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`

Feature	Description
	• `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*
`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 [1]:

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

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

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

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

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

from tqdm import tqdm
import os

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

In [2]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%
b&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.
2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fww
ogleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:

```
path_train="/content/drive/My Drive/Colab Notebooks/train_new_data.csv"
path_resource="/content/drive/My Drive/Assignments_DonorsChoose_2018/resources.csv"
```

1.1 Reading Data

In [2]:

```
project_data = pd.read_csv("train_new_data.csv")
resource_data = pd.read_csv("resources.csv")
```

In [3]:

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

```
Number of data points in train data (109248, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [4]:

```
# 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[4]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_cat
3461	5749	p096076	6eaa448903897a152320bd23a30147b2	Mrs.	CA	2016-01-05 00:00:00	Grades PreK-2
58659	98044	p001225	9568c8968f974c1fc34def91394cb005	Ms.	CA	2016-01-05 01:05:00	Grades PreK-2

In [5]:

```
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[5]:

	id	description	quantity	price
0	p233245	LC652 - Lakeshore Double-Space Mobile Drying Rack	1	149.00

	id	description	quantity	price
1	p069063	Bouncy Bands for Desks (Blue support pipes)	3	14.95

1.2 preprocessing of `project_subject_categories`

In [6]:

```
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 [7]:

```
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
```

```
# 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]))
```

preprocessing school state

In [8]:

```
from collections import Counter
my_counter = Counter()
for word in project_data['school_state'].values:
    my_counter.update(word.split())
state_dict = dict(my_counter)
sorted_state_dict = dict(sorted(state_dict.items(), key=lambda kv: kv[1]))
```

preprocessing teacher prefix

In [9]:

```
from collections import Counter
my_counter = Counter()
for word in project_data['teacher_prefix'].values:
    my_counter.update(word.split())
prefix_dict = dict(my_counter)
sorted_prefix_dict = dict(sorted(prefix_dict.items(), key=lambda kv: kv[1]))
```

preprocessing project grade category

In [10]:

```
catogories = list(project_data['project_grade_category'].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
pgc_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
    pgc_list.append(temp.strip())

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

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

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

1.3 Text preprocessing

In [11]:

```
# 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 [12]:

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

Out[12]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_title	proje
3461	5749	p096076	6eaa448903897a152320bd23a30147b2	Mrs.	CA	2016-01-05 00:00:00	Math Madness	A typ our c full o
58659	98044	p001225	9568c8968f974c1fc34def91394cb005	Ms.	CA	2016-01-05 01:05:00	Animal Adaptation Study with an Ant Farm!	The a marc two t a...

splitting data into train CV and test

In [13]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(project_data,
project_data['project_is_approved'], test_size=0.33, stratify = project_data['project_is_approved'
])
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

In [14]:

```
X_train.drop(['project_is_approved'], axis=1, inplace=True)
X_test.drop(['project_is_approved'], axis=1, inplace=True)
X_cv.drop(['project_is_approved'], axis=1, inplace=True)
```

In [15]:

```
# printing some random reviews
print(X_train['essay'].values[0])
print("="*50)
print(X_test['essay'].values[150])
print("="*50)
print(X_cv['essay'].values[1000])
print("="*50)
```

```
My students are actively becoming  life long readers through their love of books. I want to encour
age this love of reading by offering my students many different ways to succeed in reading..
\r\n\r\nOur school is filled with eager readers from a variety of  backgrounds. Robert Frost is a
Title 1 elementary school with almost 450 students. Our students come from a variety of different
economic backgrounds with over 40% of our students receiving free or reduced lunch. Our library is
one in which the students visit for not only books and learning, but to socialize and create. Our
students visit the library for a variety of reasons and I want to continue to serve them.\r\nPart
of the library curriculum is learning how to use the 5 Star Research Process.  I use topics studie
d in the classroom to teach this process.  Third graders study other countries and their cultures
in Social Studies and we continue this work in the library using the 5 Star Research
Process.\r\n\r\nUnfortunately, the country  select in our library collection is out of date.  The
```

```
average copyright date for that section is 1998.  My students need  primary sources that are
current, at their reading level, and reflect their own various cultures.  These books will help my
students find relevant information and give them the research practice they need.nannan
==================================================
District 20 Pre-K Centers boast a group of special students of many different ethnicities and
religious backgrounds.  Our sites promote unity and collaboration in a neighborhood that may be la
cking.  Our neighborhood of Bath Beach Brooklyn has a lot of heart and history, and the community
members play a big role in the learning process.  Our schools offers a vibrant early childhood cul
ture with a sole focus on the Pre-K experience. We pride ourselves on individualized attention and
advocacy of the administrators, teachers, paraprofessionals and support staff.  The climate of our
school is respect and trust. Each child is valued as an individual while all benefit from social-e
motional support. Children are part of a collaborative school community that actively encourages p
arent involvement, community engagement and strong family ties. We are committed to high academic
standards from skilled and experienced educators certified in early childhood instruction. Student
readiness for kindergarten is achieved via an ambitious curriculum. This is aimed at developing cr
itical thinking skills and academic growth across the disciplines.\r\n\r\nI am looking to create a
SHARE LIBRARY for a school population ranging from Chinese, Spanish and Arabic, most of which are
low-income and are children who are learning a new language.  \r\nThrough my initiative, I am look
ing to stress the importance of literacy for even our youngest of learners.  I also plan on provid
ing families and caregivers with workshops on the importance of literacy at home.  I know that if
you read about what District 20 pre-k centers were about, you would fall in love.  We have just
launched a social media campaign, and we can be found on Twitter, Instagram and Facebook.  Back in
November our site hosted a cultural event that created such a stir in the neighborhood.  Each clas
sroom represented a region and demonstrated their region by a dance, song or activity.  My
classroom received India, and we performed a Bollywood dance that was adorable! The children learn
ed so much. I even had a museum curator from the Tibetan Museum in Staten Island visit.  Families
were in awe!  We also did our first annual March for Peace in honor of Martin Luther King, Jr
earlier this month!  The school has over 200 students, and I am looking to do as much as possible
to provide these underprivileged students with reading material that they can be proud to take hom
e. I am also looking to teach them responsibility by returning their book on time.nannan
==================================================
The students in my class are 5 boys ages 16-18. They each have autism.  Every day we come to
school to work on academics, daily living skills, and vocational skills.  Every day brings new
surprises and new challenges. \r\n \r\nEvery day my students are challenged to be as independent a
s possible.  They are challenged to make choices, advocate for themselves, and meet their
individual goals.  These goals look different for every student, and every student needs different
levels of support to meet those goals.My students interact with technology everyday.  They use it
learn new concepts, interact with text, participate in a yoga program, and research the world outs
ide of our classroom.  Unfortunately, the laptop we were using to pair with our projector is no lo
nger working, and that limits a lot of what we can do in our classroom.  With the help of a new la
ptop we can project books, videos, and large pictures on the wall to allow my students to fully en
gage and participate in every lesson.  \r\n\r\nHaving a laptop will allow my students to read and
view books and adapted texts as a group.  Doing so would allow them to communicate with each other
about what they are reading and seeing while practicing their social skills.nannan
==================================================
```

Decontracting function for sentence

In [16]:

```
# 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 [17]:

```
# 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 [18]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays_train = []
# tqdm is for printing the status bar
for sentance in tqdm(X_train['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_train.append(sent.lower().strip())
```

```
100%|██████████| 49041/49041 [00:52<00:00, 941.67it/s]
```

In [19]:

```
# after preprocesing
preprocessed_essays_train[2000]
```

Out[19]:

```
'teach extremely energetic middle schoolers small town pontotoc mississippi students work hard bri
ng many creative ideas classroom day middle school comprised diverse group students 70 low income
homes fifty percent students african american hispanic including many english language learners st
udents rated one lowest performing groups according mississippi state assessment system however st
udents ready take charge education set goals succeed project assist students becoming 21st century
learners use technology classroom would like provide students chromebooks classroom research compl
ete projects may otherwise not opportunity economic situation home chromebooks would allow student
s explore express technology veteran teacher amazed eagerness research topics connect visuals
topics studied although school classroom set ipads not always available need research chromebooks
classroom enhance student learning allow immediate discovery advanced knowledge students
technology access chromebooks would not allow discovery students also although hesitant pull techn
ology know technology essential student engagement research authentic writing therefore growing ef
fectiveness instructional technology not come great surprise thank advance improving abilities
21st century teacher assist 21st century students nannan'
```

1.4 Preprocessing of `project_title`

In [20]:

```
# similarly you can preprocess the titles also
from tqdm import tqdm
preprocessed_project_titles_train = []
# tqdm is for printing the status bar
for sentence in tqdm(X_train['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_project_titles_train.append(sent.lower().strip())
```

```
100%|██████████| 49041/49041 [00:02<00:00, 18411.71it/s]
```

Preprocessing Test datapoints

In [21]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays_test = []
# tqdm is for printing the status bar
for sentance in tqdm(X_test['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_test.append(sent.lower().strip())
```

```
100%|██████████| 36052/36052 [00:35<00:00, 1010.09it/s]
```

In [22]:

```
preprocessed_essays_test[2000]
```

Out[22]:

```
'students love working together creating pieces work via technology need way use video audio ease
create video taped masterpieces collaborative projects also need access latest engaging
educational applications reinforcement skills teach dynamic dual immersion program marvelous
students becoming literate two languages school program improvement fourth year row title 1 2nd gr
ade students critical age learn strong foundational skills reading writing math learn year effect
rest educational experience failure not option school technology oriented strong emphasis providin
g students access multiple forms media technology however school school district often lack funds
offer students modern technology need order flourish 21st century workplace grants one help meet g
oal putting latest technology students eager hands five ipad minis used classroom daily provide
students opportunity research topics taught class students able prepare videotape oral language sp
eeches well collaborative projects students also able work video creations across content areas us
ing various apps educreations story wheel plethora reading literary math science apps available ip
ad mini also many coding programming creativity apps help children develop creative critical think
ing skills possibilities simply endless today student technology not luxury rather technology
necessity please help students feel technologically empowered gifting ipad minis classroom order s
tudents soar successful future must technological skills project give students chance work
collaboratively expand learning subject areas current technology tools donations classroom project
immediately benefit students would please donate ipad minis current future students thank generous
support today children future leaders country nannan'
```

In [23]:

```
from tqdm import tqdm
preprocessed_project_titles_test = []
# tqdm is for printing the status bar
for sentence in tqdm(X_test['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_project_titles_test.append(sent.lower().strip())
```

```
100%|██████████| 36052/36052 [00:01<00:00, 22099.67it/s]
```

Preprocessing cross validation points

In [24]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays_cv = []
# tqdm is for printing the status bar
for sentance in tqdm(X_cv['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_cv.append(sent.lower().strip())
```

```
100%|██████████| 24155/24155 [00:24<00:00, 1002.86it/s]
```

In [25]:

```
preprocessed_essays_cv[2000]
```

Out[25]:

```
'students eager challenge motivated grow readers writers using science integrative tool learning b
ecomes connected meaningful students struggle attend information group setting excited interact hi
gh interest materials project provide class tools need create dynamic engaging atmosphere students
build upon curiosity chose materials based think students find inspiring next generation science s
tandards students able light classroom explorations light sound waves high quality materials last
years impact lives many students using play hands activities students motivated extend learning sc
ience writing building social emotional skills example playing shadow puppets social interactive e
ngaging science imagination together help students grow curiosity reinforce social skills collabor
atively taught school contribution project help students practice scientific method engineering de
sign process materials give science activities pop deserve nannan'
```

In [26]:

```
from tqdm import tqdm
preprocessed_project_titles_cv = []
# tqdm is for printing the status bar
for sentence in tqdm(X_cv['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_project_titles_cv.append(sent.lower().strip())
```

```
100%|██████████| 24155/24155 [00:01<00:00, 17238.48it/s]
```

1.5 Preparing data for models

In [27]:

```
project_data.columns
```

Out[27]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', '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', 'clean_pgc', 'essay'],
      dtype='object')
```

we are going to consider

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

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

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

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

One_hot_encoding of Clean categories

In [28]:

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

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix of Train data after one hot encoding  (49041, 9)
Shape of matrix of Test data after one hot encoding  (36052, 9)
Shape of matrix of CV data after one hot encoding  (24155, 9)
```

One_hot_encoding of Clean subcategories

In [31]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=
True)
vectorizer.fit(X_train['clean_subcategories'].values)
sub_categories_one_hot_train = vectorizer.transform(X_train['clean_subcategories'].values)
sub_categories_one_hot_test = vectorizer.transform(X_test['clean_subcategories'].values)
sub_categories_one_hot_cv = vectorizer.transform(X_cv['clean_subcategories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix of Train data after one hot encoding ",sub_categories_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",sub_categories_one_hot_test.shape)
print("Shape of matrix of Cross Validation data after one hot encoding ",sub_categories_one_hot_cv
.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
```

```
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix of Train data after one hot encoding  (49041, 30)
Shape of matrix of Test data after one hot encoding  (36052, 30)
Shape of matrix of Cross Validation data after one hot encoding  (24155, 30)
```

One_hot_encoding of school state

In [32]:

```
# you can do the similar thing with state, teacher_prefix and project_grade_category also

vectorizer = CountVectorizer(vocabulary=list(sorted_state_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(vectorizer.get_feature_names())
print("Shape of matrix of Train data after one hot encoding ",school_state_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",school_state_one_hot_test.shape)
print("Shape of matrix of Cross Validation data after one hot encoding ",school_state_one_hot_cv
.shape)
```

```
['VT', 'WY', 'ND', 'MT', 'RI', 'SD', 'NE', 'DE', 'AK', 'NH', 'WV', 'ME', 'HI', 'DC', 'NM', 'KS', 'I
A', 'ID', 'AR', 'CO', 'MN', 'OR', 'KY', 'MS', 'NV', 'MD', 'CT', 'TN', 'UT', 'AL', 'WI', 'VA', 'AZ',
'NJ', 'OK', 'WA', 'MA', 'LA', 'OH', 'MO', 'IN', 'PA', 'MI', 'SC', 'GA', 'IL', 'NC', 'FL', 'NY', 'TX
', 'CA']
Shape of matrix of Train data after one hot encoding  (49041, 51)
Shape of matrix of Test data after one hot encoding  (36052, 51)
Shape of matrix of Cross Validation data after one hot encoding  (24155, 51)
```

One_hot_encoding of Teacher prefix

In [33]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_prefix_dict.keys()), lowercase=False, binary=
True)
vectorizer.fit(X_train['teacher_prefix'].values)
teacher_prefix_one_hot_train = vectorizer.transform(X_train['teacher_prefix'].values)
teacher_prefix_one_hot_test = vectorizer.transform(X_test['teacher_prefix'].values)
teacher_prefix_one_hot_cv = vectorizer.transform(X_cv['teacher_prefix'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix of Train data after one hot encoding ",teacher_prefix_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",teacher_prefix_one_hot_test.shape)
print("Shape of matrix of Cross Validation data after one hot encoding ",teacher_prefix_one_hot_cv
.shape)
```

```
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
Shape of matrix of Train data after one hot encoding  (49041, 5)
Shape of matrix of Test data after one hot encoding  (36052, 5)
Shape of matrix of Cross Validation data after one hot encoding  (24155, 5)
```

One_hot_encoding of clean project grade categories

In [34]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_pgc_dict.keys()), lowercase=False, binary=
True)
vectorizer.fit(X_train['clean_pgc'].values)
clean_project_grade_category_one_hot_train = vectorizer.transform(X_train['clean_pgc'].values)
clean_project_grade_category_one_hot_test = vectorizer.transform(X_test['clean_pgc'].values)
clean_project_grade_category_one_hot_cv = vectorizer.transform(X_cv['clean_pgc'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix of Train data after one hot encoding
",clean_project_grade_category_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding
",clean_project_grade_category_one_hot_test.shape)
print("Shape of matrix of Cross Validation data after one hot encoding
",clean_project_grade_category_one_hot_cv
.shape)
```

```
['Grades9-12', 'Grades6-8', 'Grades3-5', 'GradesPreK-2']
Shape of matrix of Train data after one hot encoding  (49041, 4)
Shape of matrix of Test data after one hot encoding  (36052, 4)
Shape of matrix of Cross Validation data after one hot encoding  (24155, 4)
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

Vectorizing essay train dataset

In [35]:

```
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(preprocessed_essays_train)
text_bow_train = vectorizer.transform(preprocessed_essays_train)
print("Shape of matrix after one hot encoding ",text_bow_train.shape)
```

```
Shape of matrix after one hot encoding  (49041, 12020)
```

Vectorizing essay test dataset

In [36]:

```
text_bow_test = vectorizer.transform(preprocessed_essays_test)
print("Shape of matrix after one hot encoding ",text_bow_test.shape)
```

```
Shape of matrix after one hot encoding  (36052, 12020)
```

Vectorizing essay cross validation dataset

In [37]:

```
text_bow_cv = vectorizer.transform(preprocessed_essays_cv)
print("Shape of matrix after one hot encoding ",text_bow_cv.shape)
```

```
Shape of matrix after one hot encoding  (24155, 12020)
```

Vectorizing project titles train dataset

In [38]:

```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
vectorizer.fit(preprocessed_project_titles_train)
title_bow_train = vectorizer.transform(preprocessed_project_titles_train)
print("Shape of matrix after one hot encoding ",title_bow_train.shape)
```

```
Shape of matrix after one hot encoding  (49041, 2092)
```

Vectorizing project titles test dataset

In [39]:

```
title_bow_test = vectorizer.transform(preprocessed_project_titles_test)
print("Shape of matrix after one hot encoding ",title_bow_test.shape)
```

```
Shape of matrix after one hot encoding  (36052, 2092)
```

Vectorizing project titles cross validation dataset

In [40]:

```
title_bow_cv = vectorizer.transform(preprocessed_project_titles_cv)
print("Shape of matrix after one hot encoding ",title_bow_cv.shape)
```

```
Shape of matrix after one hot encoding  (24155, 2092)
```

1.5.2.2 TFIDF vectorizer

TFIDF vectorization for essay train data

In [41]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(preprocessed_essays_train)
text_tfidf_train = vectorizer.transform(preprocessed_essays_train)
print("Shape of matrix after one hot encoding ",text_tfidf_train.shape)
```

```
Shape of matrix after one hot encoding  (49041, 12020)
```

TFIDF vectorization for essay test data

In [42]:

```
text_tfidf_test = vectorizer.transform(preprocessed_essays_test)
print("Shape of matrix after one hot encoding ",text_tfidf_test.shape)
```

```
Shape of matrix after one hot encoding  (36052, 12020)
```

TFIDF vectorization for essay cross validation data

In [43]:

```
text_tfidf_cv = vectorizer.transform(preprocessed_essays_cv)
print("Shape of matrix after one hot encoding ",text_tfidf_cv.shape)
```

```
Shape of matrix after one hot encoding  (24155, 12020)
```

TFIDF vectorization for project title train data

In [44]:

```
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(preprocessed_project_titles_train)
title_tfidf_train = vectorizer.transform(preprocessed_project_titles_train)
print("Shape of matrix after one hot encoding ",title_tfidf_train.shape)
```

```
Shape of matrix after one hot encoding  (49041, 2092)
```

TFIDF vectorization for project title test data

In [45]:

```
title_tfidf_test = vectorizer.transform(preprocessed_project_titles_test)
print("Shape of matrix after one hot encoding ",title_tfidf_test.shape)
```

```
Shape of matrix after one hot encoding  (36052, 2092)
```

TFIDF vectorization for project title cross validation data

In [46]:

```
title_tfidf_cv = vectorizer.transform(preprocessed_project_titles_cv)
print("Shape of matrix after one hot encoding ",title_tfidf_cv.shape)
```

```
Shape of matrix after one hot encoding  (24155, 2092)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [47]:

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

```
Loading Glove Model
```

```
279727it [02:00, 2313.69it/s]
```

```
Done. 279727  words loaded!
```

In [48]:

```
words_essays_train = []
for i in preprocessed_essays_train :
    words_essays_train.extend(i.split(' '))
print("number of the words in the corpus", len(words_essays_train))
```

```
number of the words in the corpus 6778140
```

In [49]:

```
words_essay_train = set(words_essays_train)
print("the unique words in the corpus", len(words_essay_train))
```

```
the unique words in the corpus 41133
```

In [50]:

```
inter_words = set(model.keys()).intersection(words_essay_train)
print("The number of words that are present in both glove vectors and our corpus are {} which \
is nearly {}% ".format(len(inter_words), np.round((float(len(inter_words))/len(words_essay_train))
*100)))
```

```
The number of words that are present in both glove vectors and our corpus are 34433 which is
nearly 84.0%
```

In [51]:

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

```
word 2 vec length 34433
```

In [52]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/
import pickle
with open('glove_vectors', 'wb') as f:
    pickle.dump(words_corpus_train_essay, f)
```

In [53]:

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

Average word2vec for preprocessed essays-Train

In [54]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_train): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train.append(vector)

print(len(avg_w2v_vectors_train))
print(len(avg_w2v_vectors_train[0]))
```

```
100%|██████████| 49041/49041 [00:20<00:00, 2347.97it/s]
```

```
49041
300
```

Average word2vec for preprocessed essays-test

In [55]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_test): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test.append(vector)

print(len(avg_w2v_vectors_test))
print(len(avg_w2v_vectors_test[0]))
```

```
100%|██████████| 36052/36052 [00:18<00:00, 1997.22it/s]
```

```
36052
300
```

Average word2vec for preprocessed essays-C.V

In [56]:

```
avg_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_cv): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_cv.append(vector)

print(len(avg_w2v_vectors_cv))
print(len(avg_w2v_vectors_cv[0]))
```

```
100%|██████████| 24155/24155 [00:32<00:00, 753.89it/s]
```

```
24155
300
```

Avg word2vec project title-Train

In [57]:

```
avg_w2v_ppt_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_project_titles_train): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_ppt_train.append(vector)

print(len(avg_w2v_ppt_train))
print(len(avg_w2v_ppt_train[0]))
```

```
100%|██████████| 49041/49041 [00:01<00:00, 47077.83it/s]
```

```
49041
300
```

Avg word2vec project title-test

In [58]:

```
avg_w2v_ppt_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_project_titles_test): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_ppt_test.append(vector)

print(len(avg_w2v_ppt_test))
print(len(avg_w2v_ppt_test[0]))
```

```
100%|██████████| 36052/36052 [00:00<00:00, 39481.58it/s]
```

```
36052
300
```

Avg word2vec project title-cross vaidation

In [59]:

```
avg_w2v_ppt_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_project_titles_cv): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_ppt_cv.append(vector)

print(len(avg_w2v_ppt_cv))
print(len(avg_w2v_ppt_cv[0]))
```

```
100%|██████████| 24155/24155 [00:01<00:00, 15240.43it/s]
```

```
24155
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [60]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_essays_train)
# 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())
tfidf_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_train): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_train.append(vector)

print(len(tfidf_w2v_vectors_train))
print(len(tfidf_w2v_vectors_train[0]))
```

```
100%|██████████| 49041/49041 [03:26<00:00, 237.55it/s]
```

```
49041
300
```

In [61]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_essays_test)
```

```
# 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())
tfidf_w2v_vectors_test= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_test): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_test.append(vector)

print(len(tfidf_w2v_vectors_test))
print(len(tfidf_w2v_vectors_test[0]))
```

```
100%|██████████| 36052/36052 [02:02<00:00, 294.83it/s]
```

```
36052
300
```

In [62]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_essays_cv)
# 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())
tfidf_w2v_vectors_cv= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_cv): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_cv.append(vector)

print(len(tfidf_w2v_vectors_cv))
print(len(tfidf_w2v_vectors_cv[0]))
```

```
100%|██████████| 24155/24155 [01:20<00:00, 301.28it/s]
```

```
24155
300
```

In [63]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_project_titles_train)
# 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())
tfidf_w2v_ppt_train= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_project_titles_train): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_ppt_train.append(vector)

print(len(tfidf_w2v_ppt_train))
print(len(tfidf_w2v_ppt_train[0]))
```

```
100%|██████████| 49041/49041 [00:02<00:00, 20434.79it/s]
```

```
49041
300
```

In [64]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_project_titles_test)
# 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())
tfidf_w2v_ppt_test= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_project_titles_test): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_ppt_test.append(vector)

print(len(tfidf_w2v_ppt_test))
print(len(tfidf_w2v_ppt_test[0]))
```

```
100%|██████████| 36052/36052 [00:01<00:00, 19207.36it/s]
```

```
36052
300
```

In [65]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_project_titles_cv)
# 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())
tfidf_w2v_ppt_cv= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_project_titles_cv): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_ppt_cv.append(vector)

print(len(tfidf_w2v_ppt_cv))
print(len(tfidf_w2v_ppt_cv[0]))
```

```
100%|██████████| 24155/24155 [00:01<00:00, 21300.42it/s]
```

```
24155
300
```

1.5.3 Vectorizing Numerical features

In [66]:

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

In [67]:

```
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')
```

standardizing teacher number of price

In [68]:

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

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.
73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)

price_scalar = Normalizer()
price_scalar.fit(X_train['price'].values.reshape(-1,1)) # finding the mean and standard deviation
of this data
price_standardized_train = price_scalar.transform(X_train['price'].values.reshape(-1, 1))
price_standardized_test = price_scalar.transform(X_test['price'].values.reshape(-1, 1))
price_standardized_cv = price_scalar.transform(X_cv['price'].values.reshape(-1, 1))
```

In [69]:

```
print(price_standardized_train.shape)
print(price_standardized_test.shape)
print(price_standardized_cv.shape)
```

```
(49041, 1)
(36052, 1)
```

```
(24155, 1)
```

standardizing teacher number of previously posted projects

In [70]:

```
price_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
prev_project_standardized_train =
price_scalar.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,
1))
prev_project_standardized_test =
price_scalar.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1)
)
prev_project_standardized_cv =
price_scalar.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))
```

In [71]:

```
print(prev_project_standardized_train.shape)
print(prev_project_standardized_test.shape)
print(prev_project_standardized_cv.shape)
```

```
(49041, 1)
(36052, 1)
(24155, 1)
```

1.5.4 Merging all the above features

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

In [72]:

```
# 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 :)
S_BOW_train=
hstack((categories_one_hot_train,sub_categories_one_hot_train,school_state_one_hot_train,teacher_pr
efix_one_hot_train,clean_project_grade_category_one_hot_train,text_bow_train,title_bow_train,price_
standardized_train,prev_project_standardized_train)).tocsr()
S_BOW_train.shape
```

Out[72]:

```
(49041, 14213)
```

In [73]:

```
S_BOW_test= hstack((categories_one_hot_test,sub_categories_one_hot_test,school_state_one_hot_test,
teacher_prefix_one_hot_test,clean_project_grade_category_one_hot_test,text_bow_test,title_bow_test
,price_standardized_test,prev_project_standardized_test)).tocsr()
S_BOW_test.shape
```

Out[73]:

```
(36052, 14213)
```

In [74]:

```
S_BOW_cv=
hstack((categories_one_hot_cv,sub_categories_one_hot_cv,school_state_one_hot_cv,teacher_prefix_one_
hot_cv,clean_project_grade_category_one_hot_cv,text_bow_cv,title_bow_cv,price_standardized_cv,prev_
project_standardized_cv)).tocsr()
S_BOW_cv.shape
```

Out[74]:

```
(24155, 14213)
```

In [75]:

```
print("BOW with other features Data matrix")
print(S_BOW_train.shape, y_train.shape)
print(S_BOW_cv.shape, y_cv.shape)
print(S_BOW_test.shape, y_test.shape)
print("*"*50)
```

```
BOW with other features Data matrix
(49041, 14213) (49041,)
(24155, 14213) (24155,)
(36052, 14213) (36052,)
**************************************************
```

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
       ```

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

function for predicting in batch

In [76]:

```
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 tr_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
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

finding best K hyper parameter using AUC

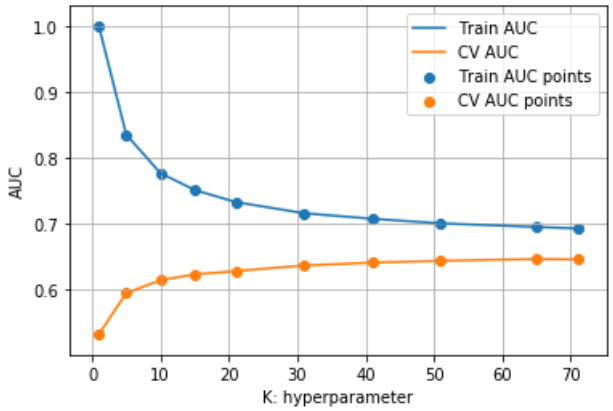
In [78]:

```
from sklearn.neighbors import KNeighborsClassifier
```

In [52]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
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(S_BOW_train, y_train)
    y_train_pred = batch_predict(neigh,S_BOW_train)
    y_cv_pred = batch_predict(neigh, S_BOW_cv)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
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 [1:05:35<00:00, 394.56s/it]
```



best_k=15

Using Randomized searchcv for finding best hyper parameter for BOW

In [1]:

```
from scipy.stats import uniform
from sklearn.model_selection import RandomizedSearchCV
```

In [59]:

```
neigh = KNeighborsClassifier()
k_range=np.arange(1,100)
weights=["uniform","distance"]
parameters=dict(n_neighbors=k_range,weights=weights)
clf = RandomizedSearchCV(neigh, parameters, cv=5, scoring='roc_auc')
best_model=clf.fit(S_BOW_train, y_train)
print('Best C:', best_model.best_estimator_.get_params()['n_neighbors'])
```

```
Best C: 80
```

In [60]:

```
best_k=79
print(best_k)
```

```
79
```

we will use k=15

2.4.1 Applying KNN brute force on BOW, SET 1

In [79]:

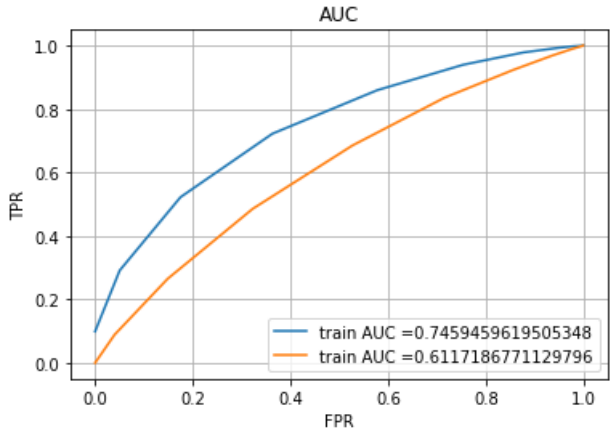
```
# Please write all the code with proper documentation
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
neigh = KNeighborsClassifier(n_neighbors=15, n_jobs=-1)
neigh.fit(S_BOW_train, y_train)
# 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, S_BOW_train)
y_test_pred = batch_predict(neigh, S_BOW_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="train AUC ="+str(auc(test_fpr, test_tpr)))
```

```
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("AUC")
plt.grid()
plt.show()
```



In [80]:

```
def prediction(proba, threshould, fpr, tpr):
    t = threshould[np.argmax(fpr*(1-tpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

Confusion matrix for Train data-BOW

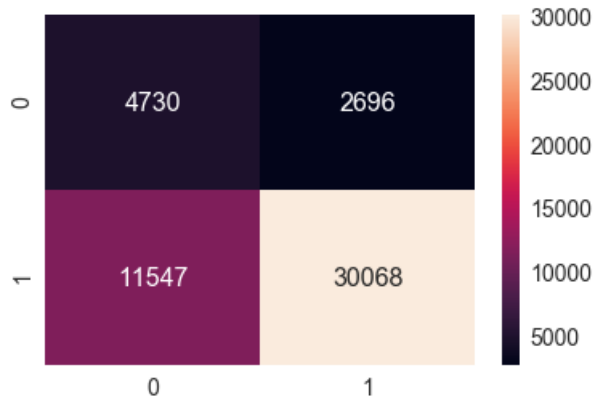
In [81]:

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

```
the maximum value of tpr*(1-fpr) 0.46021507283089746 for threshold 0.8
```

Out[81]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf967abe10>
```



confusion matrix for test data-BOW

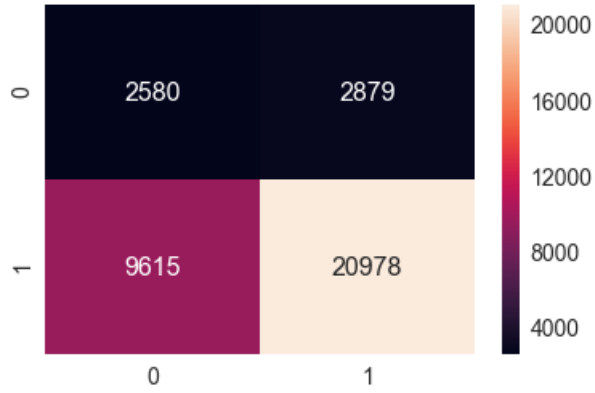
In [83]:

```
conf_matr_df_test = pd.DataFrame(confusion_matrix(y_test, prediction(y_test_pred, tr_thresholds,
train_fpr, train_tpr)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.46021507283089746 for threshold 0.8
```

Out[83]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf9cb80048>
```



2.4.2 Applying KNN brute force on TFIDF, SET 2

In [84]:

```
# Please write all the code with proper documentation
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
S_TFIDF_train=
hstack((categories_one_hot_train,sub_categories_one_hot_train,school_state_one_hot_train,teacher_pr
efix_one_hot_train,clean_project_grade_category_one_hot_train,text_tfidf_train,title_tfidf_train,p
rice_standardized_train,prev_project_standardized_train)).tocsr()
S_TFIDF_train.shape
```

Out[84]:

```
(49041, 14213)
```

In [85]:

```
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
S_TFIDF_test=
hstack((categories_one_hot_test,sub_categories_one_hot_test,school_state_one_hot_test,teacher_prefi
x_one_hot_test,clean_project_grade_category_one_hot_test,text_tfidf_test,title_tfidf_test,price_sta
ndardized_test,prev_project_standardized_test)).tocsr()
S_TFIDF_test.shape
```

Out[85]:

```
(36052, 14213)
```

In [86]:

```
S_TFIDF_cv=
hstack((categories_one_hot_cv,sub_categories_one_hot_cv,school_state_one_hot_cv,teacher_prefix_one_
hot_cv,clean_project_grade_category_one_hot_cv,text_tfidf_cv,title_tfidf_cv,price_standardized_cv,
prev_project_standardized_cv)).tocsr()
S_TFIDF_cv.shape
```

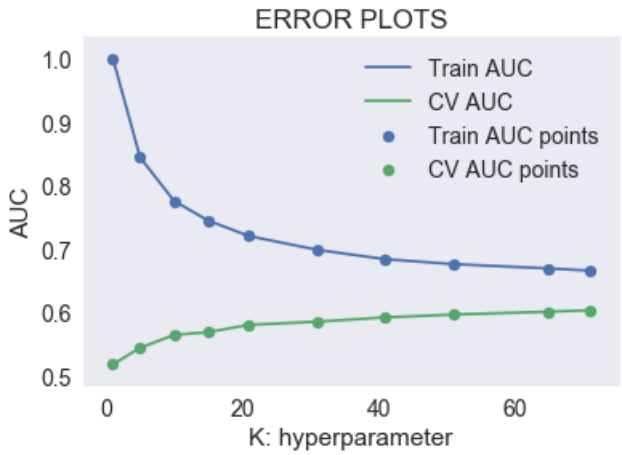
Out[86]:

```
(24155, 14213)
```

In [0]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
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(S_TFIDF_train, y_train)
    y_train_pred = batch_predict(neigh,S_TFIDF_train)
    y_cv_pred = batch_predict(neigh, S_TFIDF_cv)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
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 [1:21:44<00:00, 489.12s/it]
```



best_k=21

Using Randomized searchcv for finding best hyper parameter for TFIDF

In [74]:

```
neigh = KNeighborsClassifier()
k_range=np.arange(1,100)
weights=["uniform","distance"]
parameters=dict(n_neighbors=k_range,weights=weights)
clf = RandomizedSearchCV(neigh, parameters, cv=5, scoring='roc_auc')
best_model_1=clf.fit(S_TFIDF_train, y_train)
print('Best C:', best_model_1.best_estimator_.get_params()['n_neighbors'])
```

```
Best C: 89
```

best_k=89

best_k=89

we will choose k=21 from CV-AUC graph

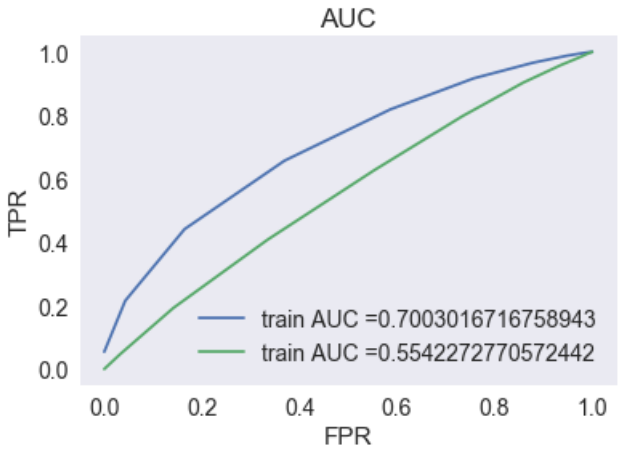
In [87]:

```
from sklearn.metrics import roc_curve, auc
neigh = KNeighborsClassifier(n_neighbors=21, n_jobs=-1)
neigh.fit(S_TFIDF_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, S_TFIDF_train)
y_test_pred = batch_predict(neigh, S_TFIDF_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="train AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("AUC")
plt.grid()
plt.show()
```



CONFUSION MATRIX FOR TRAIN DATA-TFIDF

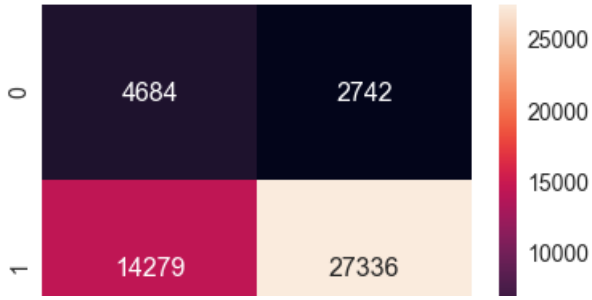
In [88]:

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

```
the maximum value of tpr*(1-fpr) 0.414330599461242 for threshold 0.857
```

Out[88]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf993e8f28>
```



CONFUSION MATRIX ON TEST DATA-TFIDF

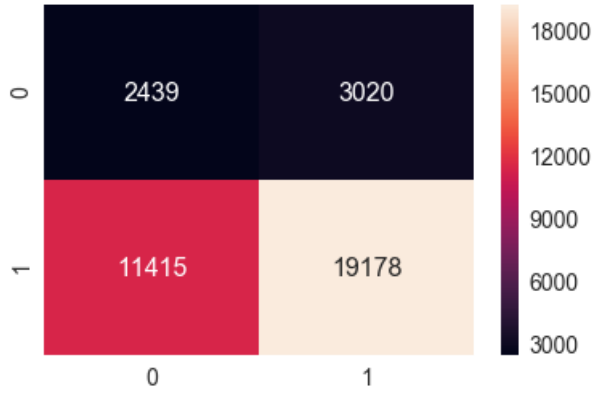
In [89]:

```
conf_matr_df_test = pd.DataFrame(confusion_matrix(y_test, prediction(y_test_pred, tr_thresholds,tra
in_fpr, train_tpr)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.414330599461242 for threshold 0.857
```

Out[89]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf9cd4b7f0>
```



2.4.3 Applying KNN brute force on AVG W2V, SET 3

note: due to memory issues i have used only 20k points train:12k test:6k cv:2k

In [90]:

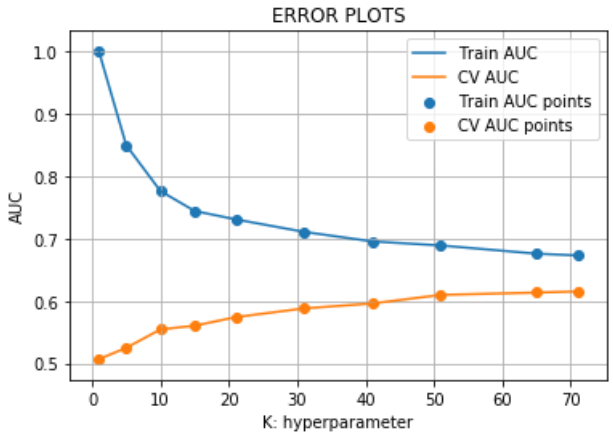
```
# Please write all the code with proper documentation
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
S_avgw2v_train=
hstack((categories_one_hot_train,sub_categories_one_hot_train,school_state_one_hot_train,teacher_pr
efix_one_hot_train,clean_project_grade_category_one_hot_train,avg_w2v_vectors_train,avg_w2v_ppt_tra
in,price_standardized_train,prev_project_standardized_train)).tocsr()
S_avgw2v_train.shape
S_avgw2v_train_new = S_avgw2v_train[0:12000,:]
S_avgw2v_test=
hstack((categories_one_hot_test,sub_categories_one_hot_test,school_state_one_hot_test,teacher_prefi
x_one_hot_test,clean_project_grade_category_one_hot_test,avg_w2v_vectors_test,avg_w2v_ppt_test,pri
ce_standardized_test,prev_project_standardized_test)).tocsr()
S_avgw2v_test.shape
S_avgw2v_test_new = S_avgw2v_test[0:6000,:]
S_avgw2v_cv=
hstack((categories_one_hot_cv,sub_categories_one_hot_cv,school_state_one_hot_cv,teacher_prefix_one_
hot_cv,clean_project_grade_category_one_hot_cv,avg_w2v_vectors_cv,avg_w2v_ppt_cv,price_standardized
_cv,prev_project_standardized_cv)).tocsr()
S_avgw2v_cv.shape
S_avgw2v_cv_new = S_avgw2v_cv[0:2000,:]
```

In [80]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
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(S_avgw2v_train_new, y_train[0:12000])
    y_train_pred = batch_predict(neigh,S_avgw2v_train_new)
    y_cv_pred = batch_predict(neigh, S_avgw2v_cv_new)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
    train_auc.append(roc_auc_score(y_train[0:12000],y_train_pred))
    cv_auc.append(roc_auc_score(y_cv[0:2000], 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 [50:16<00:00, 305.54s/it]
```



from graph we can say 51 is best K hyperparameter with a AUC value of 0.61

In [82]:

```
from scipy.stats import uniform
from sklearn.model_selection import RandomizedSearchCV
neigh = KNeighborsClassifier()
k_range=np.arange(1,100)
weights=["uniform","distance"]
parameters=dict(n_neighbors=k_range,weights=weights)
clf = RandomizedSearchCV(neigh, parameters, cv=5, scoring='roc_auc')
best_model_2=clf.fit(S_avgw2v_train_new, y_train[0:12000])
print('Best C:', best_model_2.best_estimator_.get_params()['n_neighbors'])
```

```
Best C: 87
```

In [95]:

```
#we will choose k=51
best_k=51
```

In [91]:

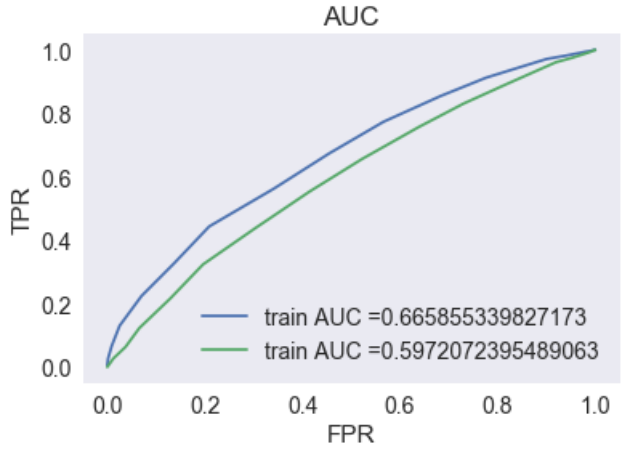
```
from sklearn.metrics import roc_curve, auc
neigh = KNeighborsClassifier(n_neighbors=51, n_jobs=-1)
neigh.fit(S_avgw2v_train_new, y_train[0:12000])
```

```
# 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, S_avgw2v_train_new)
y_test_pred = batch_predict(neigh, S_avgw2v_test_new)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train[0:12000], y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test[0:6000], 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="train AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("AUC")
plt.grid()
plt.show()
```



confusion matrix for train data using avgw2v

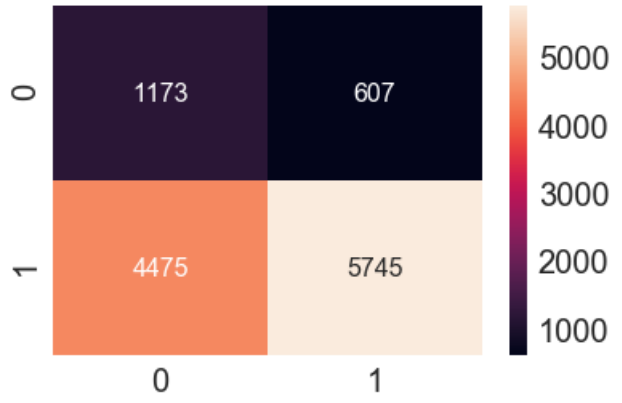
In [92]:

```
conf_matr_df_train = pd.DataFrame(confusion_matrix(y_train[0:12000], prediction(y_train_pred,
tr_thresholds,
train_fpr, train_tpr)), range(2),range(2))
sns.set(font_scale=2)#for label size
sns.heatmap(conf_matr_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.37043937861430554 for threshold 0.863
```

Out[92]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf99350240>
```



confusion matrix for avgw2v test data

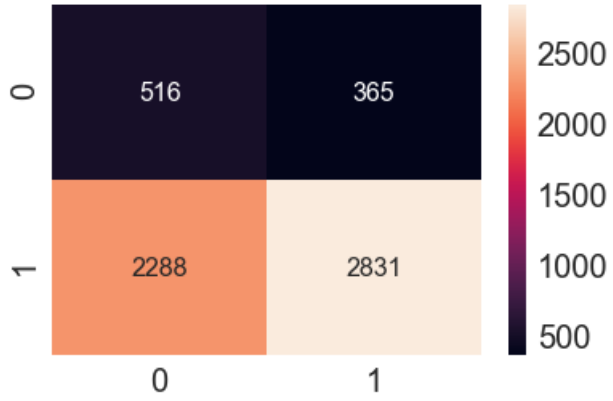
In [93]:

```
conf_matr_df_test = pd.DataFrame(confusion_matrix(y_test[0:6000], prediction(y_test_pred,
tr_thresholds,
train_fpr, train_tpr)), range(2),range(2))
sns.set(font_scale=2)#for label size
sns.heatmap(conf_matr_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.37043937861430554 for threshold 0.863
```

Out[93]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf9cbae710>
```



2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

Due to memory limitation i have used 20K points for tfidf w2v Train:12K CV:2K Test:6K

In [94]:

```
# Please write all the code with proper documentation
# Please write all the code with proper documentation
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
S_tfidf_w2v_train=
hstack((categories_one_hot_train,sub_categories_one_hot_train,school_state_one_hot_train,teacher_pr
efix_one_hot_train,clean_project_grade_category_one_hot_train,tfidf_w2v_vectors_train,tfidf_w2v_ppt
_train,price_standardized_train,prev_project_standardized_train)).tocsr()
S_tfidf_w2v_train.shape
S_tfidf_w2v_train_new=S_tfidf_w2v_train[0:12000,:]
S_tfidf_w2v_test=
hstack((categories_one_hot_test,sub_categories_one_hot_test,school_state_one_hot_test,teacher_prefi
x_one_hot_test,clean_project_grade_category_one_hot_test,tfidf_w2v_vectors_test,tfidf_w2v_ppt_test
,price_standardized_test,prev_project_standardized_test)).tocsr()
S_tfidf_w2v_test.shape
S_tfidf_w2v_test_new=S_tfidf_w2v_test[0:6000,:]
S_tfidf_w2v_cv= hstack((categories_one_hot_cv,sub_categories_one_hot_cv,school_state_one_hot_cv,te
acher_prefix_one_hot_cv,clean_project_grade_category_one_hot_cv,tfidf_w2v_vectors_cv,tfidf_w2v_ppt_
cv,price_standardized_cv,prev_project_standardized_cv)).tocsr()
S_tfidf_w2v_cv.shape

S_tfidf_w2v_cv_new=S_tfidf_w2v_cv[0:2000,:]
```

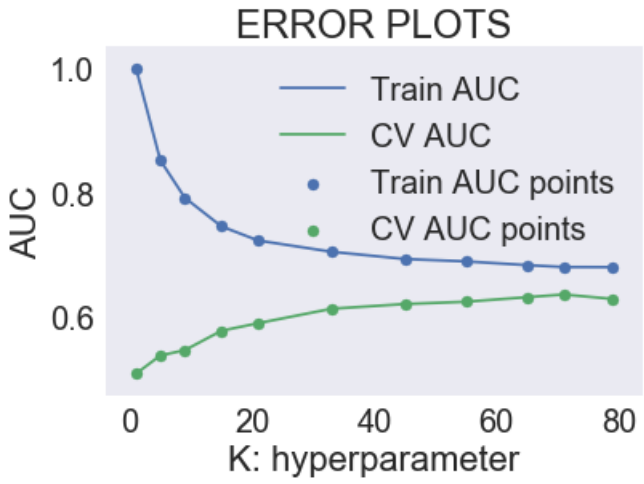
finding best k parameter using AUC

In [102]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
train_auc = []
cv_auc = []
a = []
b = []
K = [1, 5, 9, 15, 21, 33, 45, 55, 65, 71,79]
```

```
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(S_tfidf_w2v_train_new, y_train[0:12000])
    y_train_pred = batch_predict(neigh,S_tfidf_w2v_train_new)
    y_cv_pred = batch_predict(neigh, S_tfidf_w2v_cv_new)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
    train_auc.append(roc_auc_score(y_train[0:12000],y_train_pred))
    cv_auc.append(roc_auc_score(y_cv[0:2000], 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()
```

```
  0%|          | 0/11 [00:00<?, ?it/s]
  9%|▉         | 1/11 [04:09<41:35, 249.53s/it]
 18%|█▊        | 2/11 [08:07<36:55, 246.16s/it]
 27%|██▋       | 3/11 [12:05<32:28, 243.58s/it]
 36%|███▋      | 4/11 [16:07<28:21, 243.10s/it]
 45%|████▌     | 5/11 [20:15<24:27, 244.63s/it]
 55%|█████▍    | 6/11 [24:18<20:20, 244.08s/it]
 64%|██████▎   | 7/11 [28:19<16:13, 243.31s/it]
 73%|███████▎  | 8/11 [32:15<12:03, 241.02s/it]
 82%|████████▏ | 9/11 [36:24<08:07, 243.54s/it]
 91%|█████████ | 10/11 [40:43<04:07, 247.89s/it]
100%|██████████| 11/11 [44:50<00:00, 247.64s/it]
```



from AUC graph we can say 35 is best value for K

Using best RandomizedSearchCV to find best k

In [103]:

```
neigh = KNeighborsClassifier()
```

```
k_range=np.arange(1,100)
weights=["uniform","distance"]
parameters=dict(n_neighbors=k_range,weights=weights)
clf = RandomizedSearchCV(neigh, parameters, cv=5, scoring='roc_auc')
best_model_3=clf.fit(S_tfidf_w2v_train_new, y_train[0:12000])
print('Best C:', best_model_3.best_estimator_.get_params()['n_neighbors'])
```

```
Best C: 92
```

Using best k value in knn classfier

In [108]:

```
#we will choose
best_k=35
```

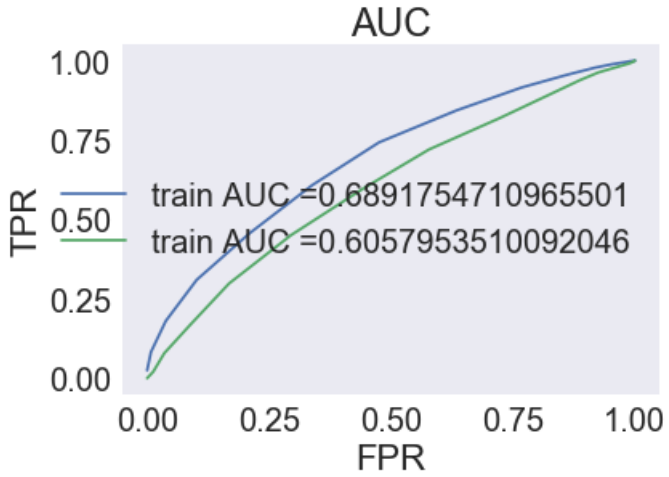
In [95]:

```
from sklearn.metrics import roc_curve, auc
neigh = KNeighborsClassifier(n_neighbors=35, n_jobs=-1)
neigh.fit(S_tfidf_w2v_train_new, y_train[0:12000])
# 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, S_tfidf_w2v_train_new)
y_test_pred = batch_predict(neigh, S_tfidf_w2v_test_new)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train[0:12000], y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test[0:6000], 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="train AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("AUC")
plt.grid()
plt.show()
```



confusion matrix for train data

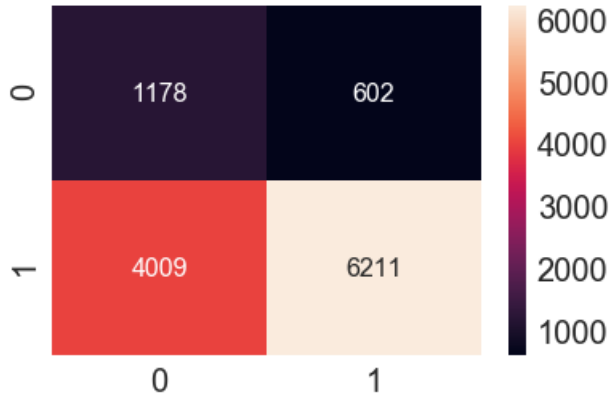
In [96]:

```
conf_matr_df_train = pd.DataFrame(confusion_matrix(y_train[0:12000], prediction(y_train_pred,
tr_thresholds,
train_fpr, train_tpr)), range(2),range(2))
sns.set(font_scale=2)#for label size
sns.heatmap(conf_matr_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.4021943094615097 for threshold 0.857
```

Out[96]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf96988438>
```



confusion matrix for test data

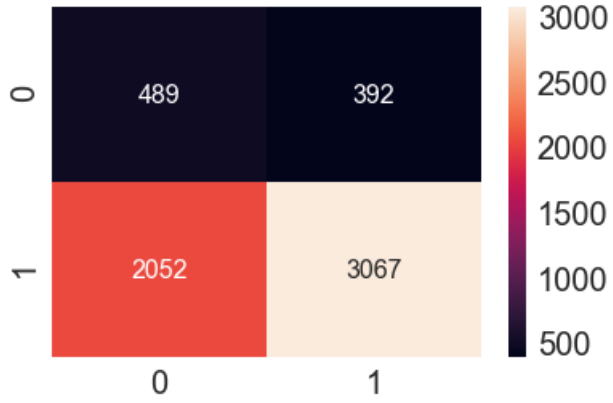
In [97]:

```
conf_matr_df_test = pd.DataFrame(confusion_matrix(y_test[0:6000], prediction(y_test_pred,
tr_thresholds,
train_fpr, train_tpr)), range(2),range(2))
sns.set(font_scale=2)#for label size
sns.heatmap(conf_matr_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.4021943094615097 for threshold 0.857
```

Out[97]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf9ef3cf28>
```



2.5 Feature selection with `SelectKBest`

In [113]:

```
# 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
S_TFIDF_train=
hstack((categories_one_hot_train,sub_categories_one_hot_train,school_state_one_hot_train,teacher_pr
efix_one_hot_train,clean_project_grade_category_one_hot_train,text_tfidf_train,title_tfidf_train,p
rice_standardized_train,prev_project_standardized_train)).tocsr()
S_TFIDF_test=
hstack((categories_one_hot_test,sub_categories_one_hot_test,school_state_one_hot_test,teacher_prefi
x_one_hot_test,clean_project_grade_category_one_hot_test,text_tfidf_test,title_tfidf_test,price_sta
ndardized_test,prev_project_standardized_test)).tocsr()
S_TFIDF_cv=
hstack((categories_one_hot_cv,sub_categories_one_hot_cv,school_state_one_hot_cv,teacher_prefix_one_
```

```
not_cv,clean_project_grade_category_one_hot_cv,text_tfidf_cv,title_tfidf_cv,price_standardized_cv,
prev_project_standardized_cv)).tocsr()
```

In [118]:

```
from sklearn.feature_selection import SelectKBest, chi2
KBest=SelectKBest(chi2, k=2000)
KBest.fit(S_TFIDF_train,y_train)
S_train_new = KBest.transform(abs(S_TFIDF_train))
S_test_new = KBest.transform(abs(S_TFIDF_test))
S_cv_new = KBest.transform(abs(S_TFIDF_cv))

print("Final Data matrix")
print(S_train_new.shape, y_train.shape)
print(S_test_new.shape, y_test.shape)
print(S_cv_new.shape, y_cv.shape)
print("*"*50)
```

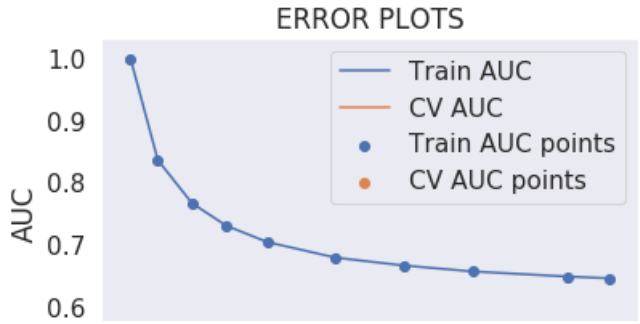
```
Final Data matrix
(49041, 2000) (49041,)
(36052, 2000) (36052,)
(24155, 2000) (24155,)
**************************************************
```

finding best k value using AUC

In [84]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
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(S_train_new, y_train)
    y_train_pred = batch_predict(neigh,S_train_new)
    y_cv_pred = batch_predict(neigh, S_cv_new)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
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 [35:58<00:00, 216.95s/it]
```



from AUC graph K=17 at AUC value of 0.55

Using Randomized searchcv for finding best hyper parameter for best 2000 features of TFIDF

In [85]:

```
neigh = KNeighborsClassifier()
k_range=np.arange(1,100)
weights=["uniform","distance"]
parameters=dict(n_neighbors=k_range,weights=weights)
clf = RandomizedSearchCV(neigh, parameters, cv=5, scoring='roc_auc')
best_model_1=clf.fit(S_train_new, y_train)
print('Best C:', best_model_1.best_estimator_.get_params()['n_neighbors'])
```

```
Best C: 96
```

In [122]:

```
best_k=17
```

In [119]:

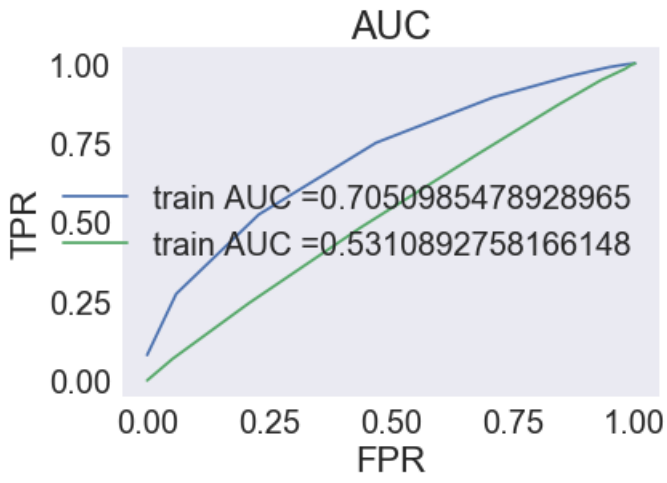
```
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=17, n_jobs=-1)
neigh.fit(S_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, S_train_new)
y_test_pred = batch_predict(neigh, S_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="train AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("AUC")
plt.grid()
plt.show()
```



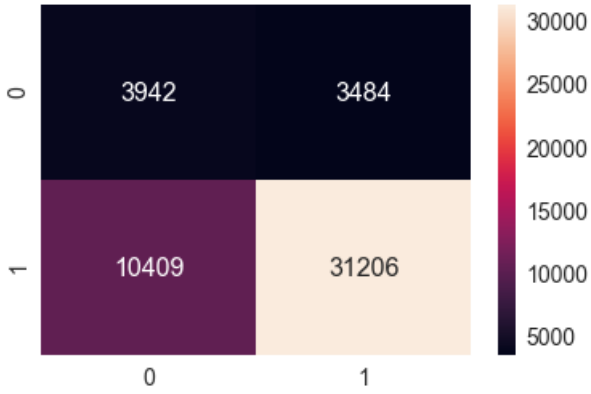
In [120]:

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

```
the maximum value of tpr*(1-fpr) 0.40370475656984056 for threshold 0.824
```

Out[120]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf9f66ac18>
```



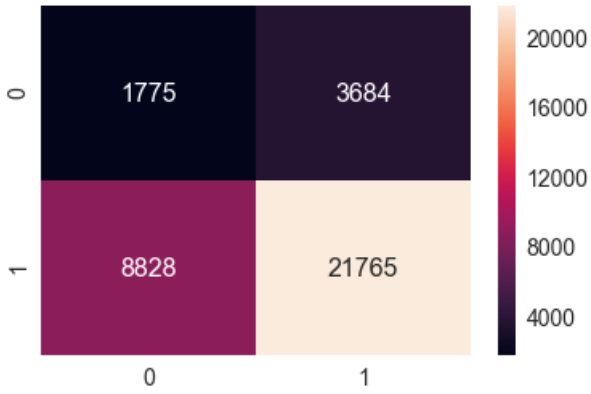
In [121]:

```
conf_matr_df_test = pd.DataFrame(confusion_matrix(y_test, prediction(y_test_pred, tr_thresholds,
train_fpr, train_tpr)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.40370475656984056 for threshold 0.824
```

Out[121]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf9cb30b70>
```



3. Conclusions

In [123]:

```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable
p = PrettyTable()
p.field_names = ["Vectorizer", "Model", "Hyper Parameter", "AUC"]
p.add_row(["BOW", "Brute", 15, 0.62])
p.add_row(["TFIDF", "Brute", 21, 0.57])
```

```
p.add_row(["AVG W2V", "Brute", 51, 0.61])
p.add_row(["TFIDF W2V", "Brute", 35, 0.63])
p.add_row(["TFIDF", "Top 2000", 17, 0.55])
print(p)
```

```
+------------+----------+-----------------+------+
| Vectorizer |  Model   | Hyper Parameter | AUC  |
+------------+----------+-----------------+------+
|    BOW     |  Brute   |        15       | 0.62 |
|   TFIDF    |  Brute   |        21       | 0.57 |
|  AVG W2V   |  Brute   |        51       | 0.61 |
| TFIDF W2V  |  Brute   |        35       | 0.63 |
|   TFIDF    | Top 2000 |        17       | 0.55 |
+------------+----------+-----------------+------+
```

Conclusion: 1.KNN takes lot of time to compute greater the number of points and dimensions more the time it takes to compute.
2.Space complexity is also high for KNN.