50)
print('Train dataset:',len(project_data_train)/len(project_data)*100,'%\n','size:',len(project_data
_train))
print('Cross validation dataset:',len(project_data_cv)/len(project_data)*100,'%\n','size:',len(proj
ect_data_cv))
print('Test dataset:',len(project_data_test)/len(project_data)*100,'%\n','size:',len(project_data_t
est))
```

```
Split ratio
--------------------------------------------------
Train dataset: 44.89 %
 size: 8978
Cross validation dataset: 22.11 %
 size: 4422
Test dataset: 33.0 %
 size: 6600
```

In [24]:

```
#Features
project_data_train.drop(['project_is_approved'], axis=1, inplace=True)
project_data_cv.drop(['project_is_approved'], axis=1, inplace=True)
project_data_test.drop(['project_is_approved'], axis=1, inplace=True)
```

1.5 Preparing data for models

In [25]:

```
project_data.columns
```

Out[25]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_title', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'project_grade_category', 'clean_categories', 'clean_subcategories',
       'essay', 'preprocessed_essays', 'preprocessed_titles'],
      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 (optional)

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

Encoding numerical, categorical features

1.5.1 Vectorizing Categorical data

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

In [26]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True
)


train_categories_one_hot = vectorizer.fit_transform(project_data_train['clean_categories'].values)
cv_categories_one_hot = vectorizer.fit_transform(project_data_cv['clean_categories'].values)
test_categories_one_hot = vectorizer.fit_transform(project_data_test['clean_categories'].values)


print(vectorizer.get_feature_names())
print("Shape of training data matrix after one hot encoding ",train_categories_one_hot.shape)
print("Shape of cross validation data matrix after one hot encoding ",cv_categories_one_hot.shape)
print("Shape of test data matrix after one hot encoding ",test_categories_one_hot.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of training data matrix after one hot encoding  (8978, 9)
Shape of cross validation data matrix after one hot encoding  (4422, 9)
Shape of test data matrix after one hot encoding  (6600, 9)
```

In [27]:

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

```
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=
True)
train_sub_categories_one_hot = vectorizer.fit_transform(project_data_train['clean_subcategories'].
values)
cv_sub_categories_one_hot = vectorizer.fit_transform(project_data_cv['clean_subcategories'].values
)
test_sub_categories_one_hot = vectorizer.fit_transform(project_data_test['clean_subcategories'].va
lues)


train_subcategories_one_hot = vectorizer.fit_transform(project_data_train['clean_subcategories'].v
alues)
cv_subcategories_one_hot = vectorizer.fit_transform(project_data_cv['clean_subcategories'].values)
test_subcategories_one_hot =
vectorizer.fit_transform(project_data_test['clean_subcategories'].values)


print(vectorizer.get_feature_names())


print("Shape of train data matrix after one hot encoding ",train_subcategories_one_hot.shape)
print("Shape of cross validation data matrix after one hot encoding ",cv_subcategories_one_hot.sha
pe)
print("Shape of test data matrix after one hot encoding ",test_subcategories_one_hot.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'College_CareerPrep', 'Oth
er', 'Music', 'History_Geography', 'Health_LifeScience', 'ESL', 'EarlyDevelopment', 'Gym_Fitness',
'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences', 'SpecialNeeds',
'Literature_Writing', 'Mathematics', 'Literacy']
Shape of train data matrix after one hot encoding  (8978, 30)
Shape of cross validation data matrix after one hot encoding  (4422, 30)
Shape of test data matrix after one hot encoding  (6600, 30)
```

In [28]:

```
# you can do the similar thing with state, teacher_prefix and project_grade_category also
my_counter = Counter()
for state in project_data['school_state'].values:
    my_counter.update(state.split())
```

In [29]:

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

In [30]:

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

vectorizer = CountVectorizer(vocabulary=list(sorted_school_state_cat_dict.keys()), lowercase=False
, binary=True)
vectorizer.fit(project_data['school_state'].values)



print(vectorizer.get_feature_names())

train_school_state_category_one_hot =
vectorizer.transform(project_data_train['school_state'].values)
cv_school_state_category_one_hot = vectorizer.transform(project_data_cv['school_state'].values)
test_school_state_category_one_hot = vectorizer.transform(project_data_test['school_state'].values
)


print("Shape of train data matrix after one hot encoding ",train_school_state_category_one_hot.sha
pe)
print("Shape of cross validation data matrix after one hot encoding
",cv_school_state_category_one_hot.shape)
print("Shape of test data matrix after one hot encoding ",test_school_state_category_one_hot.shape
)
```

```
['VT', 'WY', 'ND', 'MT', 'NH', 'RI', 'DE', 'NE', 'SD', 'AK', 'NM', 'HI', 'WV', 'ME', 'DC', 'IA', 'I
D', 'KS', 'AR', 'MN', 'MS', 'CO', 'KY', 'OR', 'MD', 'NV', 'AL', 'UT', 'TN', 'WI', 'CT', 'VA', 'NJ',
'AZ', 'MA', 'OK', 'WA', 'LA', 'MO', 'IN', 'OH', 'PA', 'MI', 'GA', 'SC', 'IL', 'NC', 'FL', 'TX', 'NY
', 'CA']
Shape of train data matrix after one hot encoding  (8978, 51)
Shape of cross validation data matrix after one hot encoding  (4422, 51)
Shape of test data matrix after one hot encoding  (6600, 51)
```

In [31]:

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

In [32]:

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

In [33]:

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

vectorizer = CountVectorizer(vocabulary=list(sorted_project_grade_cat_dict.keys()), lowercase=Fals
e, binary=True)
vectorizer.fit(project_data['project_grade_category'].values)



print(vectorizer.get_feature_names())

train_project_grade_category_one_hot =
vectorizer.transform(project_data_train['project_grade_category'].values)
cv_project_grade_category_one_hot = vectorizer.transform(project_data_cv['project_grade_category']
.values)
test_project_grade_category_one_hot =
vectorizer.transform(project_data_test['project_grade_category'].values)


print("Shape of train data matrix after one hot encoding ",train_project_grade_category_one_hot.sh
ape)
print("Shape of cross validation data matrix after one hot encoding
",cv_project_grade_category_one_hot.shape)
print("Shape of test data matrix after one hot encoding ",test_project_grade_category_one_hot.shap
e)
```

```
['Grades_9_12', 'Grades_6_8', 'Grades_3_5', 'Grades_PreK_2']
Shape of train data matrix after one hot encoding  (8978, 4)
Shape of cross validation data matrix after one hot encoding  (4422, 4)
Shape of test data matrix after one hot encoding  (6600, 4)
```

In [34]:

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

In [35]:

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

In [36]:

```
#https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-scikit-learn-valueerror-np-nan-is
-an-invalid-document
#ValueError: np.nan is an invalid document, expected byte or unicode string.
vectorizer = CountVectorizer(vocabulary=list(sorted_teacher_prefix_cat_dict.keys()), lowercase=Fal
se, binary=True)
```

```
se, binary=True)
vectorizer.fit(project_data['teacher_prefix'].values.astype("U"))


print(vectorizer.get_feature_names())


train_teacher_prefix_categories_one_hot = vectorizer.transform(project_data_train['teacher_prefix'
].values.astype("U"))
cv_teacher_prefix_categories_one_hot =
vectorizer.transform(project_data_cv['teacher_prefix'].values.astype("U"))
test_teacher_prefix_categories_one_hot = vectorizer.transform(project_data_test['teacher_prefix'].
values.astype("U"))

print("Shape of train data matrix after one hot encoding ",train_teacher_prefix_categories_one_hot
.shape)
print("Shape of cross validation data matrix after one hot encoding
",cv_teacher_prefix_categories_one_hot.shape)
print("Shape of test data matrix after one hot encoding ",test_teacher_prefix_categories_one_hot.s
hape)
```

```
['Teacher', 'Mr.', 'Ms.', 'Mrs.']
Shape of train data matrix after one hot encoding  (8978, 4)
Shape of cross validation data matrix after one hot encoding  (4422, 4)
Shape of test data matrix after one hot encoding  (6600, 4)
```

2.3 Make Data Model Ready: encoding essay, and project_title

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [37]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(project_data_train['preprocessed_essays'].values)  #Fitting has to be on Train data


train_essay_bow = vectorizer.transform(project_data_train['essay'].values)
cv_essay_bow = vectorizer.transform(project_data_cv['essay'].values)
test_essay_bow = vectorizer.transform(project_data_test['essay'].values)


print("Shape of train data matrix after one hot encoding ",train_essay_bow.shape)
print("Shape of cross validation data matrix after one hot encoding ",cv_essay_bow.shape)
print("Shape of test data matrix after one hot encoding ",test_essay_bow.shape)
```

```
Shape of train data matrix after one hot encoding  (8978, 5828)
Shape of cross validation data matrix after one hot encoding  (4422, 5828)
Shape of test data matrix after one hot encoding  (6600, 5828)
```

In [38]:

```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit_transform(project_data_train['preprocessed_titles'].values)    #Fitting has to be or
Train data


train_title_bow = vectorizer.transform(project_data_train['preprocessed_titles'].values)
cv_title_bow = vectorizer.transform(project_data_cv['preprocessed_titles'].values)
test_title_bow = vectorizer.transform(project_data_test['preprocessed_titles'].values)



print("Shape of train data matrix after one hot encoding ",train_title_bow.shape)
print("Shape of cross validation data matrix after one hot encoding ",cv_title_bow.shape)
print("Shape of test data matrix after one hot encoding ",test_title_bow.shape)
```

```
Shape of train data matrix after one hot encoding  (8978, 624)
Shape of cross validation data matrix after one hot encoding  (4422, 624)
Shape of test data matrix after one hot encoding  (6600, 624)
```

1.5.2.2 TFIDF vectorizer

In [39]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(project_data_train['preprocessed_essays'])      #Fitting has to be on Train data

train_essay_tfidf = vectorizer.transform(project_data_train['preprocessed_essays'].values)
cv_essay_tfidf = vectorizer.transform(project_data_cv['preprocessed_essays'].values)
test_essay_tfidf = vectorizer.transform(project_data_test['preprocessed_essays'].values)

print("Shape of train data matrix after one hot encoding ",train_essay_tfidf.shape)
print("Shape of cross validation data matrix after one hot encoding ",cv_essay_tfidf.shape)
print("Shape of test data matrix after one hot encoding ",test_essay_tfidf.shape)
```

```
Shape of train data matrix after one hot encoding  (8978, 5828)
Shape of cross validation data matrix after one hot encoding  (4422, 5828)
Shape of test data matrix after one hot encoding  (6600, 5828)
```

In [40]:

```
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(project_data_train['preprocessed_titles'])      #Fitting has to be on Train data

train_title_tfidf = vectorizer.transform(project_data_train['preprocessed_titles'].values)
cv_title_tfidf = vectorizer.transform(project_data_cv['preprocessed_titles'].values)
test_title_tfidf = vectorizer.transform(project_data_test['preprocessed_titles'].values)

print("Shape of train data matrix after one hot encoding ",train_title_tfidf.shape)
print("Shape of cross validation data matrix after one hot encoding ",cv_title_tfidf.shape)
print("Shape of test data matrix after one hot encoding ",test_title_tfidf.shape)
```

```
Shape of train data matrix after one hot encoding  (8978, 624)
Shape of cross validation data matrix after one hot encoding  (4422, 624)
Shape of test data matrix after one hot encoding  (6600, 624)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [41]:

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

In [42]:

```
# average Word2Vec
# compute average word2vec for each review.
train_avg_w2v_essays = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(project_data_train['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_avg_w2v_essays.append(vector)
```

```
print(len(train_avg_w2v_essays))
print(len(train_avg_w2v_essays[0]))
```

```
100%|██████████| 8978/8978 [00:01<00:00, 6109.51it/s]
```

```
8978
300
```

In [43]:

```
# average Word2Vec
# compute average word2vec for each review.
cv_avg_w2v_essays = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(project_data_cv['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_avg_w2v_essays.append(vector)

print(len(cv_avg_w2v_essays))
print(len(cv_avg_w2v_essays[0]))
```

```
100%|██████████| 4422/4422 [00:00<00:00, 6000.89it/s]
```

```
4422
300
```

In [44]:

```
# average Word2Vec
# compute average word2vec for each review.
test_avg_w2v_essays = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(project_data_test['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_avg_w2v_essays.append(vector)

print(len(test_avg_w2v_essays))
print(len(test_avg_w2v_essays[0]))
```

```
100%|██████████| 6600/6600 [00:01<00:00, 6127.03it/s]
```

```
6600
300
```

In [45]:

```
# average Word2Vec
# compute average word2vec for each review.
train_avg_w2v_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(project_data_train['preprocessed_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
```

```
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_avg_w2v_titles.append(vector)

print(len(train_avg_w2v_titles))
print(len(train_avg_w2v_titles[0]))
```

```
100%|██████████| 8978/8978 [00:00<00:00, 91195.98it/s]
```

```
8978
300
```

In [46]:

```
# average Word2Vec
# compute average word2vec for each review.
cv_avg_w2v_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(project_data_cv['preprocessed_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_avg_w2v_titles.append(vector)

print(len(cv_avg_w2v_titles))
print(len(cv_avg_w2v_titles[0]))
```

```
100%|██████████| 4422/4422 [00:00<00:00, 87074.89it/s]
```

```
4422
300
```

In [47]:

```
# average Word2Vec
# compute average word2vec for each review.
test_avg_w2v_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(project_data_test['preprocessed_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_avg_w2v_titles.append(vector)

print(len(test_avg_w2v_titles))
print(len(test_avg_w2v_titles[0]))
```

```
100%|██████████| 6600/6600 [00:00<00:00, 89703.20it/s]
```

```
6600
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [48]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(project_data_train['preprocessed_essays'].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 [49]:

```
# average Word2Vec
# compute average word2vec for each review.
train_tfidf_w2v_essays = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(project_data_train['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_tfidf_w2v_essays.append(vector)

print(len(train_tfidf_w2v_essays))
print(len(train_tfidf_w2v_essays[0]))
```

```
100%|██████████| 8978/8978 [00:10<00:00, 848.18it/s]
```

```
8978
300
```

In [50]:

```
# average Word2Vec
# compute average word2vec for each review.
cv_tfidf_w2v_essays = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(project_data_cv['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    cv_tfidf_w2v_essays.append(vector)

print(len(cv_tfidf_w2v_essays))
print(len(cv_tfidf_w2v_essays[0]))
```

```
100%|██████████| 4422/4422 [00:05<00:00, 870.96it/s]
```

```
4422
300
```

In [51]:

```
# average Word2Vec
# compute average word2vec for each review.
test_tfidf_w2v_essays = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(project_data_test['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_tfidf_w2v_essays.append(vector)

print(len(test_tfidf_w2v_essays))
print(len(test_tfidf_w2v_essays[0]))
```

```
100%|██████████| 6600/6600 [00:07<00:00, 868.51it/s]
```

```
6600
300
```

In [52]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(project_data_train['preprocessed_titles'])
# 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 [53]:

```
# average Word2Vec
# compute average word2vec for each review.
train_tfidf_w2v_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(project_data_train['preprocessed_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((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
    train_tfidf_w2v_titles.append(vector)

print(len(train_tfidf_w2v_titles))
print(len(train_tfidf_w2v_titles[0]))
```

```
100%|██████████| 8978/8978 [00:00<00:00, 41597.90it/s]
```

```
8978
300
```

In [54]:

```
# average Word2Vec
# compute average word2vec for each review.
cv_tfidf_w2v_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(project_data_cv['preprocessed_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((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
    cv_tfidf_w2v_titles.append(vector)

print(len(cv_tfidf_w2v_titles))
print(len(cv_tfidf_w2v_titles[0]))
```

```
100%|██████████| 4422/4422 [00:00<00:00, 44347.87it/s]
```

```
4422
300
```

In [55]:

```
# average Word2Vec
# compute average word2vec for each review.
test_tfidf_w2v_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(project_data_test['preprocessed_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((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
    test_tfidf_w2v_titles.append(vector)

print(len(test_tfidf_w2v_titles))
print(len(test_tfidf_w2v_titles[0]))
```

```
100%|██████████| 6600/6600 [00:00<00:00, 44011.44it/s]
```

```
6600
300
```

1.5.3 Vectorizing Numerical features

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

In [56]:

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

In [57]:

```
project_data_train = pd.merge(project_data_train, price_data, on='id', how='left')
project_data_cv = pd.merge(project_data_cv, price_data, on='id', how='left')
project_data_test = pd.merge(project_data_test, price_data, on='id', how='left')
```

In [58]:

```
project_data_train.head(5)
```

Out[58]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_title	project_resource_summ
0	106260	p141943	29b2278fc1820d9ae41a24795bf2a756	Mrs.	NJ	2017-01-09 04:23:09	Cozy Up and Learn!	My students need 4 be bag chairs and 2 fol
1	136566	p145501	74dc1304c587117795889ab03197f5bc	Mrs.	FL	2016-08-15 22:15:23	Why Yes! We Wiggle!	My students need stab yoga balls and wob
2	2213	p202348	4450918c4d82fd93d10835864c29720e	Mrs.	LA	2016-08-17 12:08:23	Wobble Stools Needed for Wood's Wiggly Kinderg...	My students need flex and comfortable sea
3	160540	p061515	c9b4fc4b82618bda408d2ba342591ff3	Ms.	CA	2016-10-21 10:44:11	Yoga Balls	My students need stabilit a way to focus t
4	84585	p227858	3ecf6e4a48272107d346991dd7891282	Ms.	CT	2016-08-09 20:01:51	Namaste Focused	My students need a class of kid-friendly

In [59]:

```
from sklearn.preprocessing import Normalizer
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

normalizer = Normalizer()
normalizer.fit(project_data_train['price'].values.reshape(-1,1))


price_normalized_train = normalizer.transform(project_data_train['price'].values.reshape(-1, 1))
price_normalized_cv = normalizer.transform(project_data_cv['price'].values.reshape(-1, 1))
price_normalized_test = normalizer.transform(project_data_test['price'].values.reshape(-1, 1))


print('After normalization')
print(price_normalized_train.shape)
print(price_normalized_cv.shape)
print(price_normalized_test.shape)
```

```
After normalization
(8978, 1)
(4422, 1)
(6600, 1)
```

In [60]:

```
normalizer = Normalizer()
normalizer.fit(project_data_train['teacher_number_of_previously_posted_projects'].values.reshape(-
1,1))
```

```
# Now standardize the data with above maen and variance.
previously_posted_projects_normalized_train =
normalizer.transform(project_data_train['teacher_number_of_previously_posted_projects'].values.res
hape(-1, 1))
previously_posted_projects_normalized_cv =
normalizer.transform(project_data_cv['teacher_number_of_previously_posted_projects'].values.reshape
(-1, 1))
previously_posted_projects_normalized_test =
normalizer.transform(project_data_test['teacher_number_of_previously_posted_projects'].values.resh
ape(-1, 1))


print('After normalization')
print(previously_posted_projects_normalized_train.shape)
print(previously_posted_projects_normalized_cv.shape)
print(previously_posted_projects_normalized_test.shape)
```

```
After normalization
(8978, 1)
(4422, 1)
(6600, 1)
```

Assignment 3: Apply KNN

1. **[Task-1] Apply KNN(brute force version) on these feature sets**
   - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)
   - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF)
   - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_essay (AVG W2V)
   - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)
2. **Hyper paramter tuning to find best K**
   - Find the best hyper parameter which results in 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 write your own for loops to do this task
3. **Representation of results**
   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, as shown in the figure
   - Once you find the best hyper parameter, you need to train your model-M using the best hyper-param. Now, find the AUC on test data and plot the ROC curve on both train and test using model-M.
   - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points
4. **[Task-2]**
   - Select top 2000 features from feature Set 2 using `SelectKBest` and then apply KNN on top of these features
     - ```
       from sklearn.datasets import load_digits
       from sklearn.feature_selection import SelectKBest, chi2
       X, y = load_digits(return_X_y=True)
       X.shape
       X_new = SelectKBest(chi2, k=20).fit_transform(X, y)
       X_new.shape
       ========
       output:
       (1797, 64)
       (1797, 20)
       ```
   - Repeat the steps 2 and 3 on the data matrix after feature selection
5. **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. K Nearest Neighbor

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

Apply KNN 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

2.4.1 Applying KNN brute force on BOW, SET 1

In [61]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train = hstack((train_categories_one_hot, train_sub_categories_one_hot, train_essay_bow,
train_title_bow, train_school_state_category_one_hot,train_teacher_prefix_categories_one_hot, prev
iously_posted_projects_normalized_train, train_project_grade_category_one_hot,
price_normalized_train)).tocsr()
X_cv = hstack((cv_categories_one_hot, cv_sub_categories_one_hot, cv_essay_bow, cv_title_bow,
cv_school_state_category_one_hot, cv_teacher_prefix_categories_one_hot,
previously_posted_projects_normalized_cv, cv_project_grade_category_one_hot, price_normalized_cv))
.tocsr()
X_test = hstack((test_categories_one_hot, test_sub_categories_one_hot, test_essay_bow,
test_title_bow, test_school_state_category_one_hot, test_teacher_prefix_categories_one_hot,
previously_posted_projects_normalized_test, test_project_grade_category_one_hot,
price_normalized_test)).tocsr()
print(X_train.shape)
print(X_cv.shape)
print(X_test.shape)
```

```
(8978, 6552)
(4422, 6552)
(6600, 6552)
```

In [62]:

```
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 [63]:

```
import matplotlib.pyplot as plt
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True class labels

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 = []
a = []
b = []
#trying different values of K
K = [1, 5, 10, 15, 21, 31, 41, 51, 65, 71]

for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train, y_train)
    #batch_predict returns the probability estimate of a point belonging to a class label
    y_train_pred = batch_predict(neigh, X_train)
    y_cv_pred = batch_predict(neigh, X_cv)

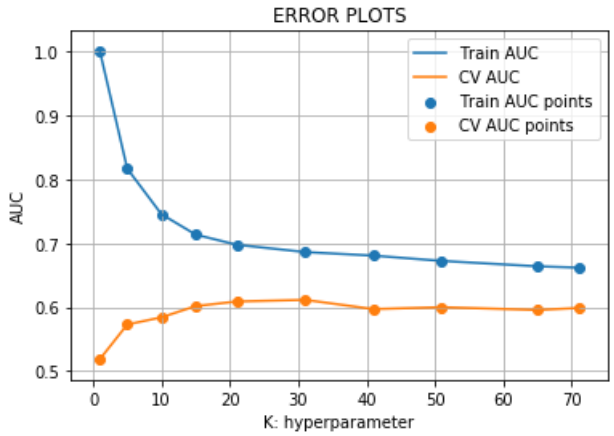

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
    a.append(y_train_pred)
    b.append(y_cv_pred)

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

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

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

```
100%|██████████| 10/10 [01:20<00:00,  8.12s/it]
```



In [64]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier()

parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51, 65, 71, 85, 91]}

clf = GridSearchCV(neigh, parameters, cv= 5, scoring='roc_auc')

clf.fit(X_train, y_train)

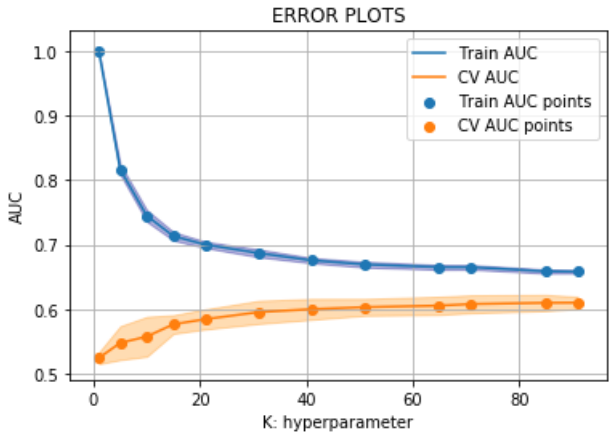
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['n_neighbors'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,
color='darkorange')

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')


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



In [65]:

```
best_k1 = 91
```

In [66]:

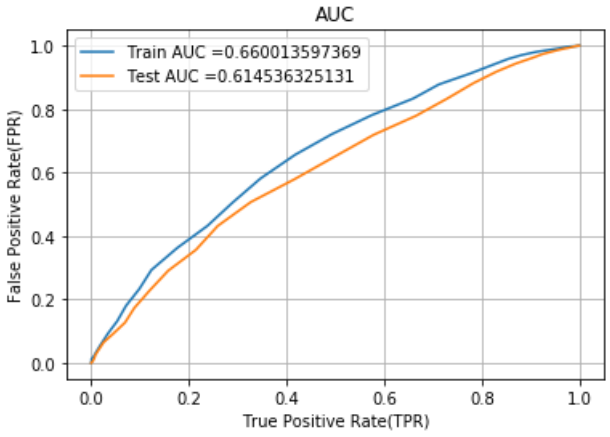
```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
from sklearn.neighbors import KNeighborsClassifier

neigh = KNeighborsClassifier(n_neighbors=best_k1)
neigh.fit(X_train, 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_train)
y_test_pred = batch_predict(neigh, X_test)

train_fpr, train_tpr, train_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_pred)
```

```
plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



In [67]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

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

In [68]:

```
#our objective here is to make auc the maximum
#so we find  the best threshold that will give the least fpr
best_t = find_best_threshold(train_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 0.382022423371 for threshold 0.78
Train confusion matrix
[[ 797  572]
 [2616 4993]]
```

In [69]:

```
#plotting confusion matrix using seaborn's heatmap
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

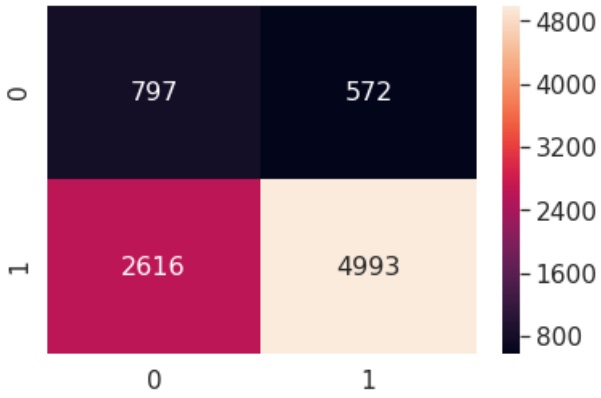
print("Train data confusion matrix")

confusion_matrix_df_train = pd.DataFrame(confusion_matrix(y_train,
predict_with_best_t(y_train_pred, best_t)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(confusion_matrix_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Train data confusion matrix
```

Out[69]:

<matplotlib.axes._subplots.AxesSubplot at 0x7fe5cca73358>



In [70]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
Test confusion matrix
[[ 503  503]
 [1950 3644]]
```

In [71]:

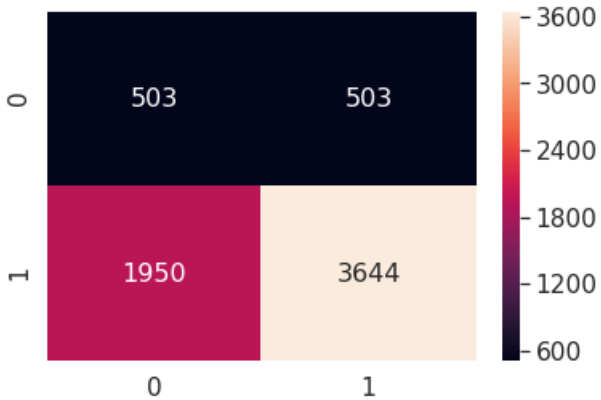
```
print("Test data confusion matrix")

confusion_matrix_df_test = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, b
est_t)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(confusion_matrix_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Test data confusion matrix
```

Out[71]:

<matplotlib.axes._subplots.AxesSubplot at 0x7fe5ccbb1cf8>



2.4.2 Applying KNN brute force on TFIDF, SET 2

In [72]:

```
X_train = hstack((train_categories_one_hot, train_sub_categories_one_hot, train_essay_tfidf,
train_title_tfidf, train_school_state_category_one_hot,train_teacher_prefix_categories_one_hot, pr
eviously_posted_projects_normalized_train, train_project_grade_category_one_hot,
price_normalized_train)).tocsr()
```

```
X_cv = hstack((cv_categories_one_hot, cv_sub_categories_one_hot, cv_essay_tfidf, cv_title_tfidf, cv
_school_state_category_one_hot, cv_teacher_prefix_categories_one_hot,
previously_posted_projects_normalized_cv, cv_project_grade_category_one_hot, price_normalized_cv))
.tocsr()
X_test = hstack((test_categories_one_hot, test_sub_categories_one_hot, test_essay_tfidf,
test_title_tfidf, test_school_state_category_one_hot, test_teacher_prefix_categories_one_hot,
previously_posted_projects_normalized_test, test_project_grade_category_one_hot,
price_normalized_test)).tocsr()
print(X_train.shape)
print(X_cv.shape)
print(X_test.shape)
```

```
(8978, 6552)
(4422, 6552)
(6600, 6552)
```

In [73]:

```
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True class labels

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.

"""
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score
train_auc = []
cv_auc = []


K = [1, 5, 10, 15, 21, 31, 41, 51, 65, 71]

for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train, y_train)

    y_train_pred = batch_predict(neigh, X_train)
    y_cv_pred = batch_predict(neigh, X_cv)

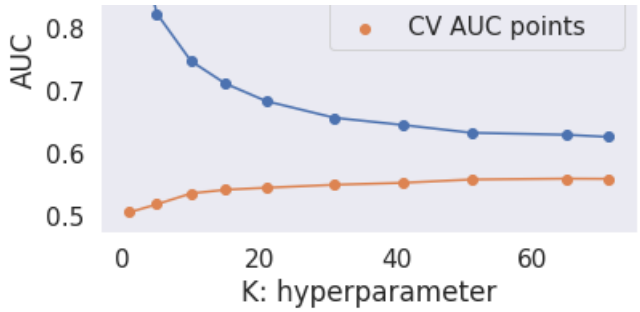

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
    a.append(y_train_pred)
    b.append(y_cv_pred)

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

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

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

```
100%|██████████| 10/10 [01:21<00:00,  8.17s/it]
```



In [74]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier()

parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51, 65, 71, 85, 91]}

clf = GridSearchCV(neigh, parameters, cv= 5, scoring='roc_auc')

clf.fit(X_train, y_train)

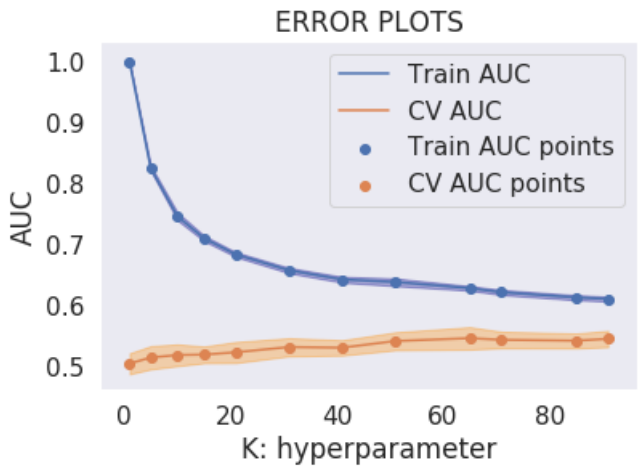
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['n_neighbors'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,
color='darkorange')

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')


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



In [75]:

```
best_k2 = 85
```

In [76]:

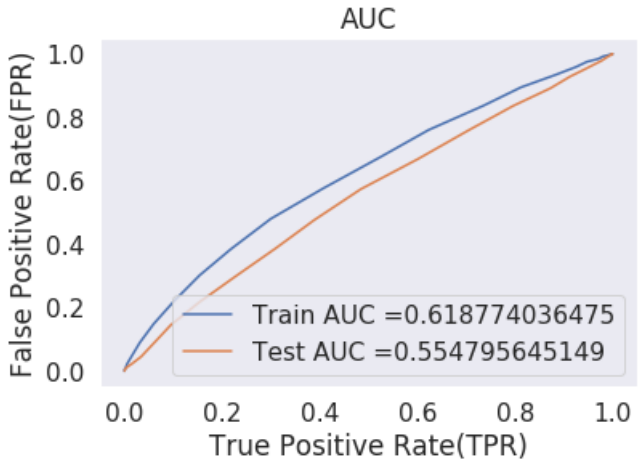
```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k2)
neigh.fit(X_train, 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_train)
y_test_pred = batch_predict(neigh, X_test)

train_fpr, train_tpr, train_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



In [77]:

```
#our objective here is to make auc the maximum
#so we find  the best threshold that will give the least fpr
best_t = find_best_threshold(train_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 0.341474634868 for threshold 0.847
Train confusion matrix
[[ 811  558]
 [3223 4386]]
```

In [78]:

```
#plotting confusion matrix using seaborn's heatmap
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

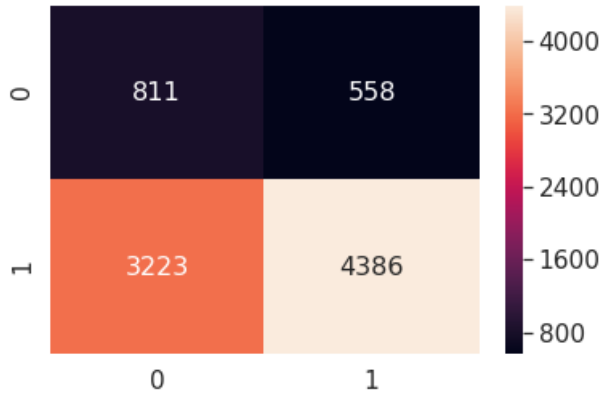
print("Train data confusion matrix")

confusion_matrix_df_train = pd.DataFrame(confusion_matrix(y_train,
predict_with_best_t(y_train_pred, best_t)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(confusion_matrix_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Train data confusion matrix
```

Out[78]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe5ccc61710>
```



In [79]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
Test confusion matrix
[[ 519  487]
 [2386 3208]]
```

In [80]:

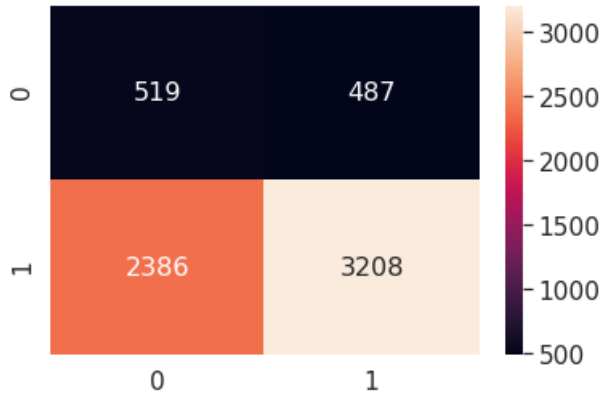
```
print("Test data confusion matrix")

confusion_matrix_df_test = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, b
est_t)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(confusion_matrix_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Test data confusion matrix
```

Out[80]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe5ccb17ba8>
```



2.4.3 Applying KNN brute force on AVG W2V, SET 3

In [81]:

```
X_train = hstack((train_categories_one_hot, train_sub_categories_one_hot, train_avg_w2v_essays,
train_avg_w2v_titles, train_school_state_category_one_hot,train_teacher_prefix_categories_one_hot,
previously_posted_projects_normalized_train, train_project_grade_category_one_hot,
price_normalized_train)).tocsr()
```

```
X_cv = hstack((cv_categories_one_hot, cv_sub_categories_one_hot, cv_avg_w2v_essays,
cv_avg_w2v_titles, cv_school_state_category_one_hot, cv_teacher_prefix_categories_one_hot,
previously_posted_projects_normalized_cv, cv_project_grade_category_one_hot, price_normalized_cv))
.tocsr()
X_test = hstack((test_categories_one_hot, test_sub_categories_one_hot, test_avg_w2v_essays,
test_avg_w2v_titles, test_school_state_category_one_hot, test_teacher_prefix_categories_one_hot, p
reviously_posted_projects_normalized_test, test_project_grade_category_one_hot,
price_normalized_test)).tocsr()
print(X_train.shape)
print(X_cv.shape)
print(X_test.shape)
```

```
(8978, 700)
(4422, 700)
(6600, 700)
```

In [82]:

```
"""
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 = []
a = []
b = []

K = [1, 5, 10, 15, 21, 31, 41, 51, 65, 71]

for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train, y_train)

    y_train_pred = batch_predict(neigh, X_train)
    y_cv_pred = batch_predict(neigh, X_cv)

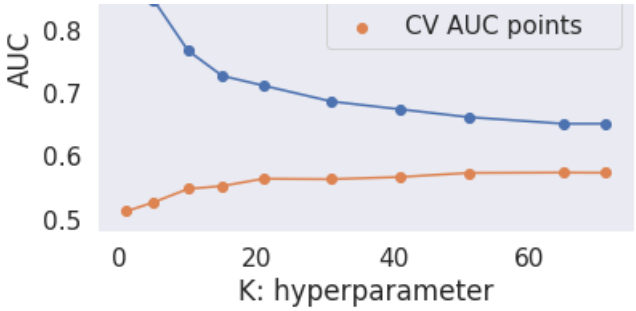

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
    a.append(y_train_pred)
    b.append(y_cv_pred)

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

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

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

```
100%|██████████| 10/10 [23:16<00:00, 142.76s/it]
```



In [83]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier()

parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51, 65, 71, 85, 91]}

clf = GridSearchCV(neigh, parameters, cv= 5, scoring='roc_auc')

clf.fit(X_train, y_train)

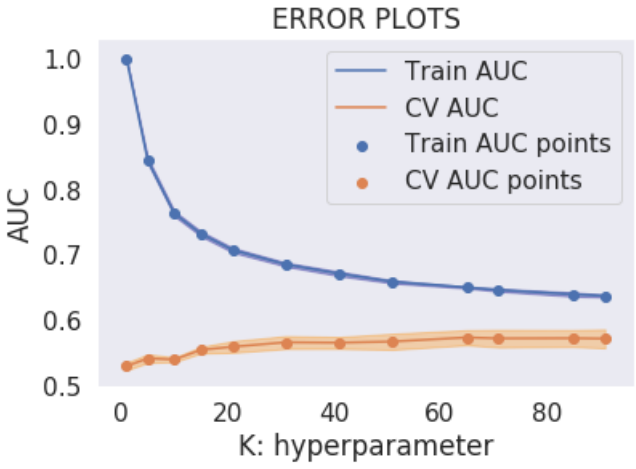
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['n_neighbors'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,
color='darkorange')

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')


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



In [84]:

```
best_k3 = 91
```

In [85]:

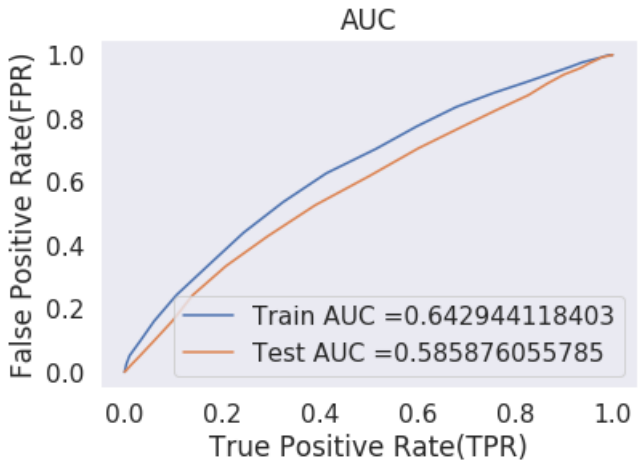
```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve


neigh = KNeighborsClassifier(n_neighbors=best_k3)
neigh.fit(X_train, 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_train)
y_test_pred = batch_predict(neigh, X_test)

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



In [86]:

```
#our objective here is to make auc the maximum
#so we find  the best threshold that will give the least fpr
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 0.367630754438 for threshold 0.857
Train confusion matrix
[[ 803  566]
 [2840 4769]]
```

In [87]:

```
#plotting confusion matrix using seaborn's heatmap
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

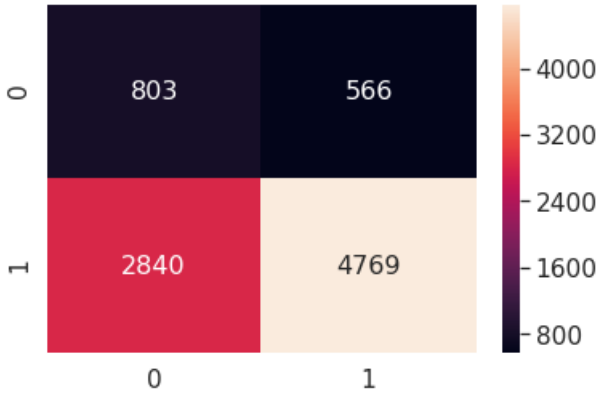
print("Train data confusion matrix")

confusion_matrix_df_train = pd.DataFrame(confusion_matrix(y_train,
predict_with_best_t(y_train_pred, best_t)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(confusion_matrix_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Train data confusion matrix
```

Out[87]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe5ccc61048>
```



In [88]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
Test confusion matrix
[[ 501  505]
 [2135 3459]]
```

In [89]:

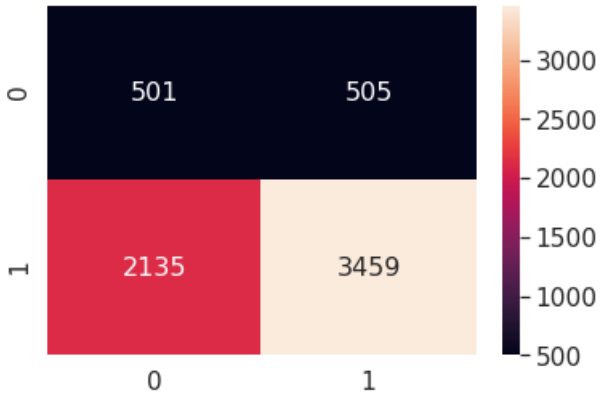
```
print("Test data confusion matrix")

confusion_matrix_df_test = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, b
est_t)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(confusion_matrix_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Test data confusion matrix
```

Out[89]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe5ccbd5e10>
```



2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

In [90]:

```
X_train = hstack((train_categories_one_hot, train_sub_categories_one_hot, train_tfidf_w2v_essays,
train_tfidf_w2v_titles,
train_school_state_category_one_hot,train_teacher_prefix_categories_one_hot,
previously_posted_projects_normalized_train, train_project_grade_category_one_hot,
price_normalized_train)).tocsr()
```

```
X_cv = hstack((cv_categories_one_hot, cv_sub_categories_one_hot, cv_tfidf_w2v_essays,
cv_tfidf_w2v_titles, cv_school_state_category_one_hot, cv_teacher_prefix_categories_one_hot,
previously_posted_projects_normalized_cv, cv_project_grade_category_one_hot, price_normalized_cv))
.tocsr()
X_test = hstack((test_categories_one_hot, test_sub_categories_one_hot, test_tfidf_w2v_essays,
test_tfidf_w2v_titles, test_school_state_category_one_hot, test_teacher_prefix_categories_one_hot,
previously_posted_projects_normalized_test, test_project_grade_category_one_hot,
price_normalized_test)).tocsr()
print(X_train.shape)
print(X_cv.shape)
print(X_test.shape)
```

```
(8978, 700)
(4422, 700)
(6600, 700)
```

In [91]:

```
"""
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 = []
a = []
b = []

K = [1, 5, 10, 15, 21, 31, 41, 51, 65, 71]

for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train, y_train)

    y_train_pred = batch_predict(neigh, X_train)
    y_cv_pred = batch_predict(neigh, X_cv)

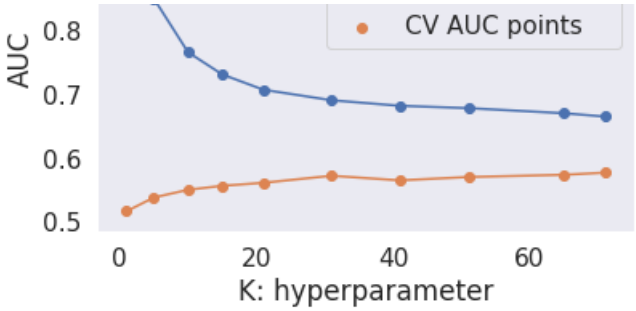

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
    a.append(y_train_pred)
    b.append(y_cv_pred)

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

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

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

```
100%|██████████| 10/10 [22:16<00:00, 133.53s/it]
```



In [92]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

neigh = KNeighborsClassifier()

parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51, 65, 71, 85, 91]}

clf = GridSearchCV(neigh, parameters, cv= 5, scoring='roc_auc')

clf.fit(X_train, y_train)

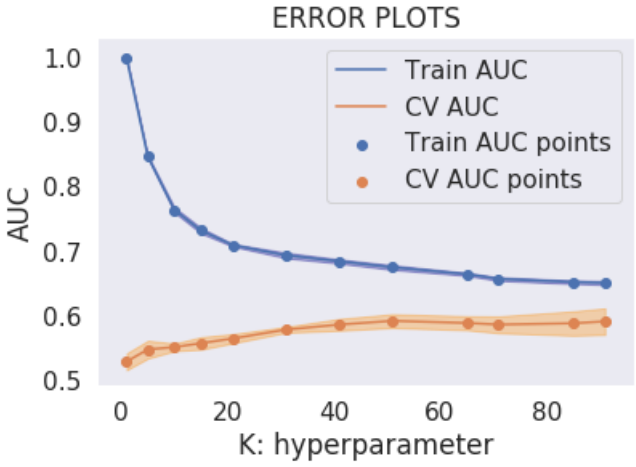
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['n_neighbors'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,
color='darkorange')

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')


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



In [93]:

```
best_k4 = 85
```

In [94]:

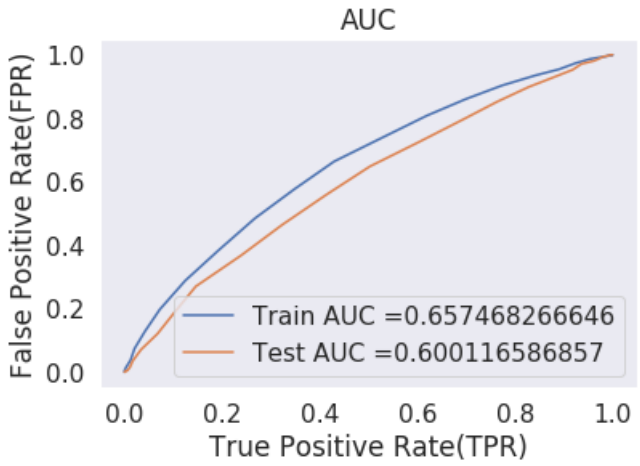
```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve


neigh = KNeighborsClassifier(n_neighbors=best_k4)
neigh.fit(X_train, 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_train)
y_test_pred = batch_predict(neigh, X_test)

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



In [95]:

```
#our objective here is to make auc the maximum
#so we find  the best threshold that will give the least fpr
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 0.378851751909 for threshold 0.847
Train confusion matrix
[[ 781  588]
 [2556 5053]]
```

In [97]:

```
#plotting confusion matrix using seaborn's heatmap
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

print("Train data confusion matrix")

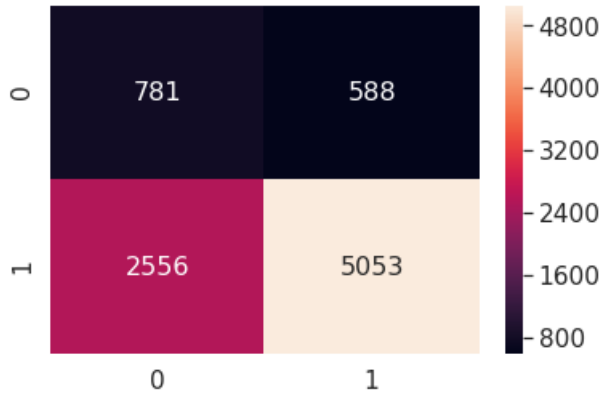
confusion_matrix_df_train = pd.DataFrame(confusion_matrix(y_train,
predict_with_best_t(y_train_pred, best_t)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(confusion_matrix_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Train data confusion matrix
```

Out[97]:

Out[97]:

<matplotlib.axes._subplots.AxesSubplot at 0x7fe5ae96e908>



In [98]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
Test confusion matrix
[[ 500  506]
 [1966 3628]]
```

In [99]:

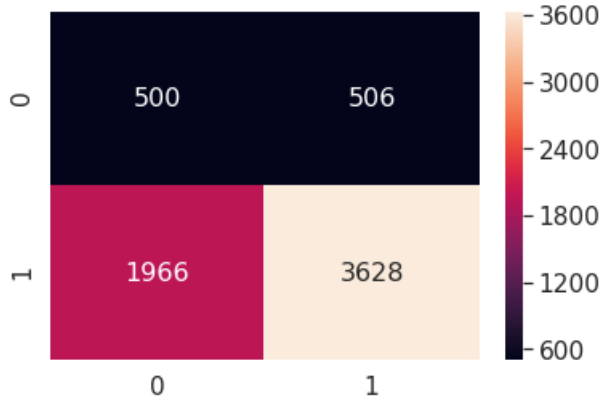
```
print("Test data confusion matrix")

confusion_matrix_df_test = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, b
est_t)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(confusion_matrix_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Test data confusion matrix
```

Out[99]:

<matplotlib.axes._subplots.AxesSubplot at 0x7fe5c77d4358>



2.5 Feature selection with `SelectKBest`

In [102]:

```
X_train = hstack((train_categories_one_hot, train_sub_categories_one_hot,
train_school_state_category_one_hot, train_project_grade_category_one_hot,
train_teacher_prefix_categories_one_hot, price_normalized_train ,
previously_posted_projects_normalized_train,  train_essay_tfidf, train_title_tfidf)).tocsr()
X_test = hstack((test_categories_one_hot, test_sub_categories_one_hot,
test_school_state_category_one_hot, test_project_grade_category_one_hot,
```

```
test_teacher_prefix_categories_one_hot, price_normalized_test,
previously_posted_projects_normalized_test,  test_essay_tfidf, test_title_tfidf)).tocsr()
X_cv = hstack((cv_categories_one_hot, cv_sub_categories_one_hot, cv_school_state_category_one_hot,
cv_project_grade_category_one_hot, cv_teacher_prefix_categories_one_hot,
price_normalized_cv,previously_posted_projects_normalized_cv,  cv_essay_tfidf,
cv_title_tfidf)).tocsr()
```

In [103]:

```
from sklearn.feature_selection import SelectKBest, chi2
#https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html
X_train_new = SelectKBest(chi2, k=2000).fit_transform(X_train, y_train)
X_test_new = SelectKBest(chi2, k=2000).fit_transform(X_test, y_test)
X_cv_new = SelectKBest(chi2, k=2000).fit_transform(X_cv, y_cv)
```

In [104]:

```
print(X_train_new.shape, y_train.shape)
print(X_cv_new.shape, y_cv.shape)
print(X_test_new.shape, y_test.shape)
```

```
(8978, 2000) (8978,)
(4422, 2000) (4422,)
(6600, 2000) (6600,)
```

In [105]:

```
train_auc = []
cv_auc = []

K = [1, 5, 10, 15, 21, 31, 41, 51, 65, 71, 85, 91]

for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train_new, y_train)

    y_train_pred = batch_predict(neigh, X_train_new)
    y_cv_pred = batch_predict(neigh, X_cv_new)

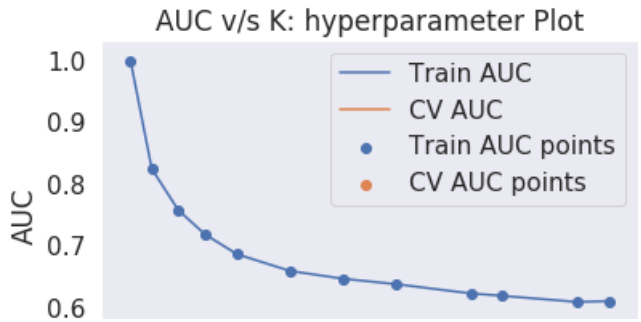
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

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

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC v/s K: hyperparameter Plot")
plt.grid()
plt.show()
```

```
100%|██████████| 12/12 [00:51<00:00,  4.37s/it]
```



In [106]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

neigh = KNeighborsClassifier()

parameters = {'n_neighbors':[1, 5, 10, 15, 21, 33, 41, 51, 65, 71, 85, 91]}

clf = GridSearchCV(neigh, parameters, cv=5, scoring='roc_auc')
clf.fit(X_train_new, y_train)

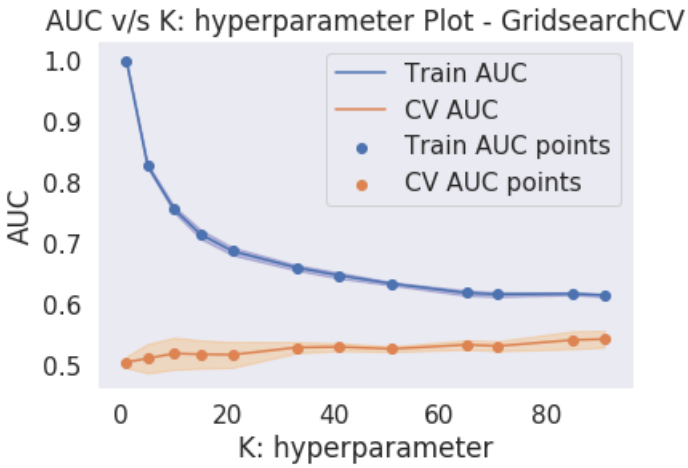
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.2,color='darkblue')

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

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC v/s K: hyperparameter Plot - GridsearchCV")
plt.grid()
plt.show()
```



In [107]:

```
best_k5 = 85
```

In [108]:

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

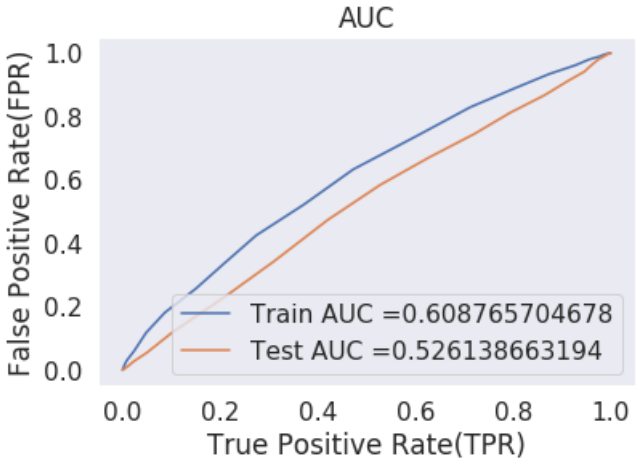
neigh = KNeighborsClassifier(n_neighbors=best_k5)
neigh.fit(X_train_new, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
```

```
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_train_new)
y_test_pred = batch_predict(neigh, X_test_new)

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



In [109]:

```
#our objective here is to make auc the maximum
#so we find  the best threshold that will give the least fpr
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 0.333666227597 for threshold 0.835
Train confusion matrix
[[ 722  647]
 [2795 4814]]
```

In [110]:

```
#plotting confusion matrix using seaborn's heatmap
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

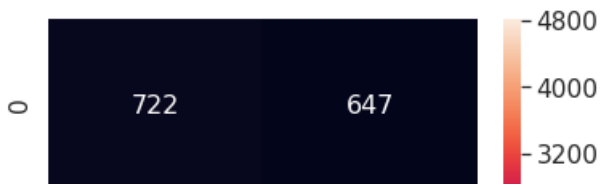
print("Train data confusion matrix")

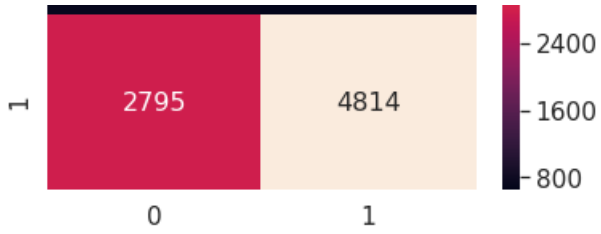
confusion_matrix_df_train = pd.DataFrame(confusion_matrix(y_train,
predict_with_best_t(y_train_pred, best_t)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(confusion_matrix_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Train data confusion matrix
```

Out[110]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe5ae9adb38>
```





In [111]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
Test confusion matrix
[[ 475  531]
 [2329 3265]]
```

In [112]:

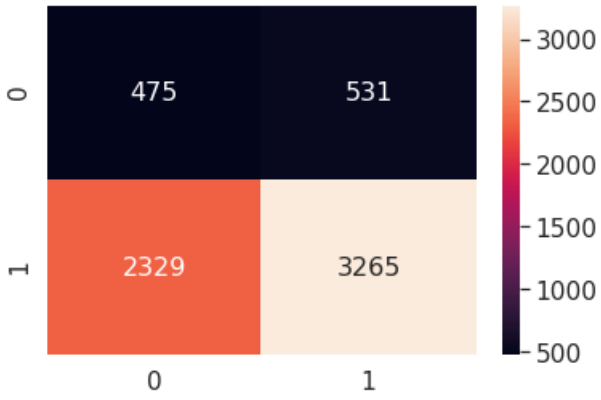
```
print("Test data confusion matrix")

confusion_matrix_df_test = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, b
est_t)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(confusion_matrix_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Test data confusion matrix
```

Out[112]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe5ccc0f908>
```



3. Conclusions

In [115]:

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

# Compare all your models using Prettytable library
# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyper Parameter", "AUC"]


x.add_row(["BOW", "Brute Force KNN", 91, 0.61])
x.add_row(["TFIDF", "Brute Force KNN", 85, 0.55])
x.add_row(["AVG W2V", "Brute Force KNN", 91, 0.58])
x.add_row(["TFIDF W2V", "Brute Force KNN", 85, 0.60])
x.add_row(["TFIDF", "Feature selection with SelectKBest(Top2000)", 85, 0.52])

print(x)
```

```
+------------+---------------------------------------------+-----------------+-------+
| Vectorizer |                    Model                    | Hyper Parameter | AUC   |
+------------+---------------------------------------------+-----------------+-------+
|    BOW     |               Brute Force KNN               |        91       | 0.61 |
|   TFIDF    |               Brute Force KNN               |        85       | 0.55 |
|  AVG W2V   |               Brute Force KNN               |        91       | 0.58 |
| TFIDF W2V  |               Brute Force KNN               |        85       | 0.6  |
|   TFIDF    | Feature selection with SelectKBest(Top2000) |        85       | 0.52 |
+------------+---------------------------------------------+-----------------+-------+
```