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

Feature	Description
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*
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."

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 [2]:

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

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

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

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

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

from tqdm import tqdm
import os

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

1.1 Reading Data

In [3]:

```
train_data= pd.read_csv('C:/Users/User/Downloads/train_data.csv', na_values=' ')
resource_data = pd.read_csv('C:/Users/User/Downloads/resourcesA.csv', na_values=' ')
```

In [4]:

```
df=pd.DataFrame(train_data)
```

In [5]:

```
project_data=df[0:25000]
```

In [6]:

```
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 (25000, 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 [7]:

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


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


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


project_data.head(2)
```

Out[7]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_categ
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2
23374	72317	p087808	598621c141cda5fb184ee7e8ccdd3fcc	Ms.	CA	2016-04-27 02:04:15	Grades PreK-2

In [8]:

```
print("Number of data points in resource data",resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

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

Out[8]:

	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 [9]:

```
print("Number of data points in train data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

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

Out[9]:

Out[9]:

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

1.2 preprocessing of `project_subject_categories`

In [10]:

```
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('&','_')
        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_grade_categ
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2
23374	72317	p087808	598621c141cda5fb184ee7e8ccdd3fcc	Ms.	CA	2016-04-27 02:04:15	Grades PreK-2

1.4.2.3 Using Pretrained Models: TFIDF weighted W2V

In [14]:

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

```
I recently read an article about giving students a choice about how they learn. We already set goa
ls; why not let them choose where to sit, and give them options of what to sit on?I teach at a low
-income (Title 1) school. Every year, I have a class with a range of abilities, yet they are all t
he same age. They learn differently, and they have different interests. Some have ADHD, and some a
re fast learners. Yet they are eager and active learners that want and need to be able to move aro
und the room, yet have a place that they can be comfortable to complete their work.We need a class
room rug that we can use as a class for reading time, and students can use during other learning t
imes. I have also requested four Kore Kids wobble chairs and four Back Jack padded portable chairs
so that students can still move during whole group lessons without disrupting the class. Having th
ese areas will provide these little ones with a way to wiggle while working.Benjamin Franklin once
said, \"Tell me and I forget, teach me and I may remember, involve me and I learn.\" I want these
children to be involved in their learning by having a choice on where to sit and how to learn, all
```

```
by giving them options for comfortable flexible seating.
====================================================
Who remembers middle school- the chaos and panic as you rocketed through puberty (or worse- hadn't
started it yet)? In my classroom, we try to immerse ourselves in literature, our own writing, or h
eated debate. Though sometimes we're (intentionally) chaotic, our class is a respite from the madn
ess.My students want to be heard. They constantly want to read out loud, tell me their opinions, a
nd spill out ALL their ideas in one run-on sentence. My students are an extremely diverse group of
kids, and they're proud of who they are.  Each kid has an astonishing backstory and they aren't af
raid to share it.  My school is in an area that typically has families of low SES. Sometimes I nee
d to give my students a pencil, a binder, or maybe even a shirt. The great thing about these kids
is that they don't let that bring them down. Every single child I teach is opinionated and in the
process of learning how to shape and share those opinions. They deserve every opportunity I can gi
ve them.My kids just need to hold books in their hands. They've expressed the desire to perform pl
ays (they love attention), because they zone out a little when just one of us reads a novel aloud.
Our school doesn't have any class sets of plays. I'd take any play, but we only have novels. I wan
t literature that can involve as many kids in the experience as possible. With a play, I can have
6-8 readers at once without the dreaded 'popcorn reading' of my youth. Our district, in order to s
upport differentiation, has a limit on how many books can be ordered through the school. Because o
f this, our library has a dwindling number of class sets. While I understand that and can work aro
und it, my students don't get why I can't just find them 30 copies of a play to read. This is a di
rect request from my kids: \"We just need some books!\"Over half of my school (and an even larger
percentage of my students) are new to English as a spoken language. Conquering words penned by Sha
kespeare can shape their understanding of English as a language and increase their confidence. Thi
s will help them increase their speaking and listening skills, but more importantly- push them out
of their comfort zones. Children who can gather the courage read a line in a play (when they USED
to be afraid to speak in class) can do anything.
====================================================
```

In [15]:

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

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

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

In [16]:

```python
sent = decontracted(project_data['essay'].values[20000])
```

In [17]:

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

In [18]:

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

In [19]:

```python
# 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', "do
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 [20]:

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

```
100%|██████████████████████████████████████| 25000/25000 [00:42<00:00, 589.26it/s]
```

In [21]:

```
# after preprocesing
project_data.head(3)
```

Out[21]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_cate
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2
23374	72317	p087808	598621c141cda5fb184ee7e8ccdd3fcc	Ms.	CA	2016-04-27 02:04:15	Grades PreK-2
7176	79341	p091436	bb2599c4a114d211b3381abe9f899bf8	Mrs.	OH	2016-04-27 07:24:47	Grades PreK-2

	Unnamed:						

1.4 Preprocessing of `project_title`

In [22]:

```
# printing some random essays.
print(project_data['project_title'].values[0])
print("="*50)
print(project_data['project_title'].values[150])
print("="*50)
print(project_data['project_title'].values[1000])
```

```
Flexible Seating for Flexible Learning
==================================================
Learning Language With Shakespeare
==================================================
Art: A hard days work.
```

In [23]:

```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_titles = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # 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())
project_data['preprocessed_titles'] = preprocessed_titles
```

```
100%|█████████████████████████████████| 25000/25000 [00:01<00:00, 22400.15it/s]
```

In [24]:

```
project_data.head(3)
```

Out[24]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_cate
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2
23374	72317	p087808	598621c141cda5fb184ee7e8ccdd3fcc	Ms.	CA	2016-04-27 02:04:15	Grades PreK-2
7176	79341	p091436	bb2599c4a114d211b3381abe9f899bf8	Mrs.	OH	2016-04-27 07:24:47	Grades PreK-2

1.4.1 Project_grade preprocessing

In [25]:

```
project_data['project_grade_category'][:4]
```

Out[25]:

```
473      Grades PreK-2
23374    Grades PreK-2
7176     Grades PreK-2
5145        Grades 3-5
Name: project_grade_category, dtype: object
```

In [26]:

```
project_data['project_grade_category'] = project_data['project_grade_category'].str.replace(" ", "_
")
project_data['project_grade_category'].value_counts()
```

Out[26]:

```
Grades_PreK-2    10186
Grades_3-5        8468
Grades_6-8        3875
Grades_9-12       2471
Name: project_grade_category, dtype: int64
```

Preprocessing teacher_prefix

In [27]:

```
project_data['teacher_prefix'][:4]
```

Out[27]:

```
473      Mrs.
23374     Ms.
7176     Mrs.
5145     Mrs.
Name: teacher_prefix, dtype: object
```

In [28]:

```
project_data['teacher_prefix'] = project_data['teacher_prefix'].str.replace(".","")
project_data['teacher_prefix'].value_counts()
```

Out[28]:

```
Mrs        13046
Ms          9019
Mr          2391
Teacher      543
Name: teacher_prefix, dtype: int64
```

1.5 Preparing data for models

In [29]:

```
project_data.columns
```

Out[29]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', 'project_title', 'project_essay_1',
       'project_essay_2', 'project_essay_3', 'project_essay_4',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay',
       'preprocessed_essays', 'preprocessed_titles'],
      dtype='object')
```

we are going to consider

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

       - preprocessed_titles: text data
       - preprocessed_essays : text data


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

Split data into train,test and Cross validate

In [30]:

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

In [31]:

```
X = project_data
X.head(1)
```

Out[31]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_categor
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs	GA	2016-04-27 00:53:00	Grades_PreK-2

In [32]:

```
# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, stratify=Y)
X_train, X_cv, Y_train, Y_cv = train_test_split(X_train, Y_train, stratify=Y_train)
```

1.5.1 Vectorizing Categorical data

One Hot Encode - Clean_Categories

In [33]:

```
# we use count vectorizer to convert the values into one hot encoded features
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True
)
vectorizer.fit(X_train['clean_categories'].values)
```

```
categories_one_hot_train = vectorizer.transform(X_train['clean_categories'].values)
categories_one_hot_test = vectorizer.transform(X_test['clean_categories'].values)
categories_one_hot_cv = vectorizer.transform(X_cv['clean_categories'].values)


print("Shape of Train data - one hot encoding ",categories_one_hot_train.shape)
print("Shape of Test data - one hot encoding ",categories_one_hot_test.shape)
print("Shape of CV data - one hot encoding ",categories_one_hot_cv.shape)
print(vectorizer.get_feature_names())
```

```
Shape of Train data - one hot encoding  (14062, 9)
Shape of Test data - one hot encoding  (6250, 9)
Shape of CV data - one hot encoding  (4688, 9)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
```

One Hot Encode - Clean_Sub-Categories

In [34]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=
True)

vectorizer.fit(X_train['clean_subcategories'].values)

sub_cat_one_hot_train = vectorizer.transform(X_train['clean_subcategories'].values)
sub_cat_one_hot_test = vectorizer.transform(X_test['clean_subcategories'].values)
sub_cat_one_hot_cv = vectorizer.transform(X_cv['clean_subcategories'].values)


print("Shape of Train data - one hot encoding ",sub_cat_one_hot_train.shape)
print("Shape of Test data - one hot encoding",sub_cat_one_hot_test.shape)
print("Shape of CV data - one hot encoding",sub_cat_one_hot_cv.shape)
print(vectorizer.get_feature_names())
```

```
Shape of Train data - one hot encoding  (14062, 30)
Shape of Test data - one hot encoding (6250, 30)
Shape of CV data - one hot encoding (4688, 30)
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'ESL',
'EarlyDevelopment', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness',
'AppliedSciences', 'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
```

One Hot Encode - School_States

In [35]:

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

In [36]:

```
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 [37]:

```
## 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(X_train['school_state'].values)
```

```
school_state_one_hot_train = vectorizer.transform(X_train['school_state'].values)
school_state_one_hot_test = vectorizer.transform(X_test['school_state'].values)
school_state_one_hot_cv = vectorizer.transform(X_cv['school_state'].values)

print("Shape of Train data - one hot encoding",school_state_one_hot_train.shape)
print("Shape of Test data - one hot encoding",school_state_one_hot_test.shape)
print("Shape of CV data - one hot encoding",school_state_one_hot_cv.shape)
print(vectorizer.get_feature_names())
```

```
Shape of Train data - one hot encoding (14062, 51)
Shape of Test data - one hot encoding (6250, 51)
Shape of CV data - one hot encoding (4688, 51)
['VT', 'WY', 'ND', 'MT', 'RI', 'NH', 'NE', 'SD', 'AK', 'DE', 'ME', 'HI', 'NM', 'WV', 'DC', 'ID', 'I
A', 'KS', 'AR', 'CO', 'KY', 'MN', 'MS', 'OR', 'NV', 'MD', 'AL', 'UT', 'CT', 'TN', 'WI', 'VA', 'NJ',
'AZ', 'OK', 'MA', 'WA', 'MO', 'LA', 'OH', 'IN', 'PA', 'MI', 'GA', 'SC', 'IL', 'NC', 'FL', 'TX', 'NY
', 'CA']
```

One Hot Encode - Project_Grade_Category

In [38]:

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

In [39]:

```
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 [40]:

```
## 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(X_train['project_grade_category'].values)

project_grade_cat_one_hot_train = vectorizer.transform(X_train['project_grade_category'].values)
project_grade_cat_one_hot_test = vectorizer.transform(X_test['project_grade_category'].values)
project_grade_cat_one_hot_cv = vectorizer.transform(X_cv['project_grade_category'].values)


print("Shape of Train data - one hot encoding",project_grade_cat_one_hot_train.shape)
print("Shape of Test data - one hot encoding",project_grade_cat_one_hot_test.shape)
print("Shape of CV data - one hot encoding",project_grade_cat_one_hot_cv.shape)
print(vectorizer.get_feature_names())
```

```
Shape of Train data - one hot encoding (14062, 4)
Shape of Test data - one hot encoding (6250, 4)
Shape of CV data - one hot encoding (4688, 4)
['Grades_9-12', 'Grades_6-8', 'Grades_3-5', 'Grades_PreK-2']
```

One Hot Encode - Teacher_Prefix

In [41]:

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

In [42]:

```
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 [43]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_teacher_prefix_cat_dict.keys()), lowercase=False, binary=True)
vectorizer.fit(X_train['teacher_prefix'].values.astype("U"))

teacher_prefix_cat_one_hot_train = vectorizer.transform(X_train['teacher_prefix'].values.astype("U"))
teacher_prefix_cat_one_hot_test = vectorizer.transform(X_test['teacher_prefix'].values.astype("U"))
teacher_prefix_cat_one_hot_cv = vectorizer.transform(X_cv['teacher_prefix'].values.astype("U"))

print("Shape of Train data - one hot encoding",teacher_prefix_cat_one_hot_train.shape)
print("Shape of Test data - one hot encoding ",teacher_prefix_cat_one_hot_test.shape)
print("Shape of CV data - one hot encoding ",teacher_prefix_cat_one_hot_cv.shape)

print(vectorizer.get_feature_names())
```

```
Shape of Train data - one hot encoding (14062, 5)
Shape of Test data - one hot encoding  (6250, 5)
Shape of CV data - one hot encoding  (4688, 5)
['nan', 'Teacher', 'Mr', 'Ms', 'Mrs']
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

BOW of eassys - FOr Train/Test/CV Datasets

In [44]:

```
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(X_train['preprocessed_essays'])

# BOW for essays Train Data
essay_bow_train = vectorizer.transform(X_train['preprocessed_essays'])
print(essay_bow_train.shape)

# BOW for essays Test Data
essay_bow_test = vectorizer.transform(X_test['preprocessed_essays'])
print(essay_bow_test.shape)

# BOW for essays CV Data
essay_bow_cv = vectorizer.transform(X_cv['preprocessed_essays'])
print(essay_bow_cv.shape)
```

```
(14062, 7135)
(6250, 7135)
(4688, 7135)
```

BOW of Project Titles - Train/Test/CV Data¶

In [45]:

```
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(X_train['preprocessed_titles'])

# BOW for title Train Data
title_bow_train = vectorizer.transform(X_train['preprocessed_titles'])
print(title_bow_train.shape)

# BOW for title Test Data
title_bow_test = vectorizer.transform(X_test['preprocessed_titles'])
print(title_bow_test.shape)

# BOW for title CV Data
title_bow_cv = vectorizer.transform(X_cv['preprocessed_titles'])
print(title_bow_cv.shape)
```

```
(14062, 863)
(6250, 863)
(4688, 863)
```

1.5.2.2 TFIDF vectorizer for essay

In [46]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['preprocessed_essays'])

#tidf Train Data
essay_tfidf_train = vectorizer.transform(X_train['preprocessed_essays'])
print(essay_tfidf_train.shape)

#tidf Test Data
essay_tfidf_test = vectorizer.transform(X_test['preprocessed_essays'])
print(essay_tfidf_test.shape)

#tidf CV Data
essay_tfidf_cv = vectorizer.transform(X_cv['preprocessed_essays'])
print(essay_tfidf_cv.shape)
```

```
(14062, 7135)
(6250, 7135)
(4688, 7135)
```

TFIDF vectorizer for Title

In [47]:

```
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['preprocessed_titles'])

#tidf Train Data
title_tfidf_train = vectorizer.transform(X_train['preprocessed_titles'])
print(title_tfidf_train.shape)

#tidf Test Data
title_tfidf_test = vectorizer.transform(X_test['preprocessed_titles'])
print(title_tfidf_test.shape)

#tidf CV Data
title_tfidf_cv = vectorizer.transform(X_cv['preprocessed_titles'])
print(title_tfidf_cv.shape)
```

```
(14062, 863)
(6250, 863)
(4688, 863)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [48]:

```
# 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 [49]:

```
# average Word2Vec Function
# computing average word2vec for each review.
# the avg-w2v for each sentence/review is stored in this list
```

```
def avg_w2v_vectors_func(sentance):
    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
    return vector
```

Train/Test/CV Data - Avg-W2V for essay

In [50]:

```
essay_avg_w2v_train = []
essay_avg_w2v_test  = []
essay_avg_w2v_cv    = []
# Avg-w2v for Train data
for sentence in tqdm(X_train['preprocessed_essays']):
    essay_avg_w2v_train.append(avg_w2v_vectors_func(sentance))

# Avg-w2v for Train data
print("len(essay_avg_w2v_train):",len(essay_avg_w2v_train))
print("len(essay_avg_w2v_train[0])",len(essay_avg_w2v_train[0]))
# Avg-w2v for Test data
for sentence in tqdm(X_test['preprocessed_essays']):
    essay_avg_w2v_test.append(avg_w2v_vectors_func(sentance))


print("len(essay_avg_w2v_test):",len(essay_avg_w2v_test))
print("len(essay_avg_w2v_test[0])",len(essay_avg_w2v_test[0]))

# Avg-w2v for CV data
for sentence in tqdm(X_cv['preprocessed_essays']):
    essay_avg_w2v_cv.append(avg_w2v_vectors_func(sentance))

# Avg-w2v for CV data
print("len(essay_avg_w2v_cv):",len(essay_avg_w2v_cv))
print("len(essay_avg_w2v_cv[0])",len(essay_avg_w2v_cv[0]))
```

```
100%|███████████████████████████████████| 14062/14062 [00:06<00:00, 2192.95it/s]
```

```
len(essay_avg_w2v_train): 14062
len(essay_avg_w2v_train[0]) 300
```

```
100%|█████████████████████████████████████| 6250/6250 [00:02<00:00, 2343.32it/s]
```

```
len(essay_avg_w2v_test): 6250
len(essay_avg_w2v_test[0]) 300
```

```
100%|█████████████████████████████████████| 4688/4688 [00:02<00:00, 2083.44it/s]
```

```
len(essay_avg_w2v_cv): 4688
len(essay_avg_w2v_cv[0]) 300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [51]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['preprocessed_essays'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [52]:

```
# Compute  TFIDF weighted W2V for each sentence of the review.

def tf_idf_weight_func(sentence): # 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
    return vector
```

Train/Test/CV Data - TFIDF weighted W2V for essay

In [53]:

```
essay_tfidf_w2v_train = []
essay_tfidf_w2v_test  = []
essay_tfidf_w2v_cv    = []
#  TFIDF weighted W2V for Train data
for sentence in tqdm(X_train['preprocessed_essays']):
    essay_tfidf_w2v_train.append(tf_idf_weight_func(sentance))
print("len(essay_tfidf_w2v_train)",len(essay_tfidf_w2v_train))
print("len(essay_tfidf_w2v_train[0])",len(essay_tfidf_w2v_train[0]))

#  TFIDF weighted W2V for Test data
for sentence in tqdm(X_test['preprocessed_essays']):
    essay_tfidf_w2v_test.append(tf_idf_weight_func(sentance))
print("len(essay_tfidf_w2v_test)",len(essay_tfidf_w2v_test))
print("len(essay_tfidf_w2v_test[0])",len(essay_tfidf_w2v_test[0]))

#  TFIDF weighted W2V for CV data
for sentence in tqdm(X_cv['preprocessed_essays']):
    essay_tfidf_w2v_cv.append(tf_idf_weight_func(sentance))
print("len(essay_tfidf_w2v_cv)",len(essay_tfidf_w2v_cv))
print("len(essay_tfidf_w2v_cv[0])",len(essay_tfidf_w2v_cv[0]))
```

```
100%|████████████████████████████████| 14062/14062 [00:00<00:00, 209868.18it/s]
```

```
len(essay_tfidf_w2v_train) 14062
len(essay_tfidf_w2v_train[0]) 300
```

```
100%|██████████████████████████████████| 6250/6250 [00:00<00:00, 215504.51it/s]
```

```
len(essay_tfidf_w2v_test) 6250
len(essay_tfidf_w2v_test[0]) 300
```

```
100%|██████████████████████████████████| 4688/4688 [00:00<00:00, 146489.88it/s]
```

```
len(essay_tfidf_w2v_cv) 4688
len(essay_tfidf_w2v_cv[0]) 300
```

Train/Test/CV Data - Avg-W2V for essay

In [54]:

```
title_avg_w2v_train = []
title_avg_w2v_test  = []
title_avg_w2v_cv    = []

for sentence in tqdm(X_train['preprocessed_titles']):
    title_avg_w2v_train.append(avg_w2v_vectors_func(sentance)) # Avg-w2v for Train data

# Avg-w2v for Train data
# Avg-w2v for Train data
print("len(title_avg_w2v_train):",len(title_avg_w2v_train))
print("len(title_avg_w2v_train[0])",len(title_avg_w2v_train[0]))

for sentence in tqdm(X_test['preprocessed_titles']):
    title_avg_w2v_test.append(avg_w2v_vectors_func(sentance)) # Avg-w2v for Test data

# Avg-w2v for Test data
print("len(title_avg_w2v_test):",len(title_avg_w2v_test))
print("len(title_avg_w2v_test[0])",len(title_avg_w2v_test[0]))


for sentence in tqdm(X_cv['preprocessed_titles']):
    title_avg_w2v_cv.append(avg_w2v_vectors_func(sentance)) # Avg-w2v for CV data

# Avg-w2v for CV data
print("len(title_avg_w2v_cv):",len(title_avg_w2v_cv))
print("len(title_avg_w2v_cv[0])",len(title_avg_w2v_cv[0]))
```

```
100%|█████████████████████████████████| 14062/14062 [00:00<00:00, 38003.19it/s]
```

```
len(title_avg_w2v_train): 14062
len(title_avg_w2v_train[0]) 300
```

```
100%|███████████████████████████████████| 6250/6250 [00:00<00:00, 39305.96it/s]
```

```
len(title_avg_w2v_test): 6250
len(title_avg_w2v_test[0]) 300
```

```
100%|███████████████████████████████████| 4688/4688 [00:00<00:00, 40066.14it/s]
```

```
len(title_avg_w2v_cv): 4688
len(title_avg_w2v_cv[0]) 300
```

Train/Test/CV Data - TFIDF weighted W2V for Project Titles

In [55]:

```
title_tfidf_w2v_train  = []
title_tfidf_w2v_test   = []
title_tfidf_w2v_cv     = []

for sentence in tqdm(X_train['preprocessed_titles']):
    title_tfidf_w2v_train.append(tf_idf_weight_func(sentance)) #  TFIDF weighted W2V for Train data
print("len(title_tfidf_w2v_train)",len(title_tfidf_w2v_train))
print("len(title_tfidf_w2v_train[0])",len(title_tfidf_w2v_train[0]))

for sentence in tqdm(X_test['preprocessed_titles']):
    title_tfidf_w2v_test.append(tf_idf_weight_func(sentance)) #  TFIDF weighted W2V for Test data
print("len(title_tfidf_w2v_test)",len(title_tfidf_w2v_test))
print("len(title_tfidf_w2v_test[0])",len(title_tfidf_w2v_test[0]))

for sentence in tqdm(X_cv['preprocessed_titles']):
    title_tfidf_w2v_cv.append(tf_idf_weight_func(sentance)) #  TFIDF weighted W2V for CV data
print("len(title_tfidf_w2v_cv)",len(title_tfidf_w2v_cv))
print("len(title_tfidf_w2v_cv[0])",len(title_tfidf_w2v_cv[0]))
```

```
100%|████████████████████████████████| 14062/14062 [00:00<00:00, 206782.28it/s]
```

```
len(title_tfidf_w2v_train) 14062
len(title_tfidf_w2v_train[0]) 300
```

```
100%|██████████████████████████████████| 6250/6250 [00:00<00:00, 173601.85it/s]
```

```
len(title_tfidf_w2v_test) 6250
len(title_tfidf_w2v_test[0]) 300
```

```
100%|██████████████████████████████████| 4688/4688 [00:00<00:00, 203817.62it/s]
```

```
len(title_tfidf_w2v_cv) 4688
len(title_tfidf_w2v_cv[0]) 300
```

1.5.3 Vectorizing Numerical features

In [56]:

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
price_data.head(3)
```

Out[56]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21
2	p000003	298.97	4

In [57]:

```
X_train = pd.merge(X_train, price_data, on='id', how='left')
X_test = pd.merge(X_test, price_data, on='id', how='left')
X_cv = pd.merge(X_cv, price_data, on='id', how='left')
```

In [58]:

```
from sklearn.preprocessing import Normalizer

normalizer = Normalizer()

# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

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

price_data_train = normalizer.transform(X_train['price'].values.reshape(-1,1))

price_data_test = normalizer.transform(X_test['price'].values.reshape(-1,1))

price_data_cv = normalizer.transform(X_cv['price'].values.reshape(-1,1))

print("After vectorizations")
print("="*100)
print(price_data_train.shape, Y_train.shape)
print(price_data_test.shape, Y_test.shape)
print(price_data_cv.shape, Y_cv.shape)
print("="*100)
```

```
After vectorizations
====================================================================================================

(14062, 1) (14062,)
(6250, 1) (6250,)
(4688, 1) (4688,)
====================================================================================================
```

Vectorizing Quantity Feature

In [59]:

```
normalizer = Normalizer()
```

```
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

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

quant_train = normalizer.transform(X_train['quantity'].values.reshape(-1,1))
quant_cv = normalizer.transform(X_cv['quantity'].values.reshape(-1,1))
quant_test = normalizer.transform(X_test['quantity'].values.reshape(-1,1))

print("="*100)
print("After vectorizations")
print(quant_train.shape, Y_train.shape)
print(quant_cv.shape, Y_cv.shape)
print(quant_test.shape, Y_test.shape)
print("="*100)
```

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

After vectorizations
(14062, 1) (14062,)
(4688, 1) (4688,)
(6250, 1) (6250,)
====================================================================================================
```

Vectorizing teacher_number_of_previously_posted_projects

In [60]:

```
normalizer = Normalizer()

# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

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

prev_no_projects_train = 
normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
prev_no_projects_cv = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].va
lues.reshape(-1,1))
prev_no_projects_test = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'
].values.reshape(-1,1))

print("="*100)
print("After vectorizations")
print(prev_no_projects_train.shape, Y_train.shape)
print(prev_no_projects_cv.shape, Y_cv.shape)
print(prev_no_projects_test.shape, Y_test.shape)
print("="*100)
```

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

After vectorizations
(14062, 1) (14062,)
(4688, 1) (4688,)
(6250, 1) (6250,)
====================================================================================================
```

Assignment 3: Apply KNN

1. [Task 1] Apply KNN(brute force version) on these feature sets

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

2.4.1 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

X_train_merge = hstack((categories_one_hot_train, sub_cat_one_hot_train,
school_state_one_hot_train, project_grade_cat_one_hot_train, teacher_prefix_cat_one_hot_train, pri
ce_data_train, quant_train, prev_no_projects_train,title_bow_train, essay_bow_train)).tocsr()
```

```
ce_data_train, quant_train, prev_no_projects_train,title_bow_train, essay_bow_train)).tocsr()
X_test_merge = hstack((categories_one_hot_test, sub_cat_one_hot_test, school_state_one_hot_test, p
roject_grade_cat_one_hot_test, teacher_prefix_cat_one_hot_test, price_data_test, quant_test,
prev_no_projects_test,title_bow_test, essay_bow_test)).tocsr()
X_cv_merge = hstack((categories_one_hot_cv, sub_cat_one_hot_cv,
school_state_one_hot_cv,project_grade_cat_one_hot_cv, teacher_prefix_cat_one_hot_cv, price_data_cv
, quant_cv, prev_no_projects_cv,title_bow_cv, essay_bow_cv)).tocsr()

print("Final Data matrix")
print("="*100)
print(X_train_merge.shape, Y_train.shape)
print(X_cv_merge.shape, Y_test.shape)
print(X_test_merge.shape, Y_cv.shape)
print("="*100)
```

```
Final Data matrix
====================================================================================================

(14062, 8100) (14062,)
(4688, 8100) (6250,)
(6250, 8100) (4688,)
====================================================================================================
```

Best hyper prameter using the ROC/AUC higest value and K-fold cross validation.

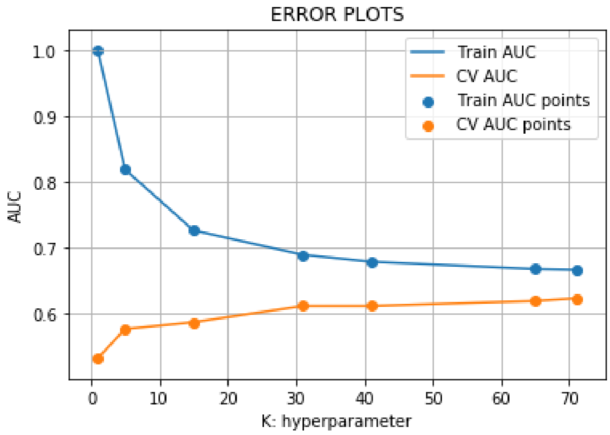
In [62]:

```
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 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 = []

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

for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train_merge, Y_train)

    y_train_pred = batch_predict(neigh, X_train_merge)
    y_cv_pred = batch_predict(neigh, X_cv_merge)
    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("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████████████████████████████████████████| 7/7 [05:10<00:00, 42.02s/it]
```



Best Hyper paramter using the grid search.

In [64]:

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


neigh = KNeighborsClassifier()

parameters = {'n_neighbors':[1, 5,  15, 31, 41,  65, 71]}

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

In [65]:

```
clf.fit(X_train_merge, Y_train)
```

Out[65]:

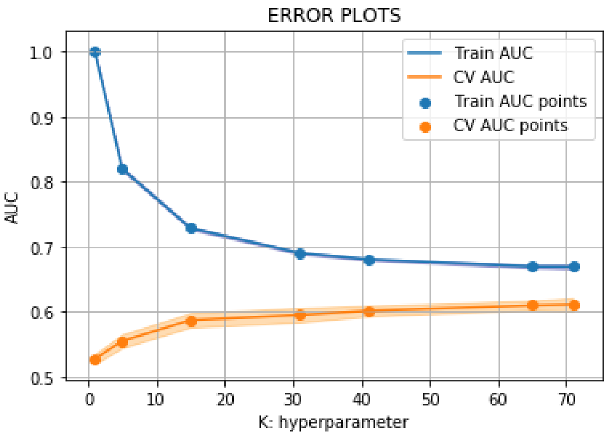
```
GridSearchCV(cv=5, error_score='raise',
       estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform'),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'n_neighbors': [1, 5, 15, 31, 41, 65, 71]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring='roc_auc', verbose=0)
```

In [66]:

```
train_auc= clf.cv_results_['mean_train_score']
```

In [67]:

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

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 [68]:

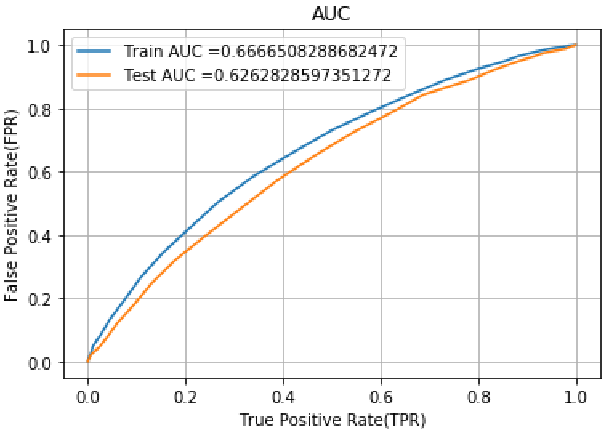
```
score_t_cv = [x for x in cv_auc]
opt_t_cv = K[score_t_cv.index(max(score_t_cv))]
print("Maximum AUC score of cv is:" + ' ' + str(max(score_t_cv)))
print("Corresponding k value of cv is:",opt_t_cv, '\n')
best_k=opt_t_cv
print(best_k)
```

```
Maximum AUC score of cv is: 0.6105877864656477
Corresponding k value of cv is: 71

71
```

In [69]:

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

neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X_train_merge, 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_merge)
y_test_pred = batch_predict(neigh, X_test_merge)

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)
```

```
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()
```



Confusion Matrix

In [70]:

```
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

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

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

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

In [71]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(Y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(Y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
print("="*100)
```

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2499980457367095 for threshold 0.775
[[1070 1076]
 [3197 8719]]
====================================================================================================
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24901111506665086 for threshold 0.803
[[ 583  371]
 [2260 3036]]
====================================================================================================
```

Confusion Matrix -Heat map - Train

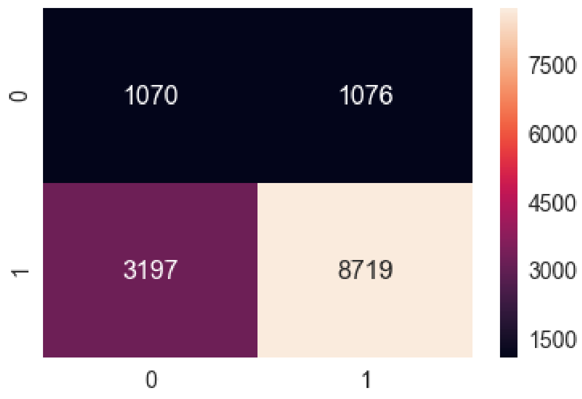
In [72]:

```
conf_mat_BOW_train = pd.DataFrame(confusion_matrix(Y_train, predict(y_train_pred, tr_thresholds,
train_fpr, train_fpr)), range(2),range(2))
sns.set(font_scale=1.4)
sns.heatmap(conf_mat_BOW_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.2499980457367095 for threshold 0.775
```

Out[72]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x12587780>
```



Confusion Matrix -Heat map - test

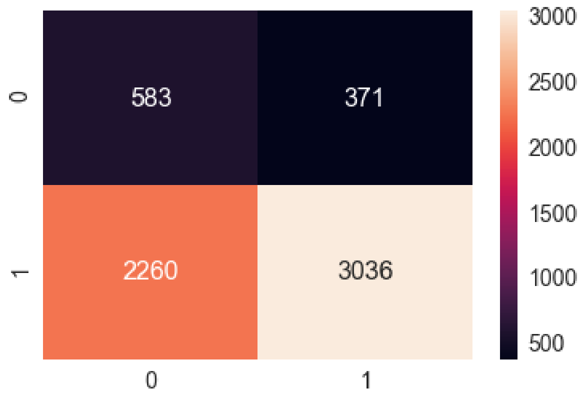
In [73]:

```
conf_mat_BOW_test= pd.DataFrame(confusion_matrix(Y_test, predict(y_test_pred, tr_thresholds, test_f
pr, test_fpr)), range(2),range(2))
sns.set(font_scale=1.4)
sns.heatmap(conf_mat_BOW_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24901111506665086 for threshold 0.803
```

Out[73]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x125e9470>
```



2.4.2 Applying KNN brute force on TFIDF, SET 2

In [74]:

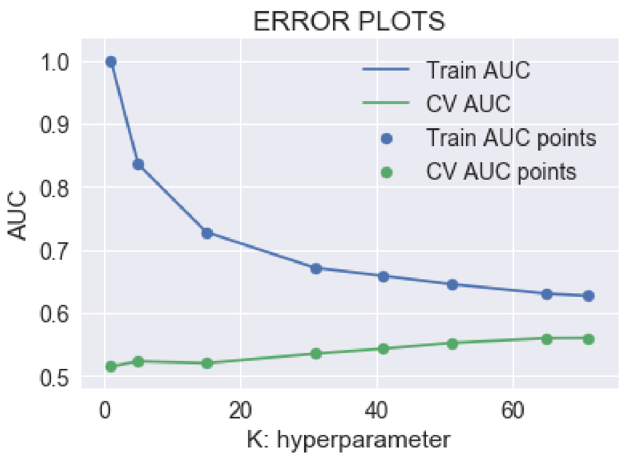
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_train_merge = hstack((categories_one_hot_train, sub_cat_one_hot_train,
school_state_one_hot_train, project_grade_cat_one_hot_train, teacher_prefix_cat_one_hot_train, pri
ce_data_train, quant_train, prev_no_projects_train,title_tfidf_train, essay_tfidf_train)).tocsr()
X_test_merge = hstack((categories_one_hot_test, sub_cat_one_hot_test, school_state_one_hot_test, p
roject_grade_cat_one_hot_test, teacher_prefix_cat_one_hot_test, price_data_test, quant_test,
prev_no_projects_test,title_tfidf_test, essay_tfidf_test)).tocsr()
X_cv_merge = hstack((categories_one_hot_cv, sub_cat_one_hot_cv,
school_state_one_hot_cv,project_grade_cat_one_hot_cv, teacher_prefix_cat_one_hot_cv, price_data_cv
, quant_cv, prev_no_projects_cv,title_tfidf_cv, essay_tfidf_cv)).tocsr()

print("Final Data matrix")
print("="*100)
print(X_train_merge.shape, Y_train.shape)
print(X_cv_merge.shape, Y_test.shape)
print(X_test_merge.shape, Y_cv.shape)
print("="*100)
```

```
Final Data matrix
====================================================================================================

(14062, 8100) (14062,)
(4688, 8100) (6250,)
(6250, 8100) (4688,)
====================================================================================================
```

In [75]:

```
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 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 = []

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

for i in tqdm(K):

    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train_merge, Y_train)

    y_train_pred = batch_predict(neigh, X_train_merge)
    y_cv_pred = batch_predict(neigh, X_cv_merge)

    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("ERROR PLOTS")
plt.grid(True)
plt.show()
```

```
100%|█████████████████████████████████████████████| 8/8 [07:01<00:00, 55.26s/it]
```



In [76]:

```
scor = [x for x in cv_auc]
opt_t_cv_2 = K[scor.index(max(scor))]
print("Maximum AUC score of cv is:" + ' ' + str(max(scor)))
print("Corresponding k value of cv is:",opt_t_cv_2, '\n')
```

```
Maximum AUC score of cv is: 0.5604203235497647
Corresponding k value of cv is: 71
```

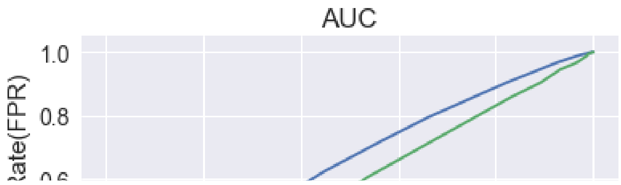
In [77]:

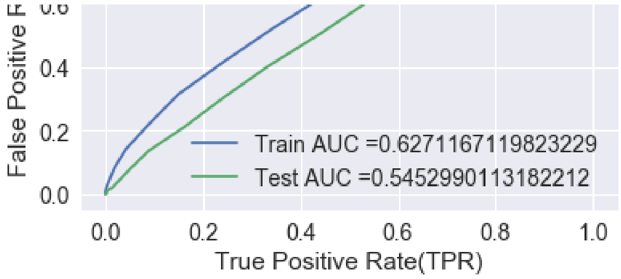
```
best_k=opt_t_cv_2
```

Train Model using the best value of K

In [78]:

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

neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X_train_merge, 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_merge)
y_test_pred = batch_predict(neigh, X_test_merge)

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(True)
plt.show()
```





Confusion Matrix

In [79]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(Y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(Y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
print("="*100)
```

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24717804380850303 for threshold 0.845
[[1187  959]
 [4498 7418]]
====================================================================================================

Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24849579614027223 for threshold 0.831
[[ 337  617]
 [1591 3705]]
====================================================================================================
```

Confusion Matrix - Heat map -train.

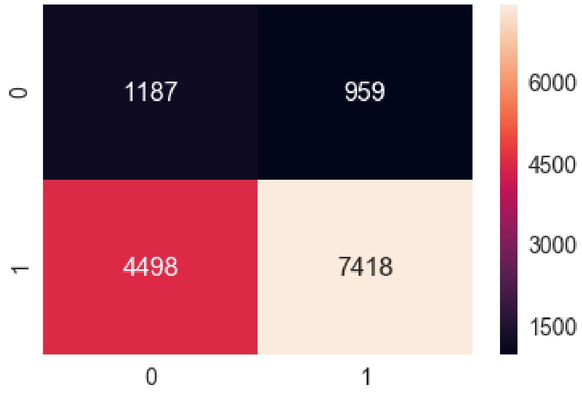
In [80]:

```
conf_matr_df_tfidf_train = pd.DataFrame(confusion_matrix(Y_train, predict(y_train_pred,
tr_thresholds, train_fpr, train_fpr)), range(2),range(2))
sns.set(font_scale=1.4)
sns.heatmap(conf_matr_df_tfidf_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24717804380850303 for threshold 0.845
```

Out[80]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x14133898>
```



Confusion Matrix - Heat map - TEST data.

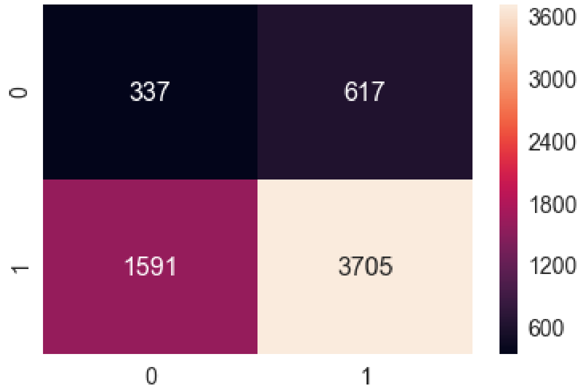
In [81]:

```
conf_matr_df_tfidf_test = pd.DataFrame(confusion_matrix(Y_test, predict(y_test_pred, tr_thresholds,
test_fpr, test_fpr)), range(2),range(2))
sns.set(font_scale=1.4)
sns.heatmap(conf_matr_df_tfidf_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24849579614027223 for threshold 0.831
```

Out[81]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x141c4dd8>
```



2.4.3 Applying KNN brute force on AVG W2V, SET 3

In [82]:

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

X_train_merge = hstack((categories_one_hot_train, sub_cat_one_hot_train,
school_state_one_hot_train, project_grade_cat_one_hot_train, teacher_prefix_cat_one_hot_train, pri
ce_data_train, quant_train, prev_no_projects_train,title_avg_w2v_train,
essay_avg_w2v_train)).tocsr()
X_test_merge = hstack((categories_one_hot_test, sub_cat_one_hot_test, school_state_one_hot_test, p
roject_grade_cat_one_hot_test, teacher_prefix_cat_one_hot_test, price_data_test, quant_test,
prev_no_projects_test,title_avg_w2v_test, essay_avg_w2v_test)).tocsr()
X_cv_merge = hstack((categories_one_hot_cv, sub_cat_one_hot_cv,
school_state_one_hot_cv,project_grade_cat_one_hot_cv, teacher_prefix_cat_one_hot_cv, price_data_cv
, quant_cv, prev_no_projects_cv,title_avg_w2v_cv, essay_avg_w2v_cv)).tocsr()

print("Final Data matrix")
print("="*100)
print(X_train_merge.shape, Y_train.shape)
print(X_cv_merge.shape, Y_test.shape)
print(X_test_merge.shape, Y_cv.shape)
print("="*100)
```

```
Final Data matrix
====================================================================================================

(14062, 702) (14062,)
(4688, 702) (6250,)
(6250, 702) (4688,)
====================================================================================================
```

In [83]:

```
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 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 = []
cnt=0
K = [1, 5, 21, 41, 51,61]

for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    print ("1")
    neigh.fit(X_train_merge, Y_train)
    print ("2")

    y_train_pred = batch_predict(neigh, X_train_merge)
    print("3")
    y_cv_pred = batch_predict(neigh, X_cv_merge)

    train_auc.append(roc_auc_score(Y_train,y_train_pred))
    cv_auc.append(roc_auc_score(Y_cv, y_cv_pred))
    cnt=cnt+1
    print ("Loop",cnt)

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()
```

```
  0%|                                            | 0/6 [00:00<?, ?it/s]
```

```
1
2
3
Loop 1
```

```
 17%|███████                                     | 1/6 [11:46<58:52, 706.44s/it]
```

```
1
2
3
Loop 2
```

```
 33%|██████████████                              | 2/6 [26:09<50:13, 753.28s/it]
```

```
1
2
3
Loop 3
```

```
 50%|██████████████████████                      | 3/6 [36:48<35:57, 719.12s/it]
```

```
1
2
3
Loop 4
```

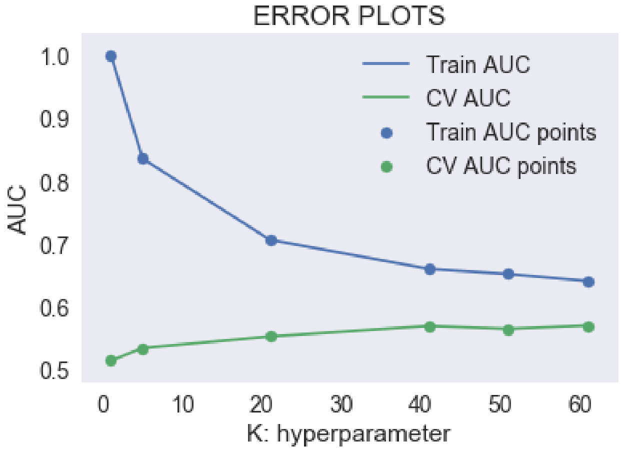
```
 67%|████████████████████████████             | 4/6 [51:32<25:37, 768.75s/it]
```

```
1
2
3
Loop 5
```

```
 83%|█████████████████████████████████        | 5/6 [1:03:36<12:35, 755.32s/it]
```

```
1
2
3
Loop 6
```

```
100%|████████████████████████████████████████| 6/6 [1:19:22<00:00, 812.27s/it]
```



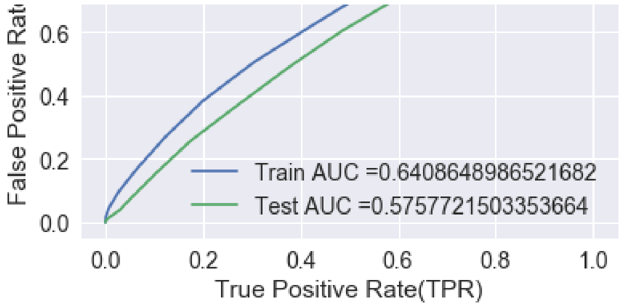
scor = [x for x in cv_auc] opt_t_cv_3 = K[scor.index(max(scor))] print("Maximum AUC score of cv is:" + ' ' + str(max(scor))) print("Corresponding k value of cv is:",opt_t_cv_3, '\n')

best_k=opt_t_cv_3

In [85]:

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

neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X_train_merge, 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_merge)
y_test_pred = batch_predict(neigh, X_test_merge)

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(True)
plt.show()
```

AUC
1.0
0.8
e(FPR)



Confusion Matrix

In [86]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(Y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(Y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
print("="*100)
```

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24929451095212576 for threshold 0.836
[[1016 1130]
 [3365 8551]]
====================================================================================================

Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24984177841066413 for threshold 0.852
[[ 489  465]
 [2087 3209]]
====================================================================================================
```

Confusion Matrix - heat map - Train

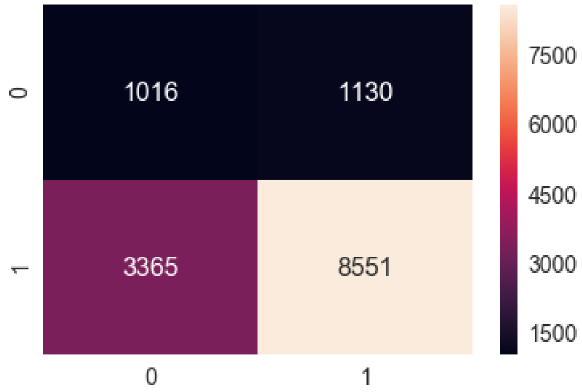
In [87]:

```
conf_matr_df_avgw2v_train = pd.DataFrame(confusion_matrix(Y_train, predict(y_train_pred,
tr_thresholds, train_fpr, train_fpr)), range(2),range(2))
sns.set(font_scale=1.4)
sns.heatmap(conf_matr_df_avgw2v_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24929451095212576 for threshold 0.836
```

Out[87]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x142006d8>
```



Confusion Matrix - heat map - Test

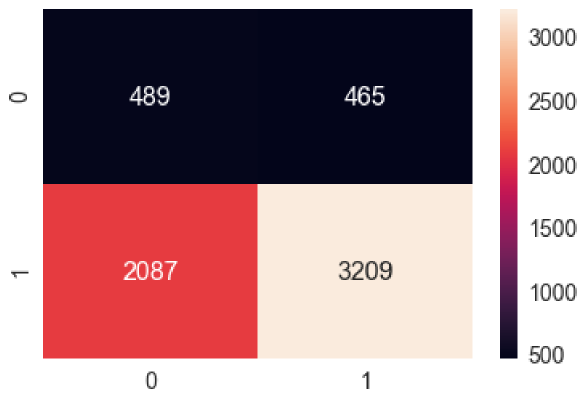
In [88]:

```
conf_matr_df_avgw2v_test = pd.DataFrame(confusion_matrix(Y_test, predict(y_test_pred, tr_thresholds
, test_fpr, test_fpr)), range(2),range(2))
sns.set(font_scale=1.4)
sns.heatmap(conf_matr_df_avgw2v_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24984177841066413 for threshold 0.852
```

Out[88]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x14975518>
```



2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

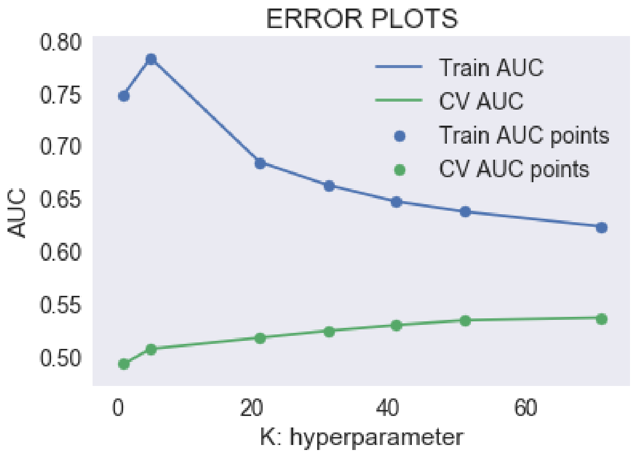
In [89]:

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

X_train_merge = hstack((categories_one_hot_train, sub_cat_one_hot_train,
school_state_one_hot_train, project_grade_cat_one_hot_train, teacher_prefix_cat_one_hot_train, pri
ce_data_train, quant_train, prev_no_projects_train,title_tfidf_w2v_train, essay_tfidf_w2v_train)).
tocsr()
X_test_merge = hstack((categories_one_hot_test, sub_cat_one_hot_test, school_state_one_hot_test, p
roject_grade_cat_one_hot_test, teacher_prefix_cat_one_hot_test, price_data_test, quant_test,
prev_no_projects_test,title_tfidf_w2v_test, essay_tfidf_w2v_test)).tocsr()
X_cv_merge = hstack((categories_one_hot_cv, sub_cat_one_hot_cv,
school_state_one_hot_cv,project_grade_cat_one_hot_cv, teacher_prefix_cat_one_hot_cv, price_data_cv
, quant_cv, prev_no_projects_cv,title_tfidf_w2v_cv, essay_tfidf_w2v_cv)).tocsr()

print("Final Data matrix")
print("="*100)
print(X_train_merge.shape, Y_train.shape)
print(X_cv_merge.shape, Y_test.shape)
print(X_test_merge.shape, Y_cv.shape)
print("="*100)
```

```
Final Data matrix
====================================================================================================

(14062, 702) (14062,)
(4688, 702) (6250,)
(6250, 702) (4688,)
====================================================================================================
```

In [90]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_auc_score

"""
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.

"""
# I am running on fewer k only as looping is taking atleast 4 hours for me
train_auc = []
cv_auc = []

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

for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train_merge, Y_train)

    y_train_pred = batch_predict(neigh, X_train_merge)
    y_cv_pred = batch_predict(neigh, X_cv_merge)

    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("ERROR PLOTS")
plt.grid()
plt.show()
```



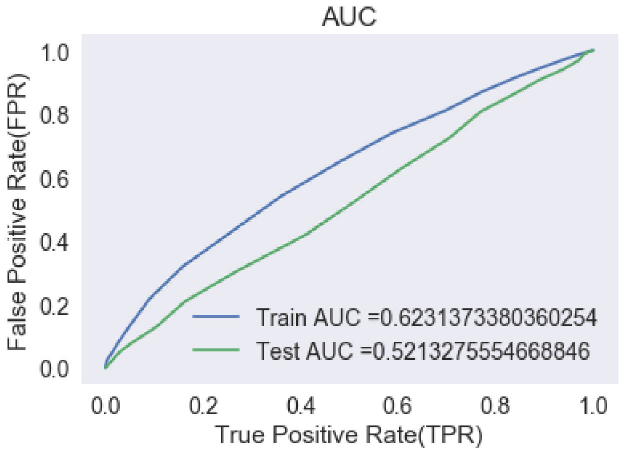
In [91]:

```
scor = [x for x in cv_auc]
opt_t_cv_3 = K[scor.index(max(scor))]
print("Maximum AUC score of cv is:" + ' ' + str(max(scor)))
print("Corresponding k value of cv is:",opt_t_cv_3, '\n')
best_k=opt_t_cv_3
```

```
Maximum AUC score of cv is: 0.5365617985113673
Corresponding k value of cv is: 71
```

In [92]:

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

neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X_train_merge, 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_merge)
y_test_pred = batch_predict(neigh, X_test_merge)

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()
```



Confusion Matrix

In [93]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(Y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(Y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
print("="*100)
```

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2497489857373522 for threshold 0.845
[[1107 1039]
 [4143 7773]]
====================================================================================================

Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24996044460266603 for threshold 0.845
[[ 377  577]
 [1982 3314]]
====================================================================================================
```

Confusion matrix -heat map - train

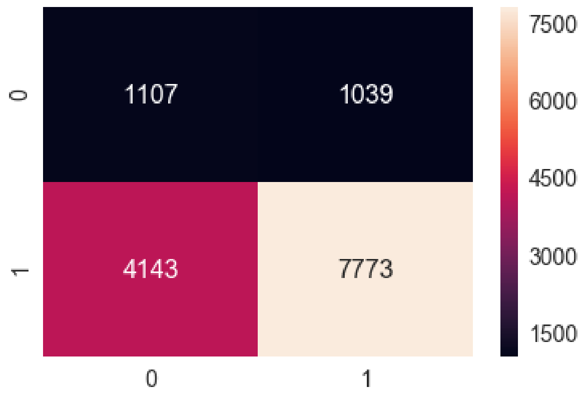
In [94]:

```
confusion_matr_tfidf_w2v_train = pd.DataFrame(confusion_matrix(Y_train, predict(y_train_pred,
tr_thresholds, train_fpr, train_fpr)), range(2),range(2))

sns.set(font_scale=1.4)
sns.heatmap(confusion_matr_tfidf_w2v_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.2497489857373522 for threshold 0.845
```

Out[94]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x149b24e0>
```



Confusion matrix -heat map - test

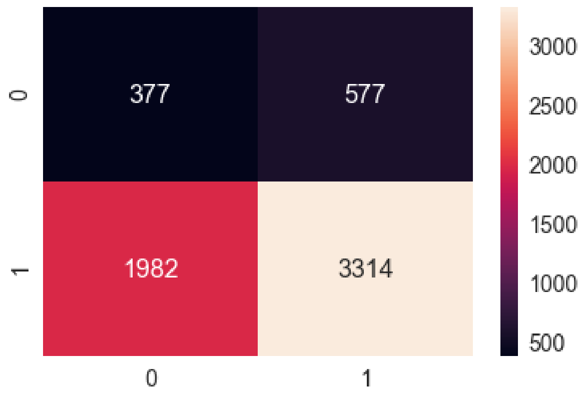
In [95]:

```
confusion_matr_tfidf_w2v_test = pd.DataFrame(confusion_matrix(Y_test, predict(y_test_pred, tr_thres
holds, test_fpr, test_fpr)), range(2),range(2))

sns.set(font_scale=1.4)
sns.heatmap(confusion_matr_tfidf_w2v_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24996044460266603 for threshold 0.845
```

Out[95]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x26560ef0>
```



2.5 Feature selection with `SelectKBest`

In [96]:

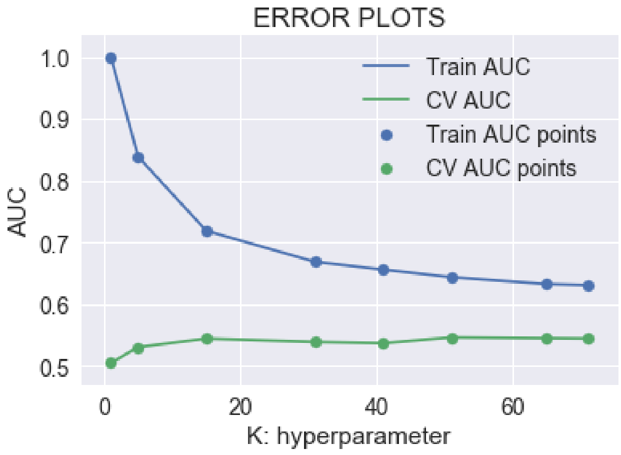
```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label

# Merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039

from scipy.sparse import hstack
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

X_train_merge = hstack((categories_one_hot_train, sub_cat_one_hot_train,
school_state_one_hot_train, project_grade_cat_one_hot_train, teacher_prefix_cat_one_hot_train, pri
ce_data_train, quant_train, prev_no_projects_train,title_tfidf_train, essay_tfidf_train)).tocsr()
X_test_merge = hstack((categories_one_hot_test, sub_cat_one_hot_test, school_state_one_hot_test, p
roject_grade_cat_one_hot_test, teacher_prefix_cat_one_hot_test, price_data_test, quant_test,
prev_no_projects_test,title_tfidf_test, essay_tfidf_test)).tocsr()
X_cv_merge = hstack((categories_one_hot_cv, sub_cat_one_hot_cv,
school_state_one_hot_cv,project_grade_cat_one_hot_cv, teacher_prefix_cat_one_hot_cv, price_data_cv
, quant_cv, prev_no_projects_cv,title_tfidf_cv, essay_tfidf_cv)).tocsr()

##Select top 2000 features from feature Set 2 using `SelectKBest` and then apply KNN on top of the
se features

select_func = SelectKBest(chi2, k=2000).fit(X_train_merge, Y_train)


X_train_2000 = select_func.transform(X_train_merge)
X_test_2000 = select_func.transform(X_test_merge)
X_cv_2000 = select_func.transform(X_cv_merge)


print("Final Data matrix")
print("="*100)
print(X_train_2000.shape, Y_train.shape)
print(X_cv_2000.shape, Y_test.shape)
print(X_test_2000.shape, Y_cv.shape)
print("="*100)
```

```
Final Data matrix
====================================================================================================

(14062, 2000) (14062,)
(4688, 2000) (6250,)
(6250, 2000) (4688,)
====================================================================================================
```

In [97]:

```
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 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 = []

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

for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train_2000, Y_train)

    y_train_pred = batch_predict(neigh, X_train_2000)
    y_cv_pred = batch_predict(neigh, X_cv_2000)

    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("ERROR PLOTS")
plt.grid(True)
plt.show()
```

```
100%|████████████████████████████████████████| 8/8 [03:49<00:00, 29.04s/it]
```



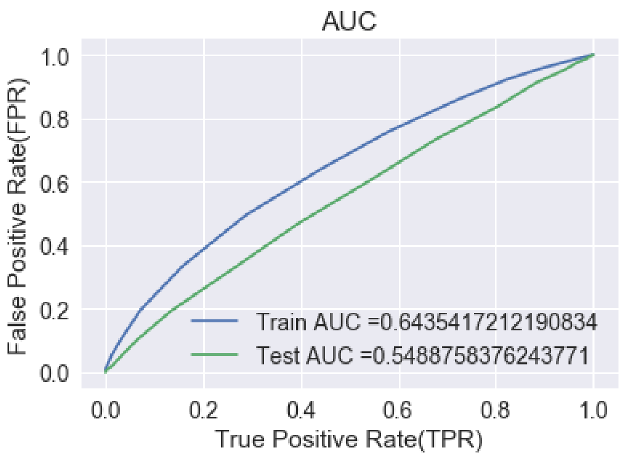
In [98]:

```
scor = [x for x in cv_auc]
opt_t_cv_4 = K[scor.index(max(scor))]
print("Maximum AUC score of cv is:" + ' ' + str(max(scor)))
print("Corresponding k value of cv is:",opt_t_cv_4, '\n')

best_k=opt_t_cv_4
```

```
Maximum AUC score of cv is: 0.5461187460266559
Corresponding k value of cv is: 51
```

In [99]:

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

neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X_train_2000, 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_2000)
y_test_pred = batch_predict(neigh, X_test_2000)

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.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(True)
plt.show()
```



Confusion Matrix

In [100]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(Y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(Y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
print("="*100)
```

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2463303278211528 for threshold 0.843
[[1203  943]
 [4315 7601]]
====================================================================================================

Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2477750088999644 for threshold 0.824
[[ 307  647]
 [1407 3889]]
====================================================================================================
```

Confusion Matrix heatmap - Train

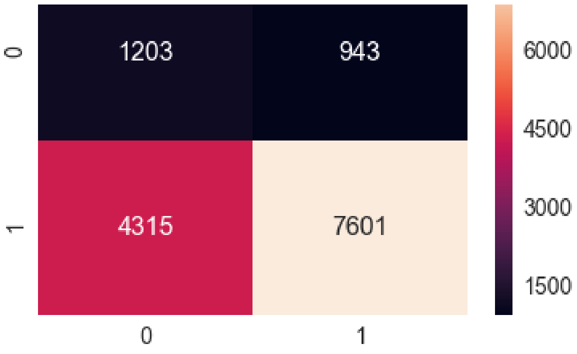
In [101]:

```
confusion_mat_2000_train = pd.DataFrame(confusion_matrix(Y_train, predict(y_train_pred,
tr_thresholds, train_fpr, train_fpr)), range(2),range(2))
sns.set(font_scale=1.4)
sns.heatmap(confusion_mat_2000_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.2463303278211528 for threshold 0.843
```

Out[101]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x14930cf8>
```



Confusion matrix Heat Map - TEST

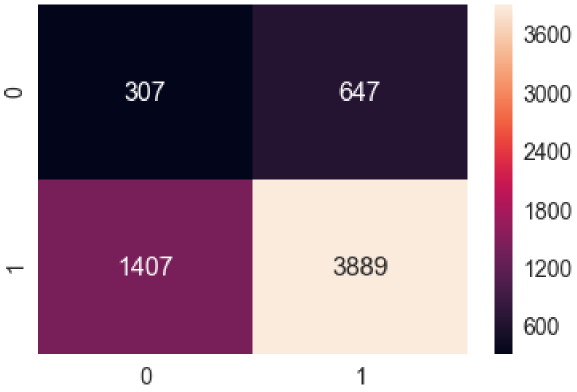
In [102]:

```
confusion_mat_2000_test = pd.DataFrame(confusion_matrix(Y_test, predict(y_test_pred, tr_thresholds,
test_fpr, test_fpr)), range(2),range(2))
sns.set(font_scale=1.4)
sns.heatmap(confusion_mat_2000_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.2477750088999644 for threshold 0.824
```

Out[102]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1416f9b0>
```



3. Conclusions

In [103]:

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

from prettytable import PrettyTable

x_pretty_table = PrettyTable()
x_pretty_table.field_names = ["Model Type","Vectorizer", "Hyper Parameter - K","Train-AUC","Test-AU
C"]

x_pretty_table.add_row(["KNN-Brute","BOW",65,0.67,0.62])
x_pretty_table.add_row([ "KNN-Brute", "TFIDF",41,0.66, 0.56])
x_pretty_table.add_row([ "KNN-Brute","AVG W2V", 60,0.66, 0.59])
x_pretty_table.add_row([ "KNN-Brute", "TFIDF W2V",71,0.61,0.55])
x_pretty_table.add_row([ "Top 2000 Features","TFIDF",71,0.67,0.55])

print(x_pretty_table)
```

```
+--------------------+------------+---------------------+-----------+----------+
|     Model Type     | Vectorizer | Hyper Parameter - K | Train-AUC | Test-AUC |
+--------------------+------------+---------------------+-----------+----------+
|     KNN-Brute      |    BOW     |          65         |    0.67   |   0.62   |
|     KNN-Brute      |   TFIDF    |          41         |    0.66   |   0.56   |
|     KNN-Brute     |   TFIDF    |          41         |    0.66   |   0.56   |
|     KNN-Brute     |  AVG W2V   |          60         |    0.66   |   0.59   |
|     KNN-Brute     | TFIDF W2V  |          71         |    0.61   |   0.55   |
| Top 2000 Features |   TFIDF    |          71         |    0.67   |   0.55   |
+-------------------+------------+---------------------+-----------+----------+
```

As asked to defferentiate betwenn fit (), fit_transform() and transform () in previous suggestion .

- fit () - lears the dictionary internally
- transform() - It applies the learned vocalbulary to give the output , (BOW in this case) or document-term
- fit_term () - combination of fit and transform() in one go