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
```

1.1 Reading Data

In [2]:

```
project_data = pd.read_csv('train_data.csv',nrows=50000)
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 (50000, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [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_categ
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5

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
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]))
```

In [7]:

```
print(my_counter)
```

```
Counter({'Literacy_Language': 23998, 'Math_Science': 18874, 'Health_Sports': 6538, 'SpecialNeeds':
6233, 'AppliedLearning': 5569, 'Music_Arts': 4699, 'History_Civics': 2689, 'Warmth': 643,
'Care_Hunger': 643})
```

1.3 preprocessing of `project_subject_subcategories`

In [8]:

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

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

project_data['clean_subcategories'] = sub_cat_list
project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
for word in project_data['clean_subcategories'].values:
    my_counter.update(word.split())

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

In [9]:

```
# 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 [10]:

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

Out[10]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_categ
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5

Splitting Train and Test Data

In [11]:

```
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)

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)
```

Text Preprocessing

In [12]:

```
# printing some random reviews
print(X_train['essay'].values[0])
print("="*50)
print(X_train['essay'].values[150])
print("="*50)
print(X_train['essay'].values[1000])
print("="*50)
print(X_train['essay'].values[20000])
print("="*50)
```

```
My students come from diverse backgrounds. A large portion of my school's population come from pov
erty-stricken homes; poverty is their everyday dark cloud. They are kids with hopes and dreams of
making their lives better and bettering the future of their families. They deserve a chance to hav
e hope. That hope begins with giving them a voice. They bring their suggestions to us and we
listen. We do whatever it takes to help them find success in school and life. It is our hope that
their many successes in school translate into finding a better lives for themselves and their fami
lies.A student backpack is a necessity for student success, but all too often it's a luxury for so
me students. This project will empower my students. They will no longer have to hide their bag und
er their desk so no one will see the rips, holes and stains. They will be able to feel comfortable
knowing they won't have to be embarrassed by the condition of their backpack.  Giving my students
confidence will allow them to focus on being the best students they can be.  The thought of the sm
iles that these bags will put on my students' faces warms my soul. Generous donors give my
students opportunities beyond comprehension.nannan
==================================================
I support our classrooms as they integrate technology into their learning.  I provide learning
opportunities for students from PreK to 12th grade of varying socioeconomic backgrounds in all con
tent areas.\r\n\r\nThese students are eager to learn, design and solve as we prepare them to be co
llaborative workers, critical thinkers, skilled problem solvers, effective communicators, and glob
```

```
al citizens.  Many of my students need extension opportunities to grow them as a learner and allow
them to work together using many types of resources.These materials (along with already created pu
zzles via Breakout.edu) will allow my students to practice their skills in a one of a kind
engaging way that they love! They need varying types of locks to solve challenging puzzles of a
wide variety. By providing toolboxes, UV lights and locks you help them to really engage in a Brea
koutEdu escape session as they use critical thinking skills to engage in content all while thinkin
g they are playing a game.\r\n\r\nI want to support my students to be critical thinkers and collab
orative workers!  Breakout boxes will provide an engaging opportunity while growing
learners!\r\n\r\nnannan
==================================================
India, Antarctica, Europe, Asia, England, Mexico, America... these are just a few of the places my
students \"time travel\" to in the various books and periods of time we read in class from the tim
e of the Ice Age to the ending of America's Civil War. I love to watch them compare and contrast a
nd wonder about the people, animals and different cultures they read about as the sands of the hou
r-glass pass our time together.\r\n\r\nI call my students my \"young scholars.\" They are growing
and learning in the same small, tight knit, school community that inspired me to become an
educator when I was young. I was once a student in the same classroom that they now call home.  Th
ey are dreamers, thinkers, talkers, problem-solvers, and questioners. They are inspirations to the
mselves and others around them, even if they don't quite know it yet. They are budding
ambassadors, world travelers, scientists, entrepreneurs, teachers...and LEADERS. They are our futu
re and I want to give them a great start to their long successful career.Organization and
preparation is key. In addition, having space for all of the tangible items you wish to share with
your students is also important!\r\n\r\nTechnology is a great aid in helping me organize my
materials on my computer in electronic file folders for my students.\r\nHowever, I've also
collected amazing literacy books and various items from different periods of history over the year
s that relate to the content I share with my students through our time of learning together. When
planning for my lessons, I want to find better ways to showcase these items for my students, and h
elp my classroom become even more organized in the process. Whether we are showcasing Studies Week
ly history newspapers on the new easel and finding key words with the pointers, or I set up statio
ns for my groups for hands-on activities where they can actually work together to manipulate the i
tems, our materials will be right there for us within reach!nannan
==================================================
My school has approximately 770 students. 68% of the students at my school receive free or reduced
price lunches. I teach gifted third, fourth, and fifth graders. They absolutely love a challenge.
Critical thinking, creative thinking, and perseverance are staples in our
classroom.\r\n\r\nReading is key to success in all subjects. Nowadays, you can't take a math or sc
ience test without knowing how to read. \r\n\r\nI have two groups of high-level readers who are re
ady for a challenge in my enrichment reading group.At a recent science contact meeting in my
district, I was introduced to a company called \"Back to the Roots\". These young men learned how
to grow mushrooms with used coffee grounds. In six years, they have expanded their business to inc
lude organic herb kits and mushroom kits. I introduced this company to my fifth grade students thr
ough their online video and told them about the young man who came to my meeting to talk about the
inspiration behind their business. My students were very interested in growing their own food in o
ur classroom. We took a look at the items available through Amazon and found a few herbs that will
work really well in our classroom with our GrowLab, an indoor lighting system to grow plants. The
mushrooms will grow anywhere in the classroom.nannan
==================================================
```

In [13]:

```
# 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 [14]:

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

```
I teach at a Title 1 school, with 73% of my students who receive free/reduced lunch. Our school pr
ovides free breakfast for all students. I am a Special Education certified teacher and I teach Kin
dergarten in a general education setting with my class that consists 52% students with special nee
ds. The disabilities include Autism Spectrum Disorder, Speech Impaired, Language Impaired, Other H
ealth Impaired (ADHD), and Developmentally Delayed. I also have about 42% of my students who are E
nglish Language Learners.\r\n\r\n\"Self-motivated learners\" is a synonym of \"my students\".  The
y love to learn and they possess a positive outlook and attitude in school. Almost everyday, my st
udents would ask me, \"Ms. Perez, what are we going to learn today?\" I could not ask for a better
greeting from my students.This project will greatly impact my students' learning on a daily basis.
The wobble chairs will provide assistance for my students who have difficulties focusing and atten
ding during lessons and discussions. Despite the fact that students participate in physical activi
ties in P.E., Recess, and GoNoodle (dance videos) sessions in our classroom, students still have e
nergy to stand or wiggle from their seats during lessons. Due to these special needs that are
beyond the students' control, there is a lot of distraction and student learning is not really ach
ieved at its full potential. The lack of appropriate stimulation hinders them to focus and learn i
n class. Students with special needs will be able to sit on the wobble chairs during whole
group/small group lessons. This will enable their little active bodies to move while "sitting stil
l" without disrupting other students. As a result, all students will improve focus and increase st
udent attention in learning all content areas. In addition, the visual timer will help my students
to actually see the allotted time for activities. This will benefit especially ELL students and st
udents with special needs. Whenever we do independent classwork or work in our centers, the studen
ts can refer to it and self-monitor their progress in completing assignments. It will encourage th
em to use their time wisely and finish tasks on time. It will also help the students have a smooth
er transition from one activity to another. \r\nBy donating to this project, you will
significantly help students with special needs have an equal opportunity to learn with their
peers. Behavior issues will be greatly minimized and classroom management will be optimized. Help
me set all students for success! I am looking forward to seeing my students become active listener
s and engaged learners, and always happy to go to school!\r\nnannan
==================================================
```

In [15]:

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

```
I teach at a Title 1 school, with 73% of my students who receive free/reduced lunch. Our school pr
ovides free breakfast for all students. I am a Special Education certified teacher and I teach Kin
dergarten in a general education setting with my class that consists 52% students with special nee
ds. The disabilities include Autism Spectrum Disorder, Speech Impaired, Language Impaired, Other H
ealth Impaired (ADHD), and Developmentally Delayed. I also have about 42% of my students who are E
nglish Language Learners.     Self-motivated learners  is a synonym of  my students .  They love
to learn and they possess a positive outlook and attitude in school. Almost everyday, my students
would ask me,  Ms. Perez, what are we going to learn today?  I could not ask for a better greeting
from my students.This project will greatly impact my students' learning on a daily basis. The wobb
le chairs will provide assistance for my students who have difficulties focusing and attending dur
ing lessons and discussions. Despite the fact that students participate in physical activities in
P.E., Recess, and GoNoodle (dance videos) sessions in our classroom, students still have energy to
stand or wiggle from their seats during lessons. Due to these special needs that are beyond the st
udents' control, there is a lot of distraction and student learning is not really achieved at its
full potential. The lack of appropriate stimulation hinders them to focus and learn in class. Stud
ents with special needs will be able to sit on the wobble chairs during whole group/small group le
ssons. This will enable their little active bodies to move while "sitting still" without
disrupting other students. As a result, all students will improve focus and increase student atten
tion in learning all content areas. In addition, the visual timer will help my students to actuall
y see the allotted time for activities. This will benefit especially ELL students and students wit
h special needs. Whenever we do independent classwork or work in our centers, the students can ref
er to it and self-monitor their progress in completing assignments. It will encourage them to use
their time wisely and finish tasks on time. It will also help the students have a smoother transit
ion from one activity to another.   By donating to this project, you will significantly help stude
nts with special needs have an equal opportunity to learn with their peers. Behavior issues will b
e greatly minimized and classroom management will be optimized. Help me set all students for succe
ss! I am looking forward to seeing my students become active listeners and engaged learners, and a
lways happy to go to school!  nannan
```

In [16]:

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

```
I teach at a Title 1 school with 73 of my students who receive free reduced lunch Our school
provides free breakfast for all students I am a Special Education certified teacher and I teach Ki
ndergarten in a general education setting with my class that consists 52 students with special nee
ds The disabilities include Autism Spectrum Disorder Speech Impaired Language Impaired Other Healt
h Impaired ADHD and Developmentally Delayed I also have about 42 of my students who are English La
nguage Learners Self motivated learners is a synonym of my students They love to learn and they po
ssess a positive outlook and attitude in school Almost everyday my students would ask me Ms Perez
what are we going to learn today I could not ask for a better greeting from my students This proje
ct will greatly impact my students learning on a daily basis The wobble chairs will provide assist
ance for my students who have difficulties focusing and attending during lessons and discussions D
espite the fact that students participate in physical activities in P E Recess and GoNoodle dance
videos sessions in our classroom students still have energy to stand or wiggle from their seats du
ring lessons Due to these special needs that are beyond the students control there is a lot of dis
traction and student learning is not really achieved at its full potential The lack of appropriate
stimulation hinders them to focus and learn in class Students with special needs will be able to
sit on the wobble chairs during whole group small group lessons This will enable their little
active bodies to move while sitting still without disrupting other students As a result all
students will improve focus and increase student attention in learning all content areas In
addition the visual timer will help my students to actually see the allotted time for activities T
his will benefit especially ELL students and students with special needs Whenever we do
independent classwork or work in our centers the students can refer to it and self monitor their p
rogress in completing assignments It will encourage them to use their time wisely and finish tasks
on time It will also help the students have a smoother transition from one activity to another By
donating to this project you will significantly help students with special needs have an equal
opportunity to learn with their peers Behavior issues will be greatly minimized and classroom
management will be optimized Help me set all students for success I am looking forward to seeing m
y students become active listeners and engaged learners and always happy to go to school nannan
```

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"]
```

Preprocessed Train Data for Text

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%|██████████████████████████████████| 22445/22445 [00:10<00:00, 2214.17it/s]
```

In [19]:

```
# after preprocesing train data
preprocessed_essays_train[20000]
```

Out[19]:

```
'school approximately 770 students 68 students school receive free reduced price lunches teach gif
ted third fourth fifth graders absolutely love challenge critical thinking creative thinking perse
verance staples classroom reading key success subjects nowadays not take math science test without
knowing read two groups high level readers ready challenge enrichment reading group recent science
contact meeting district introduced company called back roots young men learned grow mushrooms use
d coffee grounds six years expanded business include organic herb kits mushroom kits introduced co
mpany fifth grade students online video told young man came meeting talk inspiration behind
business students interested growing food classroom took look items available amazon found herbs w
ork really well classroom growlab indoor lighting system grow plants mushrooms grow anywhere class
room nannan'
```

In [20]:

```
X_test.head(2)
```

Out[20]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_cate
42067	58359	p216326	ad988abfb641e9a08002a62311df7f4b	Mrs.	NM	2016-05-31 02:05:29	Grades PreK-2
6182	143971	p249148	63f6a0a917742a5afb3d46d801110c75	Mrs.	OH	2016-09-01 15:10:17	Grades 3-5

Preprocessed Test Data for Text

In [21]:

```
#Preprocessed Test data

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%|████████████████████████████████| 16500/16500 [00:07<00:00, 2209.72it/s]
```

In [22]:

```
# after preprocesing test data
preprocessed_essays_test[16000]
```

Out[22]:

```
'although students title 1 school high poverty level still love reading love feel good book hands especially love takes wonderful adventure love making students feel special important future help feel successful school almost thousand students strive help feel loved materials requested used incentives library birthday book club help make club successful turn adds new books school library celebrating life child make big impact lives successes many emotional issues children today something little making feel special birthday small price pay taking time students not big deal major deal part birthday club take students photo bulletin board special polaroid type camera make happen much faster also eat lunch special area school lobby much fun sharing conversations getting know personal basis children truly blessing nannan'
```

Preprocessed Cross Validation Data for Text

In [23]:

```
#Preprocessed Cross Validation data

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%|████████████████████████████████| 11055/11055 [00:04<00:00, 2249.15it/s]
```

In [24]:

```
# after preprocesing cv data
preprocessed_essays_cv[11000]
```

Out[24]:

```
'going play today often hear students spotlight students day exercising pe students excited look forward p e students able unwind play games fun structured environment school serves students grades pre k 4 provides rigorous academics dedicated teachers safe learning environment result one top performing schools district goal provide best learning experience possible students school located small bayou community half students receive free reduce lunch however school not qualify title money innovative new materials must purchased grants different kinds playground balls used teach students skills needed participate various activities games p e students learn dribble throw catch skills needed play games ultimately get students moving exercising teacher first demonstrate skill students practice skill skill practiced several times game played incorporate skill donation project improve current p e curriculum providing playground balls needed students learn practice necessary skills students ball fun p e class nannan'
```

Preprocessing of Project_title for Train Data

In [25]:

```
# similarly you can preprocess the titles also
#for train data
from tqdm import tqdm
preprocessed_title_train = []
# tqdm is for printing the status bar
```

```
for sentance in tqdm(X_train['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_title_train.append(sent.lower().strip())
```

```
100%|████████████████████████████████| 22445/22445 [00:00<00:00, 48687.64it/s]
```

Preprocessing of Project_title for Test Data

In [26]:

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

```
100%|████████████████████████████████| 16500/16500 [00:00<00:00, 48387.12it/s]
```

Preprocessing of Project_title for Cross Validation Data

In [27]:

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

```
100%|████████████████████████████████| 11055/11055 [00:00<00:00, 47857.15it/s]
```

Preprocessing of project_teacher_prefix

In [28]:

```
teacher_prefix = list(project_data['teacher_prefix'].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

teacher_prefix_list = []
for i in teacher_prefix:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in str(i).split(','): # it will split it in three parts ["Math & Science", "Warmth", "Car
e & Hunger"]
```

```
        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('&','_')
    teacher_prefix_list.append(temp.strip())

project_data['clean_teacher_prefix'] = teacher_prefix_list
project_data.drop(['teacher_prefix'], axis=1, inplace=True)

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
for word in project_data['clean_teacher_prefix'].values:
    word=word.replace('.',' ')
    my_counter.update(word.split())

teacher_prefix_dict = dict(my_counter)
del teacher_prefix_dict['nan']
sorted_teacher_prefix_dict = dict(sorted(teacher_prefix_dict.items(), key=lambda kv: kv[1]))
print(sorted_teacher_prefix_dict)
```

```
{'Dr': 2, 'Teacher': 1061, 'Mr': 4859, 'Ms': 17936, 'Mrs': 26140}
```

Preprocessing of project_grade_category

In [29]:

```
project_grade_category = 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

project_grade_category_list = []
for i in project_grade_category:
    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 placing 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('&','_')

    project_grade_category_list.append(temp.strip())

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

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
for word in project_data['clean_project_grade_category'].values:
    my_counter.update(word.split())


project_grade_category_dict = dict(my_counter)
sorted_project_grade_category_dict = dict(sorted(project_grade_category_dict.items(), key=lambda
kv: kv[1]))
print (sorted_project_grade_category_dict)
```

```
{'Grades9-12': 4966, 'Grades6-8': 7750, 'Grades3-5': 16968, 'GradesPreK-2': 20316}
```

Preprocessing of school_state

In [30]:

```
school_state = list(project_data['school_state'].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

school_state_list = []
for i in school_state:
    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('&','_')
    school_state_list.append(temp.strip())

project_data['clean_school_state'] = school_state_list
project_data.drop(['school_state'], axis=1, inplace=True)

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
for word in project_data['clean_school_state'].values:
    my_counter.update(word.split())

school_state_dict = dict(my_counter)
sorted_school_state_dict = dict(sorted(school_state_dict.items(), key=lambda kv: kv[1]))
print(sorted_school_state_dict)
```

```
{'VT': 32, 'WY': 51, 'ND': 63, 'MT': 106, 'RI': 126, 'NH': 141, 'SD': 142, 'NE': 144, 'AK': 153,
'DE': 155, 'WV': 218, 'ME': 222, 'NM': 236, 'HI': 239, 'DC': 247, 'KS': 285, 'ID': 302, 'IA': 306,
'AR': 446, 'CO': 538, 'MN': 556, 'OR': 577, 'MS': 598, 'KY': 614, 'NV': 665, 'MD': 668, 'TN': 774,
'CT': 774, 'AL': 790, 'UT': 792, 'WI': 833, 'VA': 916, 'AZ': 994, 'NJ': 1005, 'OK': 1074, 'MA': 107
6, 'LA': 1094, 'WA': 1103, 'MO': 1166, 'IN': 1171, 'OH': 1180, 'PA': 1419, 'MI': 1468, 'GA': 1828,
'SC': 1830, 'IL': 1967, 'NC': 2340, 'FL': 2839, 'TX': 3320, 'NY': 3393, 'CA': 7024}
```

Preparing data for models

In [31]:

```
project_data.columns
```

Out[31]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', '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', 'essay',
       'clean_teacher_prefix', 'clean_project_grade_category',
       'clean_school_state'],
      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

- teacher_prefix : categorical data

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

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

Vectorizing Categorical data

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

One Hot Encoding for Categories

In [32]:

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

from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True
)
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  (22445, 9)
Shape of matrix of Test data after one hot encoding  (16500, 9)
Shape of matrix of CV data after one hot encoding  (11055, 9)
```

One Hot Encoding for Sub-Categories

In [33]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=
True)
vectorizer.fit(X_train['clean_subcategories'].values)
subcategories_one_hot_train = vectorizer.transform(X_train['clean_subcategories'].values)
subcategories_one_hot_test = vectorizer.transform(X_test['clean_subcategories'].values)
subcategories_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 ",subcategories_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",subcategories_one_hot_test.shape)
print("Shape of matrix of CV data after one hot encoding ",subcategories_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  (22445, 30)
Shape of matrix of Test data after one hot encoding  (16500, 30)
Shape of matrix of CV data after one hot encoding  (11055, 30)
```

One Hot Encoding for Teacher_prefix

In [34]:

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

```
['Dr', 'Teacher', 'Mr', 'Ms', 'Mrs']
Shape of matrix after one hot encoding  (22445, 5)
Shape of matrix after one hot encoding  (16500, 5)
Shape of matrix after one hot encoding  (11055, 5)
```

One Hot Encoding for project_grade_category

In [35]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_project_grade_category_dict.keys()), lowercase
=False, binary=True)
vectorizer.fit(X_train['project_grade_category'].values.astype("U"))
project_grade_category_one_hot_train =
vectorizer.transform(X_train['project_grade_category'].values.astype("U"))
project_grade_category_one_hot_test = vectorizer.transform(X_test['project_grade_category'].values
.astype("U"))
project_grade_category_one_hot_cv =
vectorizer.transform(X_cv['project_grade_category'].values.astype("U"))
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encoding ",project_grade_category_one_hot_train.shape)
print("Shape of matrix after one hot encoding ",project_grade_category_one_hot_test.shape)
print("Shape of matrix after one hot encoding ",project_grade_category_one_hot_cv.shape)
```

```
['Grades9-12', 'Grades6-8', 'Grades3-5', 'GradesPreK-2']
Shape of matrix after one hot encoding  (22445, 4)
Shape of matrix after one hot encoding  (16500, 4)
Shape of matrix after one hot encoding  (11055, 4)
```

One Hot Encoding for school_state

In [36]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_school_state_dict.keys()), lowercase=False, bi
nary=True)
vectorizer.fit(X_train['school_state'].values.astype("U"))
school_state_one_hot_train = vectorizer.transform(X_train['school_state'].values.astype("U"))
school_state_one_hot_test = vectorizer.transform(X_test['school_state'].values.astype("U"))
school_state_one_hot_cv = vectorizer.transform(X_cv['school_state'].values.astype("U"))
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encoding ",school_state_one_hot_train.shape)
print("Shape of matrix after one hot encoding ",school_state_one_hot_test.shape)
print("Shape of matrix after one hot encoding ",school_state_one_hot_cv.shape)
```

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

Vectorizing Text data

Bag of words - Text Train Data

In [37]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10)
text_bow_train = vectorizer.fit_transform(preprocessed_essays_train)
print("Shape of matrix after one hot encodig ",text_bow_train.shape)
```

```
Shape of matrix after one hot encodig  (22445, 8782)
```

Bag of words - Text Test Data

In [38]:

```
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  (16500, 8782)
```

Bag of words - Text CV Data

In [39]:

```
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  (11055, 8782)
```

Bag of words - Title Train Data

In [40]:

```
# before you vectorize the title make sure you preprocess it
vectorizer = CountVectorizer(min_df=10)
title_bow_train = vectorizer.fit_transform(preprocessed_title_train)
print("Shape of matrix after one hot encoding ",title_bow_train.shape)
```

```
Shape of matrix after one hot encoding  (22445, 1234)
```

Bag of words - Title Test Data

In [41]:

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

```
Shape of matrix after one hot encodig  (16500, 1234)
```

Bag of words - Title CV Data

In [42]:

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

```
Shape of matrix after one hot encodig  (11055, 1234)
```

TFIDF vectorizer - Text train data

In [43]:

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

```
Shape of matrix after one hot encodig  (22445, 8782)
```

TFIDF vectorizer - Text test data

In [44]:

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

```
Shape of matrix after one hot encodig  (16500, 8782)
```

TFIDF vectorizer - Text cv data

In [45]:

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

```
Shape of matrix after one hot encodig  (11055, 8782)
```

TFIDF vectorizer - Title train data

In [46]:

```
title_tfidf_train = vectorizer.fit_transform(preprocessed_title_train)
print("Shape of matrix after one hot encodig ",title_tfidf_train.shape)
```

```
Shape of matrix after one hot encodig  (22445, 1234)
```

TFIDF vectorizer - Title Test data

In [47]:

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

```
Shape of matrix after one hot encodig  (16500, 1234)
```

TFIDF vectorizer - Title CV data

In [48]:

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

```
Shape of matrix after one hot encodig  (11055, 1234)
```

Using Pretrained Models: Avg W2V

In [49]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-sa
ve-and-load-variables-in-python/
# make sure you have the glove_vectors file
```

```
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

Avg W2V - Text train Data

In [50]:

```
# 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%|██████████████████████████████████| 22445/22445 [00:04<00:00, 4666.32it/s]
```

```
22445
300
```

Avg W2V - Text test Data

In [51]:

```
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%|██████████████████████████████████| 16500/16500 [00:03<00:00, 4578.25it/s]
```

```
16500
300
```

Avg W2V - Text CV Data

In [52]:

```
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)
```

```
    avg_w2v_vectors_cv.append(vector)

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

```
100%|██████████████████████████████████| 11055/11055 [00:02<00:00, 4592.85it/s]
```

```
11055
300
```

Avg W2V - Title train Data

In [53]:

```
# Similarly you can vectorize for title also for train data
avg_w2v_vectors_title_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_title_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_title_train.append(vector)

print(len(avg_w2v_vectors_title_train))
print(len(avg_w2v_vectors_title_train[0]))
```

```
100%|██████████████████████████████████| 22445/22445 [00:00<00:00, 71480.86it/s]
```

```
22445
300
```

Avg W2V - Title Test Data

In [54]:

```
# Similarly you can vectorize for title also for test data
avg_w2v_vectors_title_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_title_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_title_test.append(vector)

print(len(avg_w2v_vectors_title_test))
print(len(avg_w2v_vectors_title_test[0]))
```

```
100%|██████████████████████████████████| 16500/16500 [00:00<00:00, 71739.12it/s]
```

```
16500
300
```

Avg W2V - Title CV Data

In [55]:

```
# Similarly you can vectorize for title also for cv data
avg_w2v_vectors_title_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_title_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_title_cv.append(vector)

print(len(avg_w2v_vectors_title_cv))
print(len(avg_w2v_vectors_title_cv[0]))
```

```
100%|██████████████████████████████████| 11055/11055 [00:00<00:00, 69968.46it/s]
```

```
11055
300
```

Using Pretrained Models: TFIDF weighted W2V

In [56]:

```
# 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 Weighted W2V - Text Train Data

In [57]:

```
# average Word2Vec
# compute average word2vec for each review.
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%|██████████████████████████████████| 22445/22445 [00:33<00:00, 672.30it/s]
```

```
22445
300
```

TFIDF Weighted W2V - Text Test Data

In [58]:

```
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%|██████████████████████████████████| 16500/16500 [00:24<00:00, 668.21it/s]
```

```
16500
300
```

TFIDF Weighted W2V - Text CV Data

In [59]:

```
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%|██████████████████████████████████| 11055/11055 [00:16<00:00, 685.02it/s]
```

```
11055
300
```

TFIDF Weighted W2V - Title Train Data

In [60]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_title_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())
```

In [61]:

```
tfidf_w2v_vectors_title_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_title_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_title_train.append(vector)

print(len(tfidf_w2v_vectors_title_train))
print(len(tfidf_w2v_vectors_title_train[0]))
```

```
100%|██████████████████████████████████| 22445/22445 [00:00<00:00, 33350.68it/s]
```

```
22445
300
```

TFIDF Weighted W2V - Title Test Data

In [62]:

```
tfidf_w2v_vectors_title_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_title_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_title_test.append(vector)

print(len(tfidf_w2v_vectors_title_test))
print(len(tfidf_w2v_vectors_title_test[0]))
```

```
100%|██████████████████████████████████| 16500/16500 [00:00<00:00, 33604.88it/s]
```

```
16500
300
```

TFIDF Weighted W2V - Title CV Data

In [63]:

```
tfidf_w2v_vectors_title_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_title_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_title_cv.append(vector)

print(len(tfidf_w2v_vectors_title_cv))
print(len(tfidf_w2v_vectors_title_cv[0]))
```

```
100%|██████████████████████████████████| 11055/11055 [00:00<00:00, 34873.83it/s]
```

```
11055
300
```

Vectorizing Numerical features

In [64]:

```
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 [65]:

```
# join two dataframes in python:
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')
```

Vectorizing Price

In [66]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()


normalizer.fit(X_train['price'].values.reshape(-1,1))
price_train = normalizer.transform(X_train['price'].values.reshape(-1,1))
price_test = normalizer.transform(X_test['price'].values.reshape(-1,1))
price_cv = normalizer.transform(X_cv['price'].values.reshape(-1,1))

print("After vectorizations")
print(price_train.shape, y_train.shape)
print(price_test.shape, y_test.shape)
print(price_cv.shape, y_cv.shape)
```

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

Vectorizing Quantity

In [67]:

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

quantity_train = normalizer.transform(X_train['quantity'].values.reshape(-1,1))
quantity_test = normalizer.transform(X_test['quantity'].values.reshape(-1,1))
quantity_cv = normalizer.transform(X_cv['quantity'].values.reshape(-1,1))

print("After vectorizations")
print(quantity_train.shape, y_train.shape)
print(quantity_test.shape, y_test.shape)
print(quantity_cv.shape, y_cv.shape)
```

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

Vectorizing Number of Projects previously proposed by Teacher

In [68]:

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

prev_projects_train = normalizer.transform(X_train['teacher_number_of_previously_posted_projects']
.values.reshape(-1,1))
prev_projects_test = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].v
alues.reshape(-1,1))
prev_projects_cv =
normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

print("After vectorizations")
print(prev_projects_train.shape, y_train.shape)
print(prev_projects_test.shape, y_test.shape)
print(prev_projects_cv.shape, y_cv.shape)
```

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

Assignment 3: Apply KNN

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

please refer to this prettytable library link

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link.

K Nearest Neighbor

SET 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)

In [69]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((categories_one_hot_train, subcategories_one_hot_train,school_state_one_hot_train, p
roject_grade_category_one_hot_train,
teacher_prefix_one_hot_train, price_train, quantity_train, prev_projects_train, title_bow_train, te
xt_bow_train)).tocsr()

X_te = hstack((categories_one_hot_test, subcategories_one_hot_test,school_state_one_hot_test,
project_grade_category_one_hot_test,
teacher_prefix_one_hot_test, price_test, quantity_test, prev_projects_test, title_bow_test,
text_bow_test)).tocsr()

X_cv = hstack((categories_one_hot_cv, subcategories_one_hot_cv,school_state_one_hot_cv,
project_grade_category_one_hot_cv,
teacher_prefix_one_hot_cv, price_cv, quantity_cv, prev_projects_cv, title_bow_cv, text_bow_cv)).toc
sr()

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_te.shape, y_test.shape)
print(X_cv.shape, y_cv.shape)
```

```
Final Data matrix
(22445, 10104) (22445,)
(16500, 10104) (16500,)
(11055, 10104) (11055,)
```

Find the best hyper parameter which results in the maximum AUC value

In [74]:

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

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000

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

# we will be predicting for the last data points
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])
    return y_data_pred
```

In [71]:

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

```
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []
a = []
b = []
K = [10, 21, 31, 41, 51, 61, 81, 91, 95, 101]

for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_tr, y_train)
    y_train_pred = batch_predict(neigh, X_tr)
    y_cv_pred = batch_predict(neigh, X_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))
    print("difference between AUC score of train and CV data :
",roc_auc_score(y_train,y_train_pred)-roc_auc_score(y_cv, y_cv_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
    print("AUC of CV data",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()
```

```
  0%|                                                    | 0/10 [00:00<?, ?it/s]
```

```
difference between AUC score of train and CV data :  0.17146941674200322
AUC of CV data 0.5975252716922546
```

```
 10%|████                                               | 1/10 [00:54<08:06, 54.07s/it]
```

```
difference between AUC score of train and CV data :  0.1126656179955059
AUC of CV data 0.6104327316637133
```

```
 20%|████████                                           | 2/10 [01:46<07:08, 53.51s/it]
```

```
difference between AUC score of train and CV data :  0.09835384818593229
AUC of CV data 0.6090218765192028
```

```
 30%|████████████                                       | 3/10 [02:38<06:11, 53.08s/it]
```

```
difference between AUC score of train and CV data :  0.08455281086463484
AUC of CV data 0.6102835635987267
```

```
 40%|████████████████                                   | 4/10 [03:30<05:17, 52.85s/it]
```

```
difference between AUC score of train and CV data :  0.07621996966265265
AUC of CV data 0.6098132262769145
```

```
 50%|████████████████████                               | 5/10 [04:22<04:22, 52.59s/it]
```

```
difference between AUC score of train and CV data :  0.06916123713779865
AUC of CV data 0.6115800021954931
```

```
 60%|████████████████████████                           | 6/10 [05:14<03:29, 52.46s/it]
```

```
difference between AUC score of train and CV data :  0.061113667948361816
AUC of CV data 0.6138343343735789
```

```
 70%|████████████████████████████                       | 7/10 [06:06<02:36, 52.30s/it]
```

```
difference between AUC score of train and CV data :  0.053768354064700374
AUC of CV data 0.6174689729797544

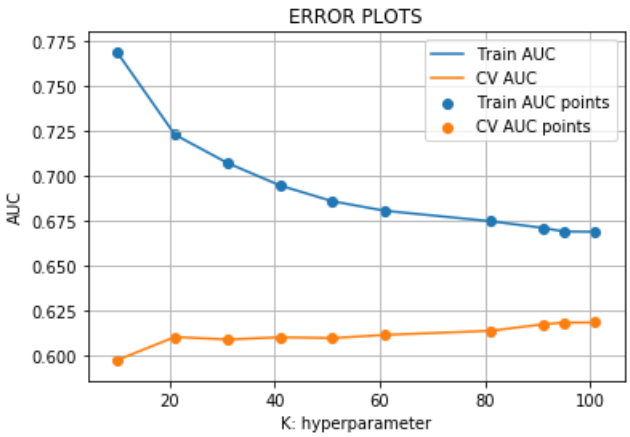
 80%|████████████████████████████████          | 8/10 [06:58<01:44, 52.25s/it]

difference between AUC score of train and CV data :  0.050806665850972155
AUC of CV data 0.6184010852008093

 90%|████████████████████████████████████      | 9/10 [07:51<00:52, 52.24s/it]

difference between AUC score of train and CV data :  0.05051463648316723
AUC of CV data 0.6185326893220631

100%|████████████████████████████████████████| 10/10 [08:43<00:00, 52.27s/it]
```



In [72]:

```
best_k = 91
```

Train model using the best hyper-parameter value

In [73]:

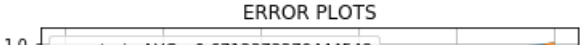
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
from sklearn.neighbors import KNeighborsClassifier

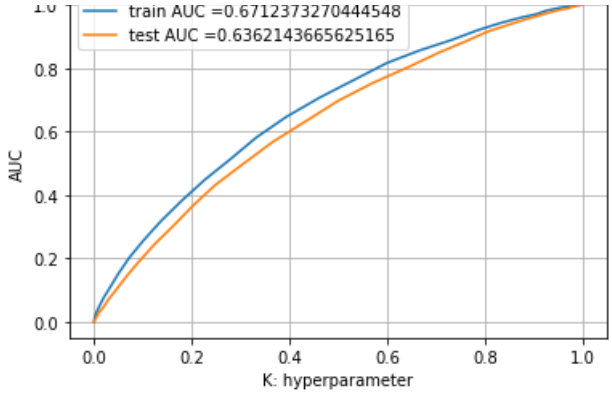
neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

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

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

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





Confusion matrix

In [79]:

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

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

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

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

Confusion matrix for train data

In [75]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24894464137986413 for threshold 0.769
[[ 1619  1844]
 [ 4497 14485]]
```

In [76]:

```
conf_matr_df_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds,
train_fpr, train_fpr)), range(2),range(2))
```

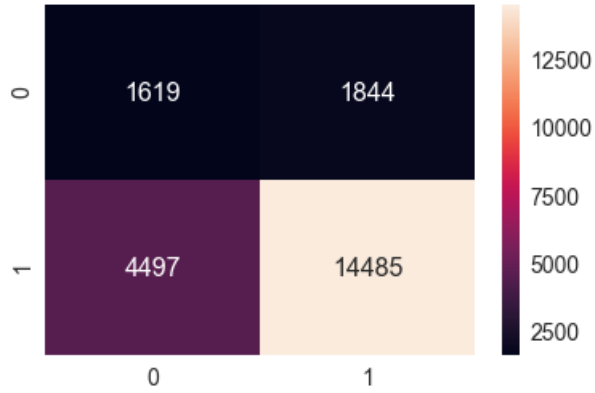
```
the maximum value of tpr*(1-fpr) 0.24894464137986413 for threshold 0.769
```

In [77]:

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

Out[77]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x16c27a90>
```



Confusion matrix for test data

In [78]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

Test confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.78
[[1273 1273]
 [4226 9728]]
```

In [79]:

```
conf_matr_df_test_2 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, tes
t_fpr, test_fpr)), range(2),range(2))
```

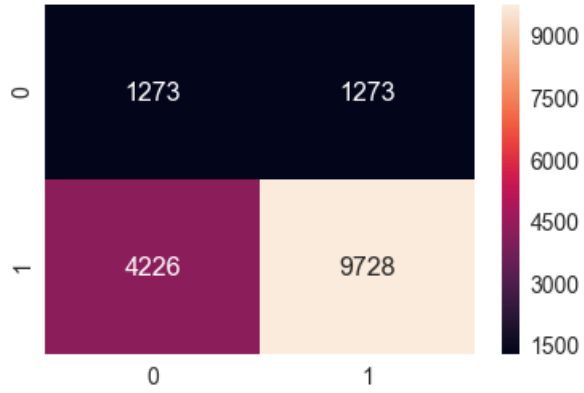
```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.78
```

In [80]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_2, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[80]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x15946a58>
```



SET 2: categorical, numerical features + project_title(TFIDF) + preprocessed_essay (TFIDF)

In [81]:

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

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((categories_one_hot_train, subcategories_one_hot_train,school_state_one_hot_train, p
roject_grade_category_one_hot_train,
teacher_prefix_one_hot_train, price_train, quantity_train, prev_projects_train, title_tfidf_train,
text_tfidf_train)).tocsr()

X_te = hstack((categories_one_hot_test, subcategories_one_hot_test,school_state_one_hot_test,
project_grade_category_one_hot_test,
teacher_prefix_one_hot_test, price_test, quantity_test, prev_projects_test, title_tfidf_test,
text_tfidf_test)).tocsr()

X_cv = hstack((categories_one_hot_cv, subcategories_one_hot_cv,school_state_one_hot_cv,
project_grade_category_one_hot_cv,
teacher_prefix_one_hot_cv, price_cv, quantity_cv, prev_projects_cv, title_tfidf_cv, text_tfidf_cv))
.tocsr()
```

In [82]:

```
print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_te.shape, y_test.shape)
print(X_cv.shape, y_cv.shape)
print("="*100)
```

```
Final Data matrix
(22445, 10104) (22445,)
(16500, 10104) (16500,)
(11055, 10104) (11055,)
====================================================================================================
```

Find the best hyper parameter which results in the maximum AUC value

In [83]:

```
train_auc = []
cv_auc = []
a = []
b = []
K = [10, 21, 31, 41, 51, 61, 81, 91, 95, 101]

for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_tr, y_train)
    y_train_pred = batch_predict(neigh, X_tr)
    y_cv_pred = batch_predict(neigh, X_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))
    print("difference between AUC score of train and CV data :
",roc_auc_score(y_train,y_train_pred)-roc_auc_score(y_cv, y_cv_pred))
    print("AUC of CV data",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.show()
```

```
  0%|                                                    | 0/10 [00:00<?, ?it/s]
```

```
difference between AUC score of train and CV data :  0.2403227216863444
AUC of CV data 0.5223839603556698
```

```
 10%|████                                              | 1/10 [00:50<07:36, 50.68s/it]

difference between AUC score of train and CV data :  0.163899590403906
AUC of CV data 0.5353162921260213

 20%|████████                                          | 2/10 [01:41<06:46, 50.79s/it]

difference between AUC score of train and CV data :  0.13186404962695453
AUC of CV data 0.545344708077846

 30%|████████████                                      | 3/10 [02:32<05:56, 50.88s/it]

difference between AUC score of train and CV data :  0.1132500440699784
AUC of CV data 0.5521977198237333

 40%|████████████████                                  | 4/10 [03:24<05:05, 50.98s/it]

difference between AUC score of train and CV data :  0.101892548866381
AUC of CV data 0.5533341697116064

 50%|████████████████████                              | 5/10 [04:15<04:15, 51.04s/it]

difference between AUC score of train and CV data :  0.09224347083818374
AUC of CV data 0.5555601173020527

 60%|████████████████████████                          | 6/10 [05:06<03:24, 51.09s/it]

difference between AUC score of train and CV data :  0.08382854031942355
AUC of CV data 0.5550153841328586

 70%|████████████████████████████                      | 7/10 [05:57<02:33, 51.06s/it]

difference between AUC score of train and CV data :  0.07560766520205586
AUC of CV data 0.5577678736650619

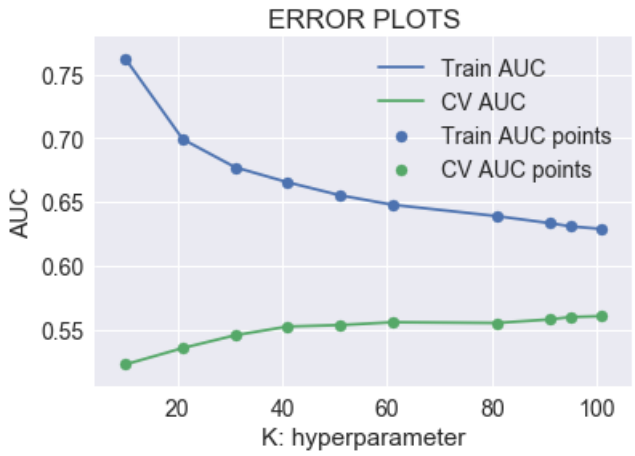
 80%|████████████████████████████████                  | 8/10 [06:48<01:42, 51.11s/it]

difference between AUC score of train and CV data :  0.07117892311501517
AUC of CV data 0.5596436401273386

 90%|████████████████████████████████████              | 9/10 [07:42<00:51, 51.84s/it]

difference between AUC score of train and CV data :  0.06840937372540201
AUC of CV data 0.5603759311242491

100%|████████████████████████████████████████| 10/10 [08:33<00:00, 51.65s/it]
```



Train model using the best hyper-parameter value

In [84]:

```
best_k = 81
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
from sklearn.neighbors import KNeighborsClassifier

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

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

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

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

plt.show()
```

confusion matrix for train data

In [85]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24890678397237445 for threshold 0.84
[[ 1846  1617]
 [ 6506 12476]]
```

In [86]:

```
conf_matr_df_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds,
train_fpr, train_fpr)), range(2),range(2))
```

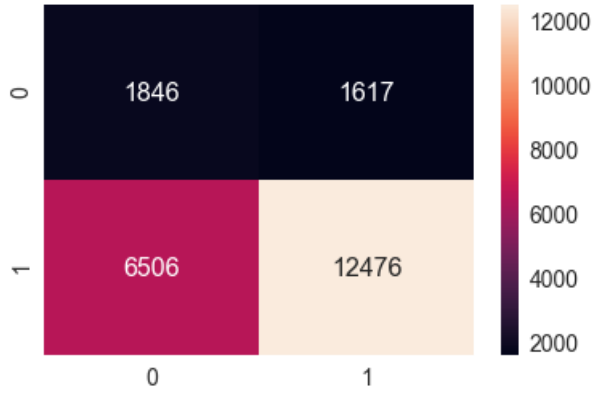
the maximum value of tpr*(1-fpr) 0.24890678397237445 for threshold 0.84

In [87]:

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

Out[87]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x16bb4748>
```



confusion matirx for test data

In [88]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24875040804576776 for threshold 0.84
[[1183 1363]
 [4991 8963]]
```

In [89]:

```
conf_matr_df_test_2 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, tes
t_fpr, test_fpr)), range(2),range(2))
```

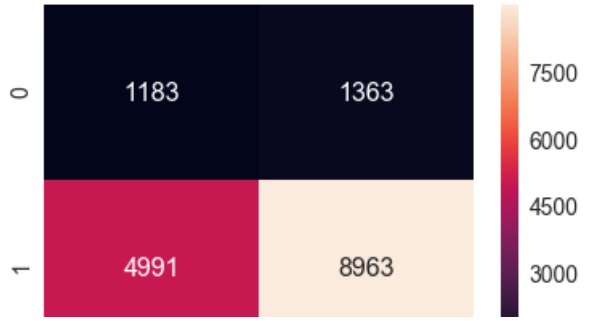
```
the maximum value of tpr*(1-fpr) 0.24875040804576776 for threshold 0.84
```

In [90]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_2, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[90]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x16c1c668>
```



SET 3: categorical, numerical features + project_title(avg_w2v) + preprocessed_essay (avg_w2v)

In [91]:

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

# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((categories_one_hot_train, subcategories_one_hot_train,school_state_one_hot_train, p
roject_grade_category_one_hot_train,
teacher_prefix_one_hot_train, price_train, quantity_train, prev_projects_train, avg_w2v_vectors_tit
le_train, avg_w2v_vectors_train)).tocsr()

X_te = hstack((categories_one_hot_test, subcategories_one_hot_test,school_state_one_hot_test,
project_grade_category_one_hot_test,
teacher_prefix_one_hot_test, price_test, quantity_test, prev_projects_test,
avg_w2v_vectors_title_test, avg_w2v_vectors_test)).tocsr()

X_cv = hstack((categories_one_hot_cv, subcategories_one_hot_cv,school_state_one_hot_cv,
project_grade_category_one_hot_cv,
teacher_prefix_one_hot_cv, price_cv, quantity_cv, prev_projects_cv, avg_w2v_vectors_title_cv, avg_w
2v_vectors_cv)).tocsr()
```

In [92]:

```
print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_te.shape, y_test.shape)
print(X_cv.shape, y_cv.shape)
print("="*100)
```

```
Final Data matrix
(22445, 702) (22445,)
(16500, 702) (16500,)
(11055, 702) (11055,)
====================================================================================================
```

Find the best hyper parameter which results in the maximum AUC value

In [93]:

```
train_auc = []
cv_auc = []
a = []
b = []
K = [10, 21, 31, 41, 51, 61, 81, 91, 95, 101]

for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_tr, y_train)
    y_train_pred = batch_predict(neigh, X_tr)
    y_cv_pred = batch_predict(neigh, X_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))
    print("difference between AUC score of train and CV data :
",roc_auc_score(y_train,y_train_pred)-roc_auc_score(y_cv, y_cv_pred))
    print("AUC of CV data",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")
```

```
  0%|                                                    | 0/10 [00:00<?, ?it/s]
```

```
difference between AUC score of train and CV data :  0.23392956454980063
AUC of CV data 0.5448446061442439
```

```
 10%|████                                      | 1/10 [11:40<1:45:02, 700.23s/it]
```

```
difference between AUC score of train and CV data :  0.15939883989654124
AUC of CV data 0.5589706274405256
```

```
 20%|████████                                  | 2/10 [23:16<1:33:13, 699.18s/it]
```

```
difference between AUC score of train and CV data :  0.12355364419425752
AUC of CV data 0.5695346495836404
```

```
 30%|████████████                              | 3/10 [34:55<1:21:32, 698.98s/it]
```

```
difference between AUC score of train and CV data :  0.10545876241360785
AUC of CV data 0.5728223062085405
```

```
 40%|████████████████                          | 4/10 [46:30<1:09:46, 697.80s/it]
```

```
difference between AUC score of train and CV data :  0.09029156681832096
AUC of CV data 0.5763933068828705
```

```
 50%|████████████████████                        | 5/10 [58:03<58:02, 696.50s/it]
```

```
difference between AUC score of train and CV data :  0.08597482959453184
AUC of CV data 0.5750624304107141
```

```
 60%|████████████████████████                  | 6/10 [1:09:37<46:22, 695.58s/it]
```

```
difference between AUC score of train and CV data :  0.07231769282089795
AUC of CV data 0.5804306302633023
```

```
 70%|████████████████████████████              | 7/10 [1:21:10<34:44, 694.96s/it]
```

```
difference between AUC score of train and CV data :  0.0690110611288024
AUC of CV data 0.5791251901453731
```

```
 80%|████████████████████████████████          | 8/10 [1:32:46<23:10, 695.11s/it]
```

```
difference between AUC score of train and CV data :  0.06868402885094049
AUC of CV data 0.5792356548057772
```

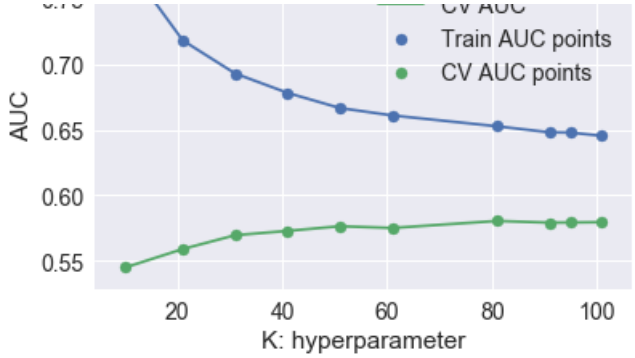
```
 90%|████████████████████████████████████      | 9/10 [1:44:15<11:33, 693.46s/it]
```

```
difference between AUC score of train and CV data :  0.06600366572126604
AUC of CV data 0.5794741480703185
```

```
100%|████████████████████████████████████████| 10/10 [1:55:52<00:00, 694.23s/it]
```

Out[93]:

```
Text(0.5,1,'ERROR PLOTS')
```



Train model using the best hyper-parameter value

In [94]:

```
best_k = 91
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
from sklearn.neighbors import KNeighborsClassifier

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

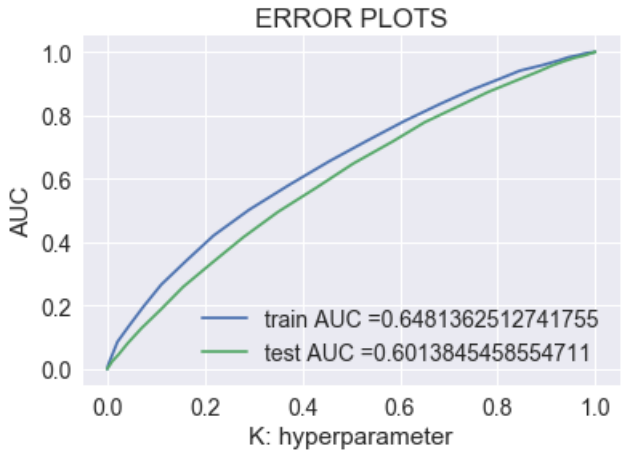
y_train_pred = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, X_te)

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

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

Out[94]:

```
Text(0.5,1,'ERROR PLOTS')
```



confusion matrix for train data

In [95]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
```

```
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24876902970547354 for threshold 0.835
[[ 1610  1853]
 [ 5274 13708]]
```

In [96]:

```
conf_matr_df_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds,
train_fpr, train_fpr)), range(2),range(2))
```

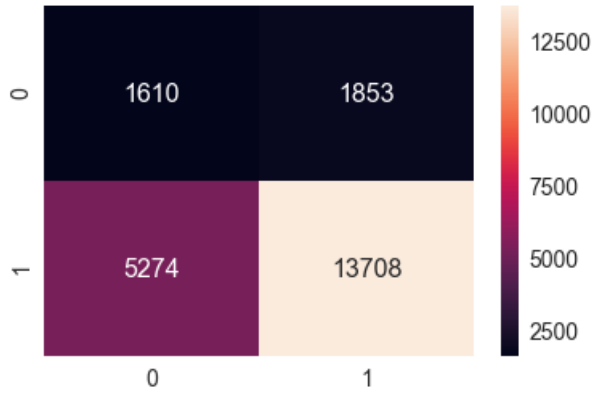
```
the maximum value of tpr*(1-fpr) 0.24876902970547354 for threshold 0.835
```

In [97]:

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

Out[97]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x16b87940>
```



confusion matirx for test data

In [98]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24998750408045767 for threshold 0.846
[[1264 1282]
 [4920 9034]]
```

In [99]:

```
conf_matr_df_test_2 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, tes
t_fpr, test_fpr)), range(2),range(2))
```

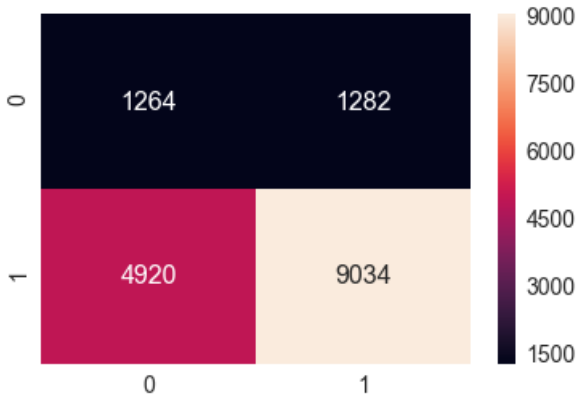
```
the maximum value of tpr*(1-fpr) 0.24998750408045767 for threshold 0.846
```

In [100]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_2, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[100]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x16309ac8>
```



SET 4: categorical, numerical features + project_title(tfidf_w2v) + preprocessed_essay (tfidf_w2v)

In [101]:

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

# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((categories_one_hot_train, subcategories_one_hot_train,school_state_one_hot_train, p
roject_grade_category_one_hot_train,
teacher_prefix_one_hot_train, price_train, quantity_train, prev_projects_train, tfidf_w2v_vectors_t
itle_train, tfidf_w2v_vectors_train)).tocsr()

X_te = hstack((categories_one_hot_test, subcategories_one_hot_test,school_state_one_hot_test,
project_grade_category_one_hot_test,
teacher_prefix_one_hot_test, price_test, quantity_test, prev_projects_test,
tfidf_w2v_vectors_title_test, tfidf_w2v_vectors_test)).tocsr()

X_cv = hstack((categories_one_hot_cv, subcategories_one_hot_cv,school_state_one_hot_cv,
project_grade_category_one_hot_cv,
teacher_prefix_one_hot_cv, price_cv, quantity_cv, prev_projects_cv, tfidf_w2v_vectors_title_cv, tfi
df_w2v_vectors_cv)).tocsr()
```

In [102]:

```
print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_te.shape, y_test.shape)
print(X_cv.shape, y_cv.shape)
print("="*100)
```

```
Final Data matrix
(22445, 702) (22445,)
(16500, 702) (16500,)
(11055, 702) (11055,)
====================================================================================================
```

Find the best hyper parameter which results in the maximum AUC value

In [103]:

```
train_auc = []
cv_auc = []
a = []
b = []
K = [10, 21, 31, 41, 51, 61, 81, 91, 95, 101]

for i in tqdm(K):
```

```
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_tr, y_train)
    y_train_pred = batch_predict(neigh, X_tr)
    y_cv_pred = batch_predict(neigh, X_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))
    print("difference between AUC score of train and CV data :
",roc_auc_score(y_train,y_train_pred)-roc_auc_score(y_cv, y_cv_pred))
    print("AUC of CV data",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")
```

```
  0%|                                                    | 0/10 [00:00<?, ?it/s]

difference between AUC score of train and CV data :  0.22021155222849964
AUC of CV data 0.5573573792086816

 10%|████                                      | 1/10 [11:42<1:45:21, 702.43s/it]

difference between AUC score of train and CV data :  0.15153512134595115
AUC of CV data 0.573061897219565

 20%|████████                                  | 2/10 [23:26<1:33:42, 702.78s/it]

difference between AUC score of train and CV data :  0.12139604402985438
AUC of CV data 0.5804166418366867

 30%|████████████                              | 3/10 [35:09<1:22:00, 702.93s/it]

difference between AUC score of train and CV data :  0.10271452556883731
AUC of CV data 0.5865311838411718

 40%|████████████████                          | 4/10 [46:59<1:10:31, 705.19s/it]

difference between AUC score of train and CV data :  0.09293465271305279
AUC of CV data 0.5835429610927282

 50%|████████████████████                        | 5/10 [59:00<59:09, 709.90s/it]

difference between AUC score of train and CV data :  0.08766576194202857
AUC of CV data 0.5850263302335064

 60%|████████████████████████                  | 6/10 [1:10:37<47:04, 706.06s/it]

difference between AUC score of train and CV data :  0.07348675391954251
AUC of CV data 0.5918277165304937

 70%|████████████████████████████              | 7/10 [1:22:16<35:11, 703.72s/it]

difference between AUC score of train and CV data :  0.07071863546936918
AUC of CV data 0.5909883168409993

 80%|████████████████████████████████          | 8/10 [1:34:31<23:46, 713.39s/it]

difference between AUC score of train and CV data :  0.0699424653337799
AUC of CV data 0.5917528188561482
```

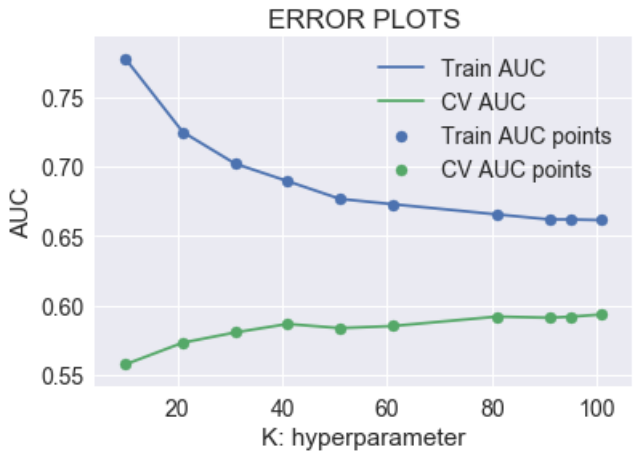
```
 90%|█████████████████████████████████████     | 9/10 [1:46:03<11:46, 706.89s/it]
```

```
difference between AUC score of train and CV data :  0.0678413079065715
AUC of CV data 0.5933691407781453
```

```
100%|██████████████████████████████████████████| 10/10 [1:57:33<00:00, 701.76s/it]
```

Out[103]:

```
Text(0.5,1,'ERROR PLOTS')
```



Train model using the best hyper-parameter value

In [104]:

```
best_k = 81
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
from sklearn.neighbors import KNeighborsClassifier

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

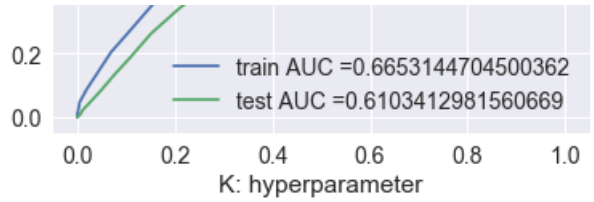
y_train_pred = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, X_te)

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

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

Out[104]:

```
Text(0.5,1,'ERROR PLOTS')
```



confusion matrix for train data

In [105]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2485580622143965 for threshold 0.827
[[ 1600  1863]
 [ 4844 14138]]
```

In [106]:

```
conf_matr_df_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds,
train_fpr, train_fpr)), range(2),range(2))
```

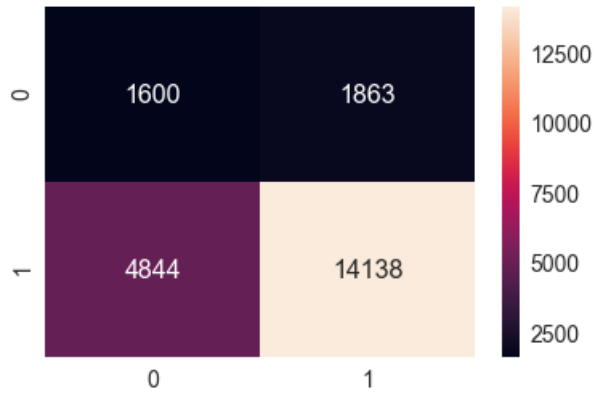
```
the maximum value of tpr*(1-fpr) 0.2485580622143965 for threshold 0.827
```

In [107]:

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

Out[107]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x159996d8>
```



confusion matrix for test data

In [108]:

```
#confusion matirx for test data
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

Test confusion matrix
```

```
the maximum value of tpr*(1-fpr) 0.24992533302396933 for threshold 0.84
[[1251 1295]
 [4737 9217]]
```

In [109]:

```
conf_matr_df_test_2 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, tes
t_fpr, test_fpr)), range(2),range(2))
```

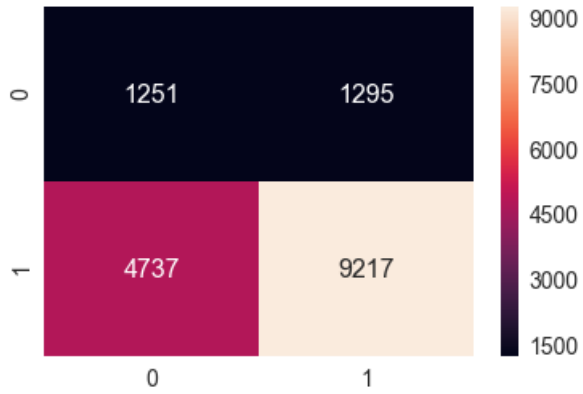
```
the maximum value of tpr*(1-fpr) 0.24992533302396933 for threshold 0.84
```

In [110]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_2, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[110]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x16e3edd8>
```



Feature selection with `SelectKBest`

In [69]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((categories_one_hot_train, subcategories_one_hot_train,school_state_one_hot_train, p
roject_grade_category_one_hot_train,
teacher_prefix_one_hot_train, price_train, quantity_train, prev_projects_train, title_tfidf_train,
text_tfidf_train)).tocsr()

X_te = hstack((categories_one_hot_test, subcategories_one_hot_test,school_state_one_hot_test,
project_grade_category_one_hot_test,
teacher_prefix_one_hot_test, price_test, quantity_test, prev_projects_test, title_tfidf_test,
text_tfidf_test)).tocsr()

X_cv = hstack((categories_one_hot_cv, subcategories_one_hot_cv,school_state_one_hot_cv,
project_grade_category_one_hot_cv,
teacher_prefix_one_hot_cv, price_cv, quantity_cv, prev_projects_cv, title_tfidf_cv, text_tfidf_cv))
.tocsr()
```

In [71]:

```
from sklearn.feature_selection import SelectKBest, chi2
#X_tr_new = SelectKBest(chi2, k=2000).fit_transform(X_tr, y_train)
#X_te_new = SelectKBest(chi2, k=2000).transform(X_te)
#X_cv_new = SelectKBest(chi2, k=2000).transform(X_cv)

clf=SelectKBest(chi2, k=2000)
X_tr_new = clf.fit_transform(X_tr, y_train)
X_te_new = clf.transform(X_te)
X_cv_new = clf.transform(X_cv)
```

In [72]:

```
print("Final Data matrix")
print(X_tr_new.shape, y_train.shape)
print(X_cv_new.shape, y_cv.shape)
print(X_te_new.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 2000) (22445,)
(11055, 2000) (11055,)
(16500, 2000) (16500,)
====================================================================================================
```

Find the best hyper parameter which results in the maximum AUC value

In [75]:

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

train_auc = []
cv_auc = []
a = []
b = []
K = [10, 21, 31, 41, 51, 61, 81, 91, 95, 101]

for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_tr, y_train)
    y_train_pred = batch_predict(neigh, X_tr)
    y_cv_pred = batch_predict(neigh, X_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))
    print("difference between AUC score of train and CV data :
",roc_auc_score(y_train,y_train_pred)-roc_auc_score(y_cv, y_cv_pred))
    print("AUC of CV data",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()
```

```
  0%|                                                    | 0/10 [00:00<?, ?it/s]
```

```
difference between AUC score of train and CV data :  0.2238508930155445
AUC of CV data 0.5355249894145875
```

```
 10%|████                                      | 1/10 [00:51<07:40, 51.21s/it]
```

```
difference between AUC score of train and CV data :  0.1448358666185573
AUC of CV data 0.5526939953267366
```

```
 20%|████████                                  | 2/10 [01:42<06:49, 51.14s/it]
```

```
difference between AUC score of train and CV data :  0.11347041893404075
AUC of CV data 0.5590564712155189
```

```
 30%|████████████                              | 3/10 [02:33<05:57, 51.13s/it]
```

```
difference between AUC score of train and CV data :  0.09926949883222236
AUC of CV data 0.5628064672949958

 40%|████████████████                                  | 4/10 [03:24<05:07, 51.30s/it]

difference between AUC score of train and CV data :  0.08592506812977385
AUC of CV data 0.5661765489986983

 50%|████████████████████                              | 5/10 [04:16<04:16, 51.23s/it]

difference between AUC score of train and CV data :  0.0834239727198407
AUC of CV data 0.5613922875468502

 60%|████████████████████████                          | 6/10 [05:07<03:25, 51.35s/it]

difference between AUC score of train and CV data :  0.06701191398212614
AUC of CV data 0.5681450593567205

 70%|████████████████████████████                      | 7/10 [05:59<02:34, 51.36s/it]

difference between AUC score of train and CV data :  0.06614744773159098
AUC of CV data 0.5643720106010947

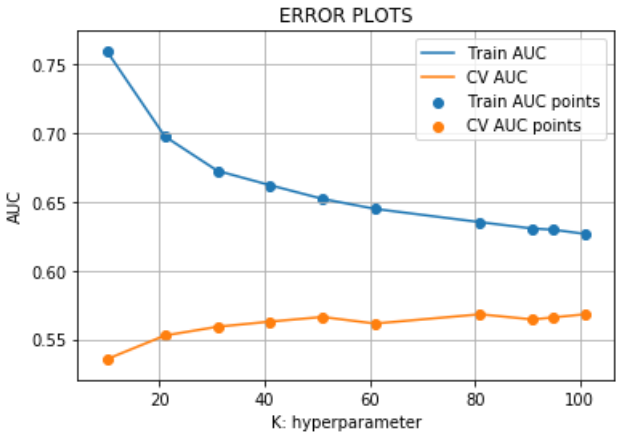
 80%|████████████████████████████████                  | 8/10 [06:50<01:42, 51.28s/it]

difference between AUC score of train and CV data :  0.06372895158963443
AUC of CV data 0.5658923581162671

 90%|████████████████████████████████████              | 9/10 [07:41<00:51, 51.30s/it]

difference between AUC score of train and CV data :  0.05842653591999769
AUC of CV data 0.5681666379161635

100%|████████████████████████████████████████| 10/10 [08:32<00:00, 51.26s/it]
```



Train model using the best hyper-parameter value

In [76]:

```
best_k = 81
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
from sklearn.neighbors import KNeighborsClassifier

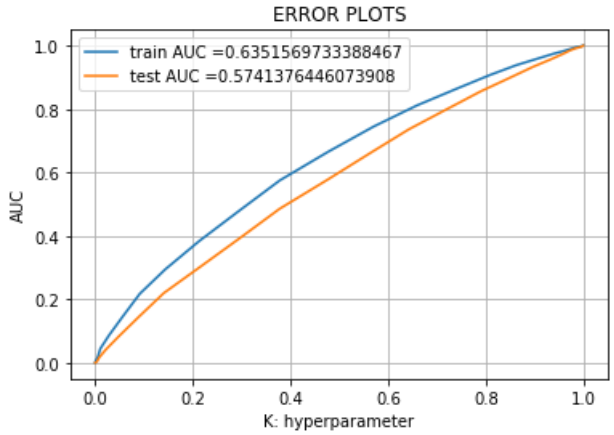
neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
```

```
# not the predicted outputs

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

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

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



confusion matrix for train data

In [80]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24947297735751794 for threshold 0.84
[[ 1811  1652]
 [ 6352 12630]]
```

In [81]:

```
conf_matr_df_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds,
train_fpr, train_fpr)), range(2),range(2))
```

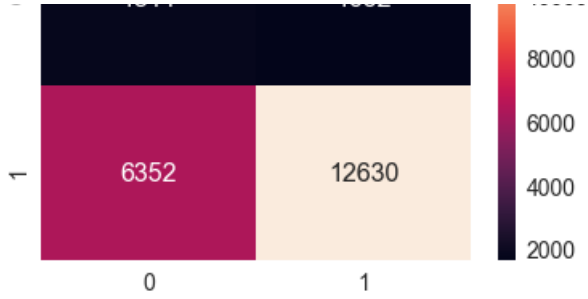
```
the maximum value of tpr*(1-fpr) 0.24947297735751794 for threshold 0.84
```

In [82]:

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

Out[82]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1619fcf8>
```



confusion matrix for test data

In [83]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24930747922437677 for threshold 0.852
[[1340 1206]
 [5926 8028]]
```

In [84]:

```
conf_matr_df_test_2 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, tes
t_fpr, test_fpr)), range(2),range(2))
```

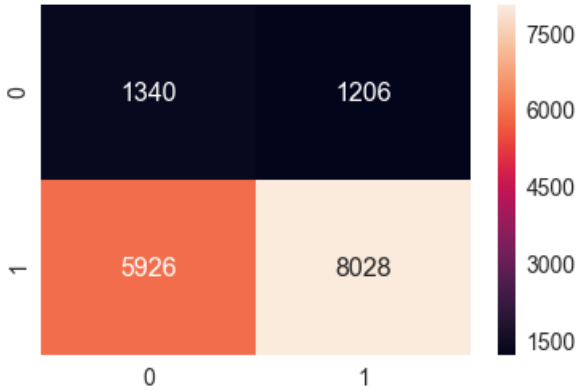
```
the maximum value of tpr*(1-fpr) 0.24930747922437677 for threshold 0.852
```

In [85]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_2, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[85]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x16412400>
```



3. Conclusions

In [86]:

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

# http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable
```

```
x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyper Parameter", "AUC"]
x.add_row(["BOW", "Brute", 91, 0.63])
x.add_row(["TFIDF", "Brute", 81, 0.57])
x.add_row(["AVG W2V", "Brute", 91, 0.60])
x.add_row(["TFIDF W2V", "Brute", 81, 0.61])
x.add_row(["TFIDF", "Top 2000", 81, 0.57])
print(x)
```

```
+------------+----------+-----------------+------+
| Vectorizer |  Model   | Hyper Parameter | AUC  |
+------------+----------+-----------------+------+
|    BOW     |  Brute   |        91       | 0.63 |
|   TFIDF    |  Brute   |        81       | 0.57 |
|  AVG W2V   |  Brute   |        91       | 0.6  |
| TFIDF W2V  |  Brute   |        81       | 0.61 |
|   TFIDF    | Top 2000 |        81       | 0.57 |
+------------+----------+-----------------+------+
```