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 p **Example:** `p036502`

Feature	Description
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 follow enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
project_subject_categories	One or more (comma-separated) su categories for the project from the fo 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, Ma & Science`

Feature	Description
school_state	State where school is located (Two-l U.S. postal code). **Example:** WY
project_subject_subcategories	One or more (comma-separated) su subcategories for the project. **Exam** • Literacy • Literature & Writing, Social Sciences
project_resource_summary	An explanation of the resources nee the project. **Example:** • My students need hands 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 w submitted. **Example:** 2016-04-28 12:43:56.245
teacher_id	A unique identifier for the teacher of proposed project. **Example:** bdf8baa8fedef6bfeec7ae4ff1c

Feature	Description
`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 previ submitted by the same teacher. **Exa** 2

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [11]:
```
%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
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.
```

1.1 Reading Data

In [12]:
```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

In [13]:
```
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 (32414, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefi
x' '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 [14]:
```
# how to replace elements in list python: https://stackoverflow.com/a/2
582163/4084039
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_d
atetime'])
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/131
48611/4084039
project_data = project_data[cols]
```

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

Out[14]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_
31477	47750	p185738	3afe10b996b7646d8641985a4b4b570d	Mrs.	UT
3287	159755	p147002	6ada7036aeb258d3653589d1f2a5b815	Mrs.	CA

In [15]:
```
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 (32414, 4)
['id' 'description' 'quantity' 'price']
```

Out[15]:

	id	description	quantity	price
0	p253737	LC652 - Lakeshore Double-Space Mobile Drying Rack	1	149.00
1	p258326	Bouncy Bands for Desks (Blue support pipes)	3	14.95

1.2 preprocessing of `project_subject_categories`

In [16]:
```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stacko
```

```
verflow.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 & H
unger"
    for j in i.split(','): # it will split it in three parts ["Math & S
cience", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory b
ased on space "Math & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are g
oing 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 int
o
    cat_list.append(temp.strip())

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

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

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

1.3 preprocessing of

project_subject_subcategories

In [17]:
```
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 & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

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

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

```
    my_counter.update(word.split())

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

1.3 Text preprocessing

In [18]:
```
# 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 [19]:
```
project_data.head(2)
```

Out[19]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_
31477	47750	p185738	3afe10b996b7646d8641985a4b4b570d	Mrs.	UT
3287	159755	p147002	6ada7036aeb258d3653589d1f2a5b815	Mrs.	CA

1.4.2.3 Using Pretrained Models: TFIDF weighted W2V

In [20]:
```
# printing some random reviews
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
print(project_data['essay'].values[1000])
print("="*50)
print(project_data['essay'].values[20000])
print("="*50)
```

```
\"The only way to learn mathematics is to do mathematics.\"\r\n-Paul Ha
lmos.\r\nMy students love math time and using the hands-on manipulativ
e's to make sense of what is being taught.My students love coming to sc
hool and working as hard as they can to learn new concepts everyday.  T
hey especially love working with the math manipulative's and they are a
lways coming up with new discoveries.\r\nOur school is a Title I school
where over 50% of our population receives free or reduced lunch. Many o
f these children come from low-income families but this doesn't hinder
their desire to learn.My students will be using these math manipulativ
e's on a daily basis during our math block time.  The students will eac
h have opportunities to explore and make connections that make math mor
e concrete and help them relate to the concepts being taught.\r\nThey w
ill be learning about our base ten number system and learn about money
which are both important real life skills.Learning about money is impor
tant so the students can feel independent when they are given or earn m
oney. They will be able to have a real sense of how much they can buy w
ith certain amounts of money.  \r\nI'm so excited to watch them exchang
e pennies for other coins and learn to save their classroom money for p
urchases in our store.
==================================================
My students are energetic 6 and 7 year olds who love to learn.   They a
re vibrant and playful in their daily activities.  They love to  engage
and learn in ways that make learning fun and memorable.\r\n\r\nEnvironm
ent shapes experience.  Where do you enjoy reading?  Do you like to rea
d in the shade, on the beach, or in your bed at night?   My students ar
e more productive, engaged, and happy when they are allowed to read in
ways that make reading magical.\r\nI teach first grade in an Indianapol
is Public School.  My school's population is 80% Hispanic. 90% of our s
tudents receive free or reduced priced lunch and 66% of them are Englis
h Language Learners.  School is a safe place for my students, and they
```

are excited to be there. My students love reading, but I want to provi
de them with reading experiences that make their reading time magical i
n ways they may have never experienced.Reading is my passion and has al
ways been an activity I do whenever possible. As a child, I loved gett
ing a flashlight and reading under my covers while I was supposed to be
sleeping. I will never forget that feeling. I want my students to loo
k forward to the joy of reading \"in the dark\" by having Flashlight F
ridays! Students will each have their own rechargeable flashlight to
use each Friday as I close the shades, turn out the lights, and let the
magic begin. I want this practice to become a habit, so that they will
continue to read in the dark, at night, long after they take home their
very own flashlight and make their own magic at home!nannan
==========
My students are hard working, humble and have huge hearts. There attitu
de and work ethic shine distinctly against the background of where they
live and come from. Our school is a Title I school where over 95% of th
e students are on free or reduced lunch. \r\n   Our school as a very l
arge population of first generation Americans. Many of their families s
acrificed very much to give their children the hope and opportunities o
ur country can provide. This unfortunately does not always afford the s
tudents the luxuries of their peers in other places. \r\n   My student
s may lack many of the modern comforts enjoyed by others their age. The
y may not have the newest clothes, shoes or technology but it does not
dampen their spirit. They have a wonderful zest for learning and an inn
ate drive. That is why it is so vital, that they see a small recognitio
n of their effort. So we may cultivate this spirit and energy to propel
them forward.Equipping my students with new sneakers for the basketball
season will have a huge impact in terms of functionality and confidenc
e. \r\n   The students I work with put in tremendous effort, but often
find themselves embarrassed when they step on the court. Many do not ha
ve money for new sneakers or basketball sneakers at all. We borrow, len
d and make do.\r\n   With the benefit of new sneakers for the team th
e students will perform better (cutting, running, jumping), but more im
portantly they will feel better about themselves. They will hold their
heads high every time they compete. \r\n   The proper basketball sn
eaker is not just an issue of cosmetics. The sneakers we have selected
will provide the proper support to help avoid foot, ankle and knee inju
ries. My students should not have to compete with a higher injury risk
just because they are less well of financially.\r\n   Teenagers very
much associate how they act and feel with how they look. They can also

```
at times be cruel to each other. When my students step on the court I w
ant them to be focused on doing their best not worried about the ridicu
le and self doubt that can come from having old, beat up sneakers.  \r
\n\r\n    \r\n  \r\nnannan
==================================================
Our school is filled with the laughter and smiles of our beautiful elem
entary students. But there are some children, our exceptional students,
who face many obstacles and challenges at school and in their communiti
es.  This wonderful group of exceptional children won't let anything ge
t in their way of learning and fun.\r\n\r\nOur ESE unit is composed of
the most amazing kindergarten through 5th grade exceptional education s
tudents, many who are on the Autism Spectrum or must contend with vario
us exceptionalities. \r\n\r\nAs part of a title I school, many of these
students' families have great financial needs.  Our students are in nee
d of basic services and qualify for the free lunch program.  Recognizin
g these needs, our teachers turn to donors like you to help us provide
basic supplies for our children.  Every donation directly helps these s
tudents!As a Title I school, many of our students families struggle eac
h day to provide the necessary home and school supplies our students ne
ed.  Children use and abuse some items they own especially backpacks.
Our students need new backpacks to replace backpacks that may be too sm
all, damaged or lost.  \r\n\r\nHave you experienced how heavy the books
students take home are?  Backpacks are getting heavier and heavier.  St
udents need sturdy and strong backpacks to help with the load they carr
y.  By providing them a replacement backpack we not only help their fam
ilies out we are saving our students' backs!  Students need appropriate
size backpacks to prevent injury to their backsides.  We appreciate any
assistance our amazing donors can provide.nannan
==================================================
```

In [21]:

```
# 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 [22]:

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

```
Our school is filled with the laughter and smiles of our beautiful elem
entary students. But there are some children, our exceptional students,
who face many obstacles and challenges at school and in their communiti
es.  This wonderful group of exceptional children will not let anything
get in their way of learning and fun.\r\n\r\nOur ESE unit is composed o
f the most amazing kindergarten through 5th grade exceptional education
students, many who are on the Autism Spectrum or must contend with vari
ous exceptionalities. \r\n\r\nAs part of a title I school, many of thes
e students' families have great financial needs.  Our students are in n
eed of basic services and qualify for the free lunch program.  Recogniz
ing these needs, our teachers turn to donors like you to help us provid
e basic supplies for our children.  Every donation directly helps these
students!As a Title I school, many of our students families struggle ea
ch day to provide the necessary home and school supplies our students n
eed.  Children use and abuse some items they own especially backpacks.
Our students need new backpacks to replace backpacks that may be too sm
all, damaged or lost.  \r\n\r\nHave you experienced how heavy the books
students take home are?  Backpacks are getting heavier and heavier.  St
udents need sturdy and strong backpacks to help with the load they carr
y.  By providing them a replacement backpack we not only help their fam
ilies out we are saving our students' backs!  Students need appropriate
size backpacks to prevent injury to their backsides.  We appreciate any
assistance our amazing donors can provide.nannan
==================================================
```

In [23]:
```
# \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)
```

```
Our school is filled with the laughter and smiles of our beautiful elem
entary students. But there are some children, our exceptional students,
who face many obstacles and challenges at school and in their communiti
es.  This wonderful group of exceptional children will not let anything
get in their way of learning and fun.    Our ESE unit is composed of th
e most amazing kindergarten through 5th grade exceptional education stu
dents, many who are on the Autism Spectrum or must contend with various
exceptionalities.     As part of a title I school, many of these studen
ts' families have great financial needs.  Our students are in need of b
asic services and qualify for the free lunch program.  Recognizing thes
e needs, our teachers turn to donors like you to help us provide basic
supplies for our children.  Every donation directly helps these student
s!As a Title I school, many of our students families struggle each day
to provide the necessary home and school supplies our students need.  C
hildren use and abuse some items they own especially backpacks.  Our st
udents need new backpacks to replace backpacks that may be too small, d
amaged or lost.      Have you experienced how heavy the books students
take home are?  Backpacks are getting heavier and heavier.  Students ne
ed sturdy and strong backpacks to help with the load they carry.  By pr
oviding them a replacement backpack we not only help their families out
we are saving our students' backs!  Students need appropriate size back
packs to prevent injury to their backsides.  We appreciate any assistan
ce our amazing donors can provide.nannan
```

In [24]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

```
Our school is filled with the laughter and smiles of our beautiful elem
entary students But there are some children our exceptional students wh
o face many obstacles and challenges at school and in their communities
This wonderful group of exceptional children will not let anything get
```

```
in their way of learning and fun Our ESE unit is composed of the most a
mazing kindergarten through 5th grade exceptional education students ma
ny who are on the Autism Spectrum or must contend with various exceptio
nalities As part of a title I school many of these students families ha
ve great financial needs Our students are in need of basic services and
qualify for the free lunch program Recognizing these needs our teachers
turn to donors like you to help us provide basic supplies for our child
ren Every donation directly helps these students As a Title I school ma
ny of our students families struggle each day to provide the necessary
home and school supplies our students need Children use and abuse some
items they own especially backpacks Our students need new backpacks to
replace backpacks that may be too small damaged or lost Have you experi
enced how heavy the books students take home are Backpacks are getting
heavier and heavier Students need sturdy and strong backpacks to help w
ith the load they carry By providing them a replacement backpack we not
only help their families out we are saving our students backs Students
need appropriate size backpacks to prevent injury to their backsides We
appreciate any assistance our amazing donors can provide nannan
```

In [25]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'no
t'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves'
, 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselve
s', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'it
s', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'th
is', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'h
ave', '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', 'h
```

```
ow', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 's
o', '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', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "is
n't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',
 "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [26]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwor
ds)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████
██████████| 32414/32414 [00:38<00:00, 835.55it/s]
```

In [27]:
```
# after preprocesing
preprocessed_essays[20000]
```

Out[27]:
```
'school filled laughter smiles beautiful elementary students children e
xceptional students face many obstacles challenges school communities w
onderful group exceptional children not let anything get way learning f
un ese unit composed amazing kindergarten 5th grade exceptional educati
on students many autism spectrum must contend various exceptionalities
```

```
part title school many students families great financial needs students
need basic services qualify free lunch program recognizing needs teache
rs turn donors like help us provide basic supplies children every donat
ion directly helps students title school many students families struggl
e day provide necessary home school supplies students need children use
abuse items especially backpacks students need new backpacks replace ba
ckpacks may small damaged lost experienced heavy books students take ho
me backpacks getting heavier heavier students need sturdy strong backpa
cks help load carry providing replacement backpack not help families sa
ving students backs students need appropriate size backpacks prevent in
jury backsides appreciate assistance amazing donors provide nannan'
```

1.4 Preprocessing of `project_title`

In [28]:
```
# printing some project titles.
for i in range (0,21):

    print(project_data['project_title'].values[i])
    print("="*50)
```

```
Math is Fun!
==================================================
Multimedia, Apps, and a Game
==================================================
Colorful Writing
==================================================
Let's Walk a Mile
==================================================
Listen, Listen Who's got the Story?
==================================================
Kinder fun
==================================================
New Volleyballs for our Girls Volleyball Team
==================================================
Classroom Supplies
==================================================
Sharpening to Success!
==================================================
```

```
iPads for my Little Learners
==================================================
Bullying, Boys and Books: A Novel Study Through \"Crash\"
==================================================
Love to Learn, Love to Play
==================================================
Empowering Students Through Art
==================================================
360 Camera
==================================================
Empowering Students Through Art: \"Glass\" Panel Poetry Books
==================================================
Back to Basics
==================================================
Twenty-First Century Learning
==================================================
Writers Wanted!
==================================================
***Just BOOKS***
==================================================
Comfy Cozy Cooperative learning
==================================================
Hot New Books for the END of the School Year!!!!
==================================================
```

In [29]:

```
preprocessed_titles = []

for dataset in tqdm(project_data['project_title'].values):
    data = decontracted(dataset)  # Replacing some specific and general
 short form into proper word/stopword.
    data = re.sub(r"it's", "it is", data) # Replacing it's with it is a
s it is not part of function decontracted
    data = data.replace('\\r', ' ') # Replacing \r with space
    data = data.replace('\\"', ' ') # Replacing \ with space
    data = data.replace('\\n', ' ') # Replacing \n with space
    data = re.sub('[^A-Za-z0-9]+', ' ', data) # Replacing special chara
cters with space
    data = re.sub("\S*\d\S*", "", data).strip() # Trimming numbers cont
aining digits
```

```
    data = ' '.join(e for e in data.split() if e not in stopwords) # Re
moving stopwords
    preprocessed_titles.append(data.lower().strip()) # Creating array i
n all the lower cases.
```

```
100%|██████████████████████████████████████████████████████████
███████| 32414/32414 [00:02<00:00, 16162.39it/s]
```

In [30]:
```
for i in range (0,21):
    print(preprocessed_titles[i])
    print("="*50)
```

```
math fun
==================================================
multimedia apps game
==================================================
colorful writing
==================================================
let walk mile
==================================================
listen listen who got story
==================================================
kinder fun
==================================================
new volleyballs girls volleyball team
==================================================
classroom supplies
==================================================
sharpening success
==================================================
ipads little learners
==================================================
bullying boys books a novel study through crash
==================================================
love learn love play
==================================================
empowering students through art
==================================================
camera
```

```
==================================================
empowering students through art glass panel poetry books
==================================================
back basics
==================================================
twenty first century learning
==================================================
writers wanted
==================================================
just books
==================================================
comfy cozy cooperative learning
==================================================
hot new books end school year
==================================================
```

Observation: As we can see after preprocessing data do not have any special characters, symbols, stopwords and all the words are in lowercase

1.5 Preparing data for models

In [31]:
```
project_data.columns
```

Out[31]:
```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_stat
e',
       'Date', 'project_grade_category', 'project_title', 'project_essa
y_1',
       'project_essay_2', 'project_essay_3', 'project_essay_4',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_appr
oved',
       'clean_categories', 'clean_subcategories', 'essay'],
      dtype='object')
```

we are going to consider

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

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

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

1.5.1 Vectorizing Categorical data

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

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)
categories_one_hot = vectorizer.fit_transform(project_data['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",categories_one_hot.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig  (32414, 9)
```

In [33]:
```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys
()), lowercase=False, binary=True)
sub_categories_one_hot = vectorizer.fit_transform(project_data['clean_s
ubcategories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",sub_categories_one_hot.s
hape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolveme
nt', 'Extracurricular', 'Civics_Government', 'ForeignLanguages', 'Nutri
tionEducation', 'Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingA
rts', 'CharacterEducation', 'TeamSports', 'Other', 'College_CareerPre
p', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopme
nt', 'ESL', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Healt
h_Wellness', 'AppliedSciences', 'SpecialNeeds', 'Literature_Writing',
'Mathematics', 'Literacy']
Shape of matrix after one hot encodig  (32414, 30)
```

In [34]:
```
school_state_vectorizer = CountVectorizer(lowercase=False, binary=True)
school_state_vectorizer.fit(project_data['school_state'].values)
print(school_state_vectorizer.get_feature_names())

school_state_one_hot = school_state_vectorizer.transform(project_data[
'school_state'].values)
print("Shape of matrix after one hot encodig ",school_state_one_hot.sha
pe)
print("the type of count vectorizer ",type(school_state_one_hot))
```

```
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'H
I', 'IA', 'ID', 'IL', 'IN', 'KS', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI',
'MN', 'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM', 'NV', 'NY',
'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT',
'WA', 'WI', 'WV', 'WY']
Shape of matrix after one hot encodig  (32414, 51)
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
```

In [35]:
```
# https://www.geeksforgeeks.org/python-pandas-dataframe-fillna-to-repla
ce-null-values-in-dataframe/
```

```
project_data["teacher_prefix"].fillna("No_Prefix", inplace = True)

teacher_prefix_vectorizer = CountVectorizer(lowercase=False, binary=Tru
e)
teacher_prefix_vectorizer.fit(project_data['teacher_prefix'].values)
print(teacher_prefix_vectorizer.get_feature_names())

teacher_prefix_one_hot = teacher_prefix_vectorizer.transform(project_da
ta['teacher_prefix'].values)
print("Shape of matrix after one hot encodig ",teacher_prefix_one_hot.s
hape)
```

```
['Mr', 'Mrs', 'Ms', 'No_Prefix', 'Teacher']
Shape of matrix after one hot encodig  (32414, 5)
```

In [47]:

```
#new_grade_cat = []

#for i in range(len(project_data)):

my_grade_counter = Counter()

for project_grade in project_data['project_grade_category'].values:

    if (' ' in project_grade):

        project_grade = project_grade.replace(" ", "~")


    my_grade_counter.update(project_grade.split())


project_grade_cat_dict = dict(my_grade_counter)
sorted_project_grade_cat_dict = dict(sorted(project_grade_cat_dict.item
s(), key=lambda kv: kv[1]))

grade_cat_vectorizer = CountVectorizer(vocabulary=list(sorted_project_g
rade_cat_dict.keys()), lowercase=False, binary=True)
grade_cat_vectorizer.fit(project_data['project_grade_category'].values)
print(grade_cat_vectorizer.get_feature_names())
```

```
grade_cat_one_hot = grade_cat_vectorizer.transform(project_data['project_grade_category'].values)
print("Shape of matrix after one hot encodig ",grade_cat_one_hot.shape)
```

```
['Grades~9-12', 'Grades~6-8', 'Grades~3-5', 'Grades~PreK-2']
Shape of matrix after one hot encodig  (32414, 4)
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [48]:
```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10)
text_bow = vectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",text_bow.shape)
```

```
Shape of matrix after one hot encodig  (32414, 10197)
```

In [49]:
```
titles_vectorizer = CountVectorizer(min_df=10)
titles_bow = titles_vectorizer.fit_transform(preprocessed_titles)
print("some sample features(unique words in the corpus)",titles_vectorizer.get_feature_names()[0:10])
print("Shape of matrix after one hot encodig ",titles_bow.shape)
print("the type of count vectorizer ",type(titles_bow))
print("the number of unique words ", titles_bow.get_shape()[1])
```

```
some sample features(unique words in the corpus) ['abc', 'about', 'academic', 'access', 'accessible', 'accessing', 'accessories', 'achieve', 'achievement', 'achieving']
Shape of matrix after one hot encodig  (32414, 1601)
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the number of unique words  1601
```

1.5.2.2 TFIDF vectorizer

In [50]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
text_tfidf = vectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",text_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (32414, 10197)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [51]:
```
'''
# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print ("Done.",len(model)," words loaded!")
    return model
model = loadGloveModel('glove.42B.300d.txt')

# ==============================
Output:

Loading Glove Model
1917495it [06:32, 4879.69it/s]
Done. 1917495  words loaded!

# ==============================

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

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

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

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


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

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


'''
```

Out[51]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230
349/4084039\ndef loadGloveModel(gloveFile):\n    print ("Loading Glove
Model")\n    f = open(gloveFile,\'r\', encoding="utf8")\n    model = {}
\n    for line in tqdm(f):\n        splitLine = line.split()\n        w
ord = splitLine[0]\n        embedding = np.array([float(val) for val in
splitLine[1:]])\n        model[word] = embedding\n    print ("Done.",le
n(model)," words loaded!")\n    return model\nmodel = loadGloveModel
(\'glove.42B.300d.txt\')\n\n# ============================\nOutput:\n
  \nLoading Glove Model\n1917495it [06:32, 4879.69it/s]\nDone. 1917495
```

```
words loaded!\n\n# ============================\n\nwords = []\nfor i in
preproced_texts:\n    words.extend(i.split(\' \'))\n\nfor i in preproce
d_titles:\n    words.extend(i.split(\' \'))\nprint("all the words in th
e coupus", len(words))\nwords = set(words)\nprint("the unique words in
the coupus", len(words))\n\ninter_words = set(model.keys()).intersectio
n(words)\nprint("The number of words that are present in both glove vec
tors and our coupus",       len(inter_words),"(",np.round(len(inter_wor
ds)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nwords_glove = set(mo
del.keys())\nfor i in words:\n    if i in words_glove:\n        words_c
ourpus[i] = model[i]\nprint("word 2 vec length", len(words_courpus))\n
\n\n# stronging variables into pickle files python: http://www.jessicay
ung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimpo
rt pickle\nwith open(\'glove_vectors\', \'wb\') as f:\n    pickle.dump
(words_courpus, f)\n\n\n'
```

In [52]:
```
# stronging variables into pickle files python: http://www.jessicayung.
com/how-to-use-pickle-to-save-and-load-variables-in-python/
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [53]:
```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored
 in this list
for sentence in tqdm(preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    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.append(vector)
```

```
print(len(avg_w2v_vectors))
print(len(avg_w2v_vectors[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 32414/32414 [00:07<00:00, 4083.81it/s]
```

```
32414
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [54]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_essays)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model
.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [55]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is store
d in this list
for sentence in tqdm(preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentenc
e/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 t
he tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentenc
e.split())) # getting the tfidf 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.append(vector)

print(len(tfidf_w2v_vectors))
print(len(tfidf_w2v_vectors[0]))
```

```
100%|██████████| 32414/32414 [00:58<00:00, 558.23it/s]
```

```
32414
300
```

In [56]:

```
# TFIDF on project titles
titles_tfidf_vectorizer = TfidfVectorizer(min_df=10)
titles_tfidf = titles_tfidf_vectorizer.fit_transform(preprocessed_title
s)
print("some sample features(unique words in the corpus)",titles_tfidf_v
ectorizer.get_feature_names()[10:21])
print("Shape of matrix after one hot encodig ",titles_tfidf.shape)
```

```
some sample features(unique words in the corpus) ['across', 'act', 'act
ion', 'active', 'activities', 'activity', 'add', 'adding', 'adventure',
'adventures', 'after']
Shape of matrix after one hot encodig  (32414, 1601)
```

In [57]:

```
# AVG W2V on project title
avg_w2v_titles_vectors = [];

for sentence in tqdm(preprocessed_titles):

    vector_titles = np.zeros(300)
    cnt_words_titles = 0;

    for word in sentence.split():

        if word in glove_words:

            vector += model[word]
            cnt_words_titles += 1
```

```
    if cnt_words_titles != 0:

        vector_titles /= cnt_words_titles

    avg_w2v_titles_vectors.append(vector_titles)

print(len(avg_w2v_titles_vectors))
print(len(avg_w2v_titles_vectors[0]))
```

```
100%|██████████| 32414/32414 [00:00<00:00, 71155.27it/s]
```

```
32414
300
```

In [58]:
```
# TFIDF weighted W2V on project_title
titles_tfidf_model = TfidfVectorizer()
titles_tfidf_model.fit(preprocessed_titles)
titles_dictionary = dict(zip(titles_tfidf_model.get_feature_names(), li
st(titles_tfidf_model.idf_)))
titles_tfidf_words = set(titles_tfidf_model.get_feature_names())
```

In [59]:
```
titles_tfidf_w2v_vectors = [];

for titles_sentence in tqdm(preprocessed_titles):

    titles_vector = np.zeros(300)
    titles_tfidf_weight = 0;

    for word in titles_sentence.split():

        if (word in glove_words) and (word in titles_tfidf_words):

            titles_vec = model[word]

            titles_tf_idf = titles_dictionary[word]*(titles_sentence.co
unt(word)/len(titles_sentence.split()))
            titles_vector += (titles_vec * titles_tf_idf)
            titles_tfidf_weight += titles_tf_idf
```

```
    if titles_tfidf_weight != 0:

        titles_vector /= titles_tfidf_weight

    titles_tfidf_w2v_vectors.append(titles_vector)

print(len(titles_tfidf_w2v_vectors))
print(len(titles_tfidf_w2v_vectors[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████| 32414/32414 [00:00<00:00, 35081.83it/s]
```

```
32414
300
```

1.5.3 Vectorizing Numerical features

In [60]:
```
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 [61]:
```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/gene
rated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 21
3.03 329.   ... 399.   287.73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)

price_scalar = StandardScaler()
price_scalar.fit(project_data['price'].values.reshape(-1,1)) # finding
 the mean and standard deviation of this data
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(p
rice_scalar.var_[0])}")
```

```
# Now standardize the data with above maen and variance.
price_standardized = price_scalar.transform(project_data['price'].value
s.reshape(-1, 1))
```

```
Mean : 51.22791725797495, Standard deviation : 149.72149781041014
```

In [62]:
```
price_standardized
```

Out[62]:
```
array([[-0.27543084],
       [-0.30882617],
       [-0.19407979],
       ...,
       [-0.20890732],
       [-0.13343386],
       [-0.29880757]])
```

1.5.4 Merging all the above features

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

In [63]:
```
print(categories_one_hot.shape)
print(sub_categories_one_hot.shape)
print(text_bow.shape)
print(price_standardized.shape)
```

```
(32414, 9)
(32414, 30)
(32414, 10197)
(32414, 1)
```

In [64]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/40840
39
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix an
d a dense matirx :)
X = hstack((categories_one_hot, sub_categories_one_hot, text_bow, price
```

```
_standardized))
X.shape
```

Out[64]: (32414, 10237)

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_d
igits
from sklearn.feature_selection impo
rt SelectKBest, chi2
X, y = load_digits(return_X_y=True)
X.shape
X_new = SelectKBest(chi2, k=20).fit
_transform(X, y)
X_new.shape
========
output:
(1797, 64)
(1797, 20)
```

- Repeat the steps 2 and 3 on the data matrix after feature selection

5. **Conclusion**

- You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage

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

2. K Nearest Neighbor

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

In [65]:
```
approved_project = project_data['project_is_approved'].values
project_data.drop(['project_is_approved'], axis=1, inplace=True)
project_data.head(1)
```

Out[65]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
0	47750	p185738	3afe10b996b7646d8641985a4b4b570d	Mrs.	UT

In [66]:
```
# Data splitting

from sklearn.model_selection import train_test_split

# Splitting in train and test
X_train, X_test, y_train, y_test = train_test_split(project_data, appro
ved_project, test_size=0.33, stratify=approved_project)
```

```
# Splitting in Train Test and Cross Validation
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_
size=0.33, stratify=y_train)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [67]:
```
# Vectorizing Categories on Train, Test and CV data
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), l
owercase=False, binary=True)

# Fit only to train data
vectorizer.fit(X_train['clean_categories'].values)

# Transform to train,test and CV data
X_Train_categories_one_hot = vectorizer.transform(X_train['clean_catego
ries'].values)
X_Test_categories_one_hot = vectorizer.transform(X_test['clean_categori
es'].values)
X_CV_categories_one_hot = vectorizer.transform(X_cv['clean_categories']
.values)

print("Shape of train matrix after one hot encodig ",X_Train_categories
_one_hot.shape)
print("Shape of test matrix after one hot encodig ",X_Test_categories_o
ne_hot.shape)
print("Shape of cv matrix after one hot encodig ",X_CV_categories_one_h
ot.shape)
```

```
Shape of train matrix after one hot encodig  (14550, 9)
Shape of test matrix after one hot encodig  (10697, 9)
Shape of cv matrix after one hot encodig  (7167, 9)
```

In [68]:
```
# Vectorizing subcategories on train, test and cv
```

```
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys
()), lowercase=False, binary=True)
vectorizer.fit(X_train['clean_subcategories'].values)

X_Train_sub_categories_one_hot = vectorizer.transform(X_train['clean_su
bcategories'].values)
X_Test_sub_categories_one_hot = vectorizer.transform(X_test['clean_subc
ategories'].values)
X_CV_sub_categories_one_hot = vectorizer.transform(X_cv['clean_subcateg
ories'].values)

print("Shape of train matrix after one hot encodig ",X_Train_sub_catego
ries_one_hot.shape)
print("Shape of test matrix after one hot encodig ",X_Test_sub_categori
es_one_hot.shape)
print("Shape of cv matrix after one hot encodig ",X_CV_sub_categories_o
ne_hot.shape)
```

```
Shape of train matrix after one hot encodig  (14550, 30)
Shape of test matrix after one hot encodig  (10697, 30)
Shape of cv matrix after one hot encodig  (7167, 30)
```

In [69]:

```
# Vectorizing school state on train , test and cv

school_state_vectorizer = CountVectorizer(lowercase=False, binary=True)

school_state_vectorizer.fit(X_train['school_state'].values)
print(school_state_vectorizer.get_feature_names())

X_Train_school_state_one_hot = school_state_vectorizer.transform(X_trai
n['school_state'].values)
X_Test_school_state_one_hot = school_state_vectorizer.transform(X_test[
'school_state'].values)
X_CV_school_state_one_hot = school_state_vectorizer.transform(X_cv['sch
ool_state'].values)

print("Shape of train matrix after one hot encodig ",X_Train_school_sta
te_one_hot.shape)
```

```
print("Shape of test matrix after one hot encodig ",X_Test_school_state
_one_hot.shape)
print("Shape of cv matrix after one hot encodig ",X_CV_school_state_one
_hot.shape)

print("the type of count vectorizer ",type(X_Train_school_state_one_hot
))
print("the type of count vectorizer ",type(X_Test_school_state_one_hot
))
print("the type of count vectorizer ",type(X_CV_school_state_one_hot))
```

```
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'H
I', 'IA', 'ID', 'IL', 'IN', 'KS', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI',
'MN', 'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM', 'NV', 'NY',
'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT',
'WA', 'WI', 'WV', 'WY']
Shape of train matrix after one hot encodig  (14550, 51)
Shape of test matrix after one hot encodig  (10697, 51)
Shape of cv matrix after one hot encodig  (7167, 51)
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
```

In [70]:

```
# Vectorizing teacher prefix on train , test and cv

project_data["teacher_prefix"].fillna("No_Prefix", inplace = True)

teacher_prefix_vectorizer = CountVectorizer(lowercase=False, binary=Tru
e)
teacher_prefix_vectorizer.fit(X_train['teacher_prefix'].values)

print(teacher_prefix_vectorizer.get_feature_names())

X_Train_teacher_prefix_one_hot = teacher_prefix_vectorizer.transform(X_
train['teacher_prefix'].values)
X_Test_teacher_prefix_one_hot = teacher_prefix_vectorizer.transform(X_t
est['teacher_prefix'].values)
X_CV_teacher_prefix_one_hot = teacher_prefix_vectorizer.transform(X_cv[
'teacher_prefix'].values)
```

```
print("Shape of train matrix after one hot encodig ",X_Train_teacher_prefix_one_hot.shape)
print("Shape of test matrix after one hot encodig ",X_Test_teacher_prefix_one_hot.shape)
print("Shape of cv matrix after one hot encodig ",X_CV_teacher_prefix_one_hot.shape)
```

```
['Mr', 'Mrs', 'Ms', 'Teacher']
Shape of train matrix after one hot encodig  (14550, 4)
Shape of test matrix after one hot encodig  (10697, 4)
Shape of cv matrix after one hot encodig  (7167, 4)
```

In [73]:

```
# Vectorizing grade category on train , test and cv

my_grade_counter = Counter()

for project_grade in project_data['project_grade_category'].values:

    if (' ' in project_grade):

        project_grade = project_grade.replace(" ", "~")

    my_grade_counter.update(project_grade.split())

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

grade_cat_vectorizer = CountVectorizer(vocabulary=list(sorted_project_grade_cat_dict.keys()), lowercase=False, binary=True)
grade_cat_vectorizer.fit(X_train['project_grade_category'].values)
print(grade_cat_vectorizer.get_feature_names())

X_Train_grade_cat_one_hot = grade_cat_vectorizer.transform(X_train['project_grade_category'].values)
X_Test_grade_cat_one_hot = grade_cat_vectorizer.transform(X_test['project_grade_category'].values)
X_CV_grade_cat_one_hot = grade_cat_vectorizer.transform(X_cv['project_g
```

```
rade_category'].values)

print("Shape of train matrix after one hot encodig ",X_Train_grade_cat_
one_hot.shape)
print("Shape of test matrix after one hot encodig ",X_Test_grade_cat_on
e_hot.shape)
print("Shape of cv matrix after one hot encodig ",X_CV_grade_cat_one_ho
t.shape)
```

```
['Grades~9-12', 'Grades~6-8', 'Grades~3-5', 'Grades~PreK-2']
Shape of train matrix after one hot encodig  (14550, 4)
Shape of test matrix after one hot encodig  (10697, 4)
Shape of cv matrix after one hot encodig  (7167, 4)
```

2.3 Make Data Model Ready: encoding eassay, and project_title

In [74]:
```
# merge two column text dataframe:
X_train["essay"] = X_train["project_essay_1"].map(str) +\
                        X_train["project_essay_2"].map(str) + \
                        X_train["project_essay_3"].map(str) + \
                        X_train["project_essay_4"].map(str)
```

In [75]:
```
# preprocessing essay train data
from tqdm import tqdm
X_Train_preprocessed_essays = []
# tqdm is for printing the status bar
for sentence in tqdm(X_train['essay'].values):
    X_Train_essay_sent = decontracted(sentance)
    X_Train_essay_sent = X_Train_essay_sent.replace('\\r', ' ')
    X_Train_essay_sent = X_Train_essay_sent.replace('\\"', ' ')
    X_Train_essay_sent = X_Train_essay_sent.replace('\\n', ' ')
    X_Train_essay_sent = re.sub('[^A-Za-z0-9]+', ' ', X_Train_essay_sen
t)

    X_Train_essay_sent = ' '.join(e for e in X_Train_essay_sent.split()
 if e.lower() not in stopwords)
```

```
    X_Train_preprocessed_essays.append(X_Train_essay_sent.lower().strip
())
```

```
100%|████████████████████████████████████████████████████████████████
████████| 14550/14550 [00:06<00:00, 2191.49it/s]
```

In [76]:
```
# preprocessing essay test data
from tqdm import tqdm
X_Test_preprocessed_essays = []
# tqdm is for printing the status bar
for sentence in tqdm(X_test['essay'].values):
    X_Test_essay_sent = decontracted(sentence)
    X_Test_essay_sent = X_Test_essay_sent.replace('\\r', ' ')
    X_Test_essay_sent = X_Test_essay_sent.replace('\\"', ' ')
    X_Test_essay_sent = X_Test_essay_sent.replace('\\n', ' ')
    X_Test_essay_sent = re.sub('[^A-Za-z0-9]+', ' ', X_Test_essay_sent)

    X_Test_essay_sent = ' '.join(e for e in X_Test_essay_sent.split() i
f e.lower() not in stopwords)
    X_Test_preprocessed_essays.append(X_Test_essay_sent.lower().strip
())
```

```
100%|████████████████████████████████████████████████████████████████
████████| 10697/10697 [00:05<00:00, 1939.05it/s]
```

In [77]:
```
# preprocessing essay cv data
from tqdm import tqdm
X_CV_preprocessed_essays = []
# tqdm is for printing the status bar
for sentence in tqdm(X_cv['essay'].values):
    X_CV_essay_sent = decontracted(sentence)
    X_CV_essay_sent = X_CV_essay_sent.replace('\\r', ' ')
    X_CV_essay_sent = X_CV_essay_sent.replace('\\"', ' ')
    X_CV_essay_sent = X_CV_essay_sent.replace('\\n', ' ')
    X_CV_essay_sent = re.sub('[^A-Za-z0-9]+', ' ', X_CV_essay_sent)

    X_CV_essay_sent = ' '.join(e for e in X_CV_essay_sent.split() if e.
lower() not in stopwords)
    X_CV_preprocessed_essays.append(X_CV_essay_sent.lower().strip())
```

```
100%|██████████| 7167/7167 [00:03<00:00, 1918.80it/s]
```

In [78]:
```
# preprocessing project title train data
X_Train_preprocessed_titles = []

for dataset in tqdm(X_train['project_title'].values):
    data = decontracted(dataset)  # Replacing some specific and general
 short form into proper word/stopword.
    data = re.sub(r"it's", "it is", data) # Replacing it's with it is a
s it is not part of function decontracted
    data = data.replace('\\r', ' ') # Replacing \r with space
    data = data.replace('\\"', ' ') # Replacing \ with space
    data = data.replace('\\n', ' ') # Replacing \n with space
    data = re.sub('[^A-Za-z0-9]+', ' ', data) # Replacing special chara
cters with space
    data = re.sub("\S*\d\S*", "", data).strip() # Trimming numbers cont
aining digits

    data = ' '.join(e for e in data.split() if e not in stopwords) # Re
moving stopwords
    X_Train_preprocessed_titles.append(data.lower().strip()) # Creating
 array in all the lower cases.
```

```
100%|██████████| 14550/14550 [00:00<00:00, 34522.49it/s]
```

In [79]:
```
# preprocessing project title test data
X_Test_preprocessed_titles = []

for dataset in tqdm(X_test['project_title'].values):
    data = decontracted(dataset)  # Replacing some specific and general
 short form into proper word/stopword.
    data = re.sub(r"it's", "it is", data) # Replacing it's with it is a
s it is not part of function decontracted
    data = data.replace('\\r', ' ') # Replacing \r with space
    data = data.replace('\\"', ' ') # Replacing \ with space
    data = data.replace('\\n', ' ') # Replacing \n with space
    data = re.sub('[^A-Za-z0-9]+', ' ', data) # Replacing special chara
```

```
cters with space
    data = re.sub("\S*\d\S*", "", data).strip() # Trimming numbers cont
aining digits

    data = ' '.join(e for e in data.split() if e not in stopwords) # Re
moving stopwords
    X_Test_preprocessed_titles.append(data.lower().strip()) # Creating
 array in all the lower cases.
```

```
100%|██████████████████████████████████████████████████████████████
██████| 10697/10697 [00:00<00:00, 33252.21it/s]
```

In [80]:
```
# preprocessing project title cv data
X_CV_preprocessed_titles = []

for dataset in tqdm(X_cv['project_title'].values):
    data = decontracted(dataset)  # Replacing some specific and general
 short form into proper word/stopword.
    data = re.sub(r"it's", "it is", data) # Replacing it's with it is a
s it is not part of function decontracted
    data = data.replace('\\r', ' ') # Replacing \r with space
    data = data.replace('\\"', ' ') # Replacing \ with space
    data = data.replace('\\n', ' ') # Replacing \n with space
    data = re.sub('[^A-Za-z0-9]+', ' ', data) # Replacing special chara
cters with space
    data = re.sub("\S*\d\S*", "", data).strip() # Trimming numbers cont
aining digits

    data = ' '.join(e for e in data.split() if e not in stopwords) # Re
moving stopwords
    X_CV_preprocessed_titles.append(data.lower().strip()) # Creating ar
ray in all the lower cases.
```

```
100%|██████████████████████████████████████████████████████████████
████████| 7167/7167 [00:00<00:00, 33731.17it/s]
```

In [81]:
```
# BOW Essay train, test and cv data
# We are considering only the words which appeared in at least 10 docum
ents(rows or projects).
```

```
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(X_Train_preprocessed_essays)

X_Train_essay_bow = vectorizer.transform(X_Train_preprocessed_essays)
X_Test_essay_bow = vectorizer.transform(X_Test_preprocessed_essays)
X_CV_essay_bow = vectorizer.transform(X_CV_preprocessed_essays)

print("Shape of train matrix after one hot encodig ",X_Train_essay_bow.
shape)
print("Shape of test matrix after one hot encodig ",X_Test_essay_bow.sh
ape)
print("Shape of CV matrix after one hot encodig ",X_CV_essay_bow.shape)
```

```
Shape of train matrix after one hot encodig  (14550, 92)
Shape of test matrix after one hot encodig  (10697, 92)
Shape of CV matrix after one hot encodig  (7167, 92)
```

In [82]:

```
# BOW title train,test and cv data

titles_vectorizer = CountVectorizer(min_df=10)
titles_vectorizer.fit(X_Train_preprocessed_titles)

X_Train_titles_bow = titles_vectorizer.transform(X_Train_preprocessed_t
itles)
X_Test_titles_bow = titles_vectorizer.transform(X_Test_preprocessed_tit
les)
X_CV_titles_bow = titles_vectorizer.transform(X_CV_preprocessed_titles)

print("some sample features(unique words in the corpus)",titles_vectori
zer.get_feature_names()[0:10])
print("Shape of train matrix after one hot encodig ",X_Train_titles_bow
.shape)
print("Shape of test matrix after one hot encodig ",X_Test_titles_bow.s
hape)
print("Shape of CV matrix after one hot encodig ",X_CV_titles_bow.shape
)
```

```
some sample features(unique words in the corpus) ['about', 'academic',
'access', 'action', 'active', 'activities', 'activity', 'add', 'addin
g', 'adventure']
```

```
Shape of train matrix after one hot encodig  (14550, 878)
Shape of test matrix after one hot encodig  (10697, 878)
Shape of CV matrix after one hot encodig  (7167, 878)
```

In [83]:
```
#TFIDF essay train,test and cv data

vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_Train_preprocessed_essays)

X_Train_essay_tfidf = vectorizer.transform(X_Train_preprocessed_essays)
X_Test_essay_tfidf = vectorizer.transform(X_Test_preprocessed_essays)
X_CV_essay_tfidf = vectorizer.transform(X_CV_preprocessed_essays)

print("Shape of train matrix after one hot encodig ",X_Train_essay_tfid
f.shape)
print("Shape of test matrix after one hot encodig ",X_Test_essay_tfidf.
shape)
print("Shape of CV matrix after one hot encodig ",X_CV_essay_tfidf.shap
e)
```

```
Shape of train matrix after one hot encodig  (14550, 92)
Shape of test matrix after one hot encodig  (10697, 92)
Shape of CV matrix after one hot encodig  (7167, 92)
```

In [84]:
```
# TFIDF on project titles train,test and cv data

titles_tfidf_vectorizer = TfidfVectorizer(min_df=10)
titles_tfidf_vectorizer.fit(X_Train_preprocessed_titles)

X_Train_titles_tfidf = titles_vectorizer.transform(X_Train_preprocessed
_titles)
X_Test_titles_tfidf = titles_vectorizer.transform(X_Test_preprocessed_t
itles)
X_CV_titles_tfidf = titles_vectorizer.transform(X_CV_preprocessed_title
s)

print("Shape of train matrix after one hot encodig ",X_Train_titles_tfi
df.shape)
print("Shape of test matrix after one hot encodig ",X_Test_titles_tfidf
```

```
.shape)
print("Shape of CV matrix after one hot encodig ",X_CV_titles_tfidf.sha
pe)
```

```
Shape of train matrix after one hot encodig  (14550, 878)
Shape of test matrix after one hot encodig  (10697, 878)
Shape of CV matrix after one hot encodig  (7167, 878)
```

In [85]:
```
# average Word2Vec essay on train
# compute average word2vec for each review.
X_Train_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is
 stored in this list
for sentence in tqdm(X_Train_preprocessed_essays): # for each review/se
ntence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    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
    X_Train_avg_w2v_vectors.append(vector)

print(len(X_Train_avg_w2v_vectors))
print(len(X_Train_avg_w2v_vectors[0]))
```

```
100%|████████████████████████████████████████████████████████████████
████████| 14550/14550 [00:02<00:00, 5691.67it/s]

14550
300
```

In [86]:
```
# average Word2Vec essay on test
# compute average word2vec for each review.
X_Test_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is
 stored in this list
for sentence in tqdm(X_Test_preprocessed_essays): # for each review/sen
tence
```

```
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    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
    X_Test_avg_w2v_vectors.append(vector)

print(len(X_Test_avg_w2v_vectors))
print(len(X_Test_avg_w2v_vectors[0]))
```

```
100%|████████████████████████████████████████████████████████████████
████████| 10697/10697 [00:02<00:00, 4022.94it/s]
```

```
10697
300
```

In [87]:

```
# average Word2Vec essay on cv
# compute average word2vec for each review.
X_CV_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is st
ored in this list
for sentence in tqdm(X_CV_preprocessed_essays): # for each review/sente
nce
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    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
    X_CV_avg_w2v_vectors.append(vector)

print(len(X_CV_avg_w2v_vectors))
print(len(X_CV_avg_w2v_vectors[0]))
```

```
100%|██████████| 7167/7167 [00:01<00:00, 4016.67it/s]
```

```
7167
300
```

In [88]:

```
# AVG W2V on project title train
X_Train_avg_w2v_titles_vectors = [];

for sentence in tqdm(X_Train_preprocessed_titles):

    vector_titles = np.zeros(300)
    cnt_words_titles = 0;

    for word in sentence.split():

        if word in glove_words:

            vector += model[word]
            cnt_words_titles += 1

    if cnt_words_titles != 0:

        vector_titles /= cnt_words_titles

    X_Train_avg_w2v_titles_vectors.append(vector_titles)

print(len(X_Train_avg_w2v_titles_vectors))
print(len(X_Train_avg_w2v_titles_vectors[0]))
```

```
100%|██████████| 14550/14550 [00:00<00:00, 72683.35it/s]
```

```
14550
300
```

In [89]:

```
# AVG W2V on project title test
X_Test_avg_w2v_titles_vectors = [];

for sentence in tqdm(X_Test_preprocessed_titles):
```

```
    vector_titles = np.zeros(300)
    cnt_words_titles = 0;

    for word in sentence.split():

        if word in glove_words:

            vector += model[word]
            cnt_words_titles += 1

    if cnt_words_titles != 0:

        vector_titles /= cnt_words_titles

    X_Test_avg_w2v_titles_vectors.append(vector_titles)

print(len(X_Test_avg_w2v_titles_vectors))
print(len(X_Test_avg_w2v_titles_vectors[0]))
```

```
100%|██████████| 10697/10697 [00:00<00:00, 67882.82it/s]
```

```
10697
300
```

In [90]:

```
# AVG W2V on project title cv
X_CV_avg_w2v_titles_vectors = [];

for sentence in tqdm(X_CV_preprocessed_titles):

    vector_titles = np.zeros(300)
    cnt_words_titles = 0;

    for word in sentence.split():

        if word in glove_words:

            vector += model[word]
            cnt_words_titles += 1
```

```
    if cnt_words_titles != 0:

        vector_titles /= cnt_words_titles

    X_CV_avg_w2v_titles_vectors.append(vector_titles)

print(len(X_CV_avg_w2v_titles_vectors))
print(len(X_CV_avg_w2v_titles_vectors[0]))
```

```
100%|██████████| 7167/7167 [00:00<00:00, 64740.52it/s]
```

```
7167
300
```

In [91]:
```
# TFIDF W2V
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_Train_preprocessed_essays)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model
.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [92]:
```
# TFIDF w2v essay train
# compute average word2vec for each review.
X_Train_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review
 is stored in this list
for sentence in tqdm(X_Train_preprocessed_essays): # for each review/se
ntence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentenc
e/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 t
```

```
he tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentenc
e.split())) # getting the tfidf 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
    X_Train_tfidf_w2v_vectors.append(vector)

print(len(X_Train_tfidf_w2v_vectors))
print(len(X_Train_tfidf_w2v_vectors[0]))
```

```
100%|██████████| 14550/14550 [00:19<00:00, 740.19it/s]
```

```
14550
300
```

In [93]:

```
# TFIDF w2v essay test
# compute average word2vec for each review.
X_Test_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review i
s stored in this list
for sentence in tqdm(X_Test_preprocessed_essays): # for each review/sen
tence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentenc
e/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 t
he tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentenc
e.split())) # getting the tfidf 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
    X_Test_tfidf_w2v_vectors.append(vector)
```

```
print(len(X_Test_tfidf_w2v_vectors))
print(len(X_Test_tfidf_w2v_vectors[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 10697/10697 [00:04<00:00, 2249.37it/s]
```

```
10697
300
```

In [94]:

```
# TFIDF w2v essay cv
# compute average word2vec for each review.
X_CV_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is
 stored in this list
for sentence in tqdm(X_CV_preprocessed_essays): # for each review/sente
nce
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentenc
e/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 t
he tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentenc
e.split())) # getting the tfidf 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
    X_CV_tfidf_w2v_vectors.append(vector)

print(len(X_CV_tfidf_w2v_vectors))
print(len(X_CV_tfidf_w2v_vectors[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 7167/7167 [00:03<00:00, 2114.89it/s]
```

```
7167
300
```

In [95]:
```
# TFIDF weighted W2V on project_title
titles_tfidf_model = TfidfVectorizer()
titles_tfidf_model.fit(X_Train_preprocessed_titles)
titles_dictionary = dict(zip(titles_tfidf_model.get_feature_names(), li
st(titles_tfidf_model.idf_)))
titles_tfidf_words = set(titles_tfidf_model.get_feature_names())
```

In [96]:
```
# TFIDF w2v title train
X_Train_titles_tfidf_w2v_vectors = [];

for titles_sentence in tqdm(X_Train_preprocessed_titles):

    titles_vector = np.zeros(300)
    titles_tfidf_weight = 0;

    for word in titles_sentence.split():

        if (word in glove_words) and (word in titles_tfidf_words):

            titles_vec = model[word]

            titles_tf_idf = titles_dictionary[word]*(titles_sentence.co
unt(word)/len(titles_sentence.split()))
            titles_vector += (titles_vec * titles_tf_idf)
            titles_tfidf_weight += titles_tf_idf

    if titles_tfidf_weight != 0:

        titles_vector /= titles_tfidf_weight

    X_Train_titles_tfidf_w2v_vectors.append(titles_vector)

print(len(X_Train_titles_tfidf_w2v_vectors))
print(len(X_Train_titles_tfidf_w2v_vectors[0]))
```

```
100%|██████████████████████████████████████████████████████████
██████| 14550/14550 [00:00<00:00, 34412.79it/s]

14550
300
```

In [97]:
```
# TFIDF w2v title train
X_Test_titles_tfidf_w2v_vectors = [];

for titles_sentence in tqdm(X_Test_preprocessed_titles):

    titles_vector = np.zeros(300)
    titles_tfidf_weight = 0;

    for word in titles_sentence.split():

        if (word in glove_words) and (word in titles_tfidf_words):

            titles_vec = model[word]

            titles_tf_idf = titles_dictionary[word]*(titles_sentence.co
unt(word)/len(titles_sentence.split()))
            titles_vector += (titles_vec * titles_tf_idf)
            titles_tfidf_weight += titles_tf_idf

    if titles_tfidf_weight != 0:

        titles_vector /= titles_tfidf_weight

    X_Test_titles_tfidf_w2v_vectors.append(titles_vector)

print(len(X_Test_titles_tfidf_w2v_vectors))
print(len(X_Test_titles_tfidf_w2v_vectors[0]))
```

```
100%|██████████| 10697/10697 [00:00<00:00, 36149.78it/s]
```

```
10697
300
```

In [98]:
```
# TFIDF w2v title cv
X_CV_titles_tfidf_w2v_vectors = [];

for titles_sentence in tqdm(X_CV_preprocessed_titles):
```

```
    titles_vector = np.zeros(300)
    titles_tfidf_weight = 0;

    for word in titles_sentence.split():

        if (word in glove_words) and (word in titles_tfidf_words):

            titles_vec = model[word]

            titles_tf_idf = titles_dictionary[word]*(titles_sentence.co
unt(word)/len(titles_sentence.split()))
            titles_vector += (titles_vec * titles_tf_idf)
            titles_tfidf_weight += titles_tf_idf

    if titles_tfidf_weight != 0:

        titles_vector /= titles_tfidf_weight

    X_CV_titles_tfidf_w2v_vectors.append(titles_vector)

print(len(X_CV_titles_tfidf_w2v_vectors))
print(len(X_CV_titles_tfidf_w2v_vectors[0]))
```

```
100%|████████████████████████████████████████████████████████████
████████| 7167/7167 [00:00<00:00, 33500.43it/s]
```

```
7167
300
```

In [102]:
```
# Vectorizing numerical feature

# Merging price data with train, test and cv
X_train = pd.merge(X_train, price_data, on='id', how='left')
X_test = pd.merge(X_test, price_data, on='id', how='left')
X_cv = pd.merge(X_cv, price_data, on='id', how='left')
```

In [103]:
```
# Standardizing price train test and cv data
```

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

X_train_price_norm = normalizer.transform(X_train['price'].values.resha
pe(-1,1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape
(-1,1))
X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(-1,
1))

print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_test_price_norm.shape, y_test.shape)
print(X_cv_price_norm.shape, y_cv.shape)
print("="*100)
```

```
After vectorizations
(14550, 1) (14550,)
(10697, 1) (10697,)
(7167, 1) (7167,)
====================================================================
============================
```

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

Apply KNN on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instructions

In [104]:
```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code

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

2.4.1 Applying KNN brute force on BOW, SET 1

In [105]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

# Train data stack
X_tr = hstack((X_Train_categories_one_hot,X_Train_sub_categories_one_hot,X_Train_school_state_one_hot,
               X_Train_teacher_prefix_one_hot,X_Train_grade_cat_one_hot,X_Train_essay_bow,X_Train_titles_bow,
               X_train_price_norm)).tocsr()

# CV data Stack
X_cr = hstack((X_CV_categories_one_hot,X_CV_sub_categories_one_hot,X_CV_school_state_one_hot,
               X_CV_teacher_prefix_one_hot,X_CV_grade_cat_one_hot,X_CV_essay_bow,X_CV_titles_bow,
               X_cv_price_norm)).tocsr()

# Test Data Stack
X_te = hstack((X_Test_categories_one_hot,X_Test_sub_categories_one_hot,X_Test_school_state_one_hot,
               X_Test_teacher_prefix_one_hot,X_Test_grade_cat_one_hot,X
```

```
_Test_essay_bow,X_Test_titles_bow,
                X_test_price_norm)).tocsr()

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

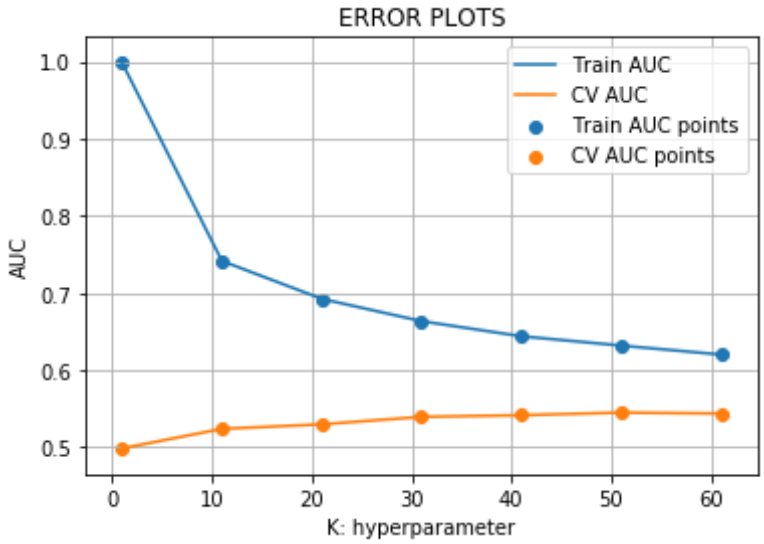
```
Final Data matrix
(14550, 1069) (14550,)
(7167, 1069) (7167,)
(10697, 1069) (10697,)
====================================================================
============================
```

In [106]:
```
def batch_predict(clf, data):

    y_data_pred = []

    # Changing the shape of predicted data in the multiple of 1000
    tr_loop = data.shape[0] - data.shape[0]%1000

    # Running the loop for each 1000th data
    for i in range(0, tr_loop, 1000):

        # Predicting probability
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])

    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [107]:
```
# Plotting error plot, AUC vs K plot to get best K(Bias-Variance trade-
off)

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

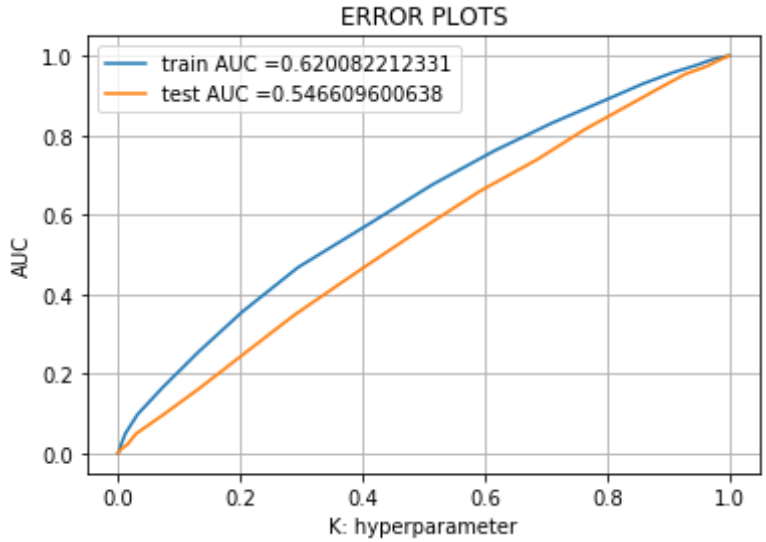
```
train_auc = []
cv_auc = []

# Execute for different K values
K = [1, 11, 21, 31, 41, 51, 61]

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_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probab
ility 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))

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

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

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

```
100%|██████████| 7/7 [07:06<00:00, 60.94s/it]
```



In [108]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=61)
neigh.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilities 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()
```



In [109]:
```
# 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 th
reshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
```

```
            predictions.append(0)
    return predictions
```

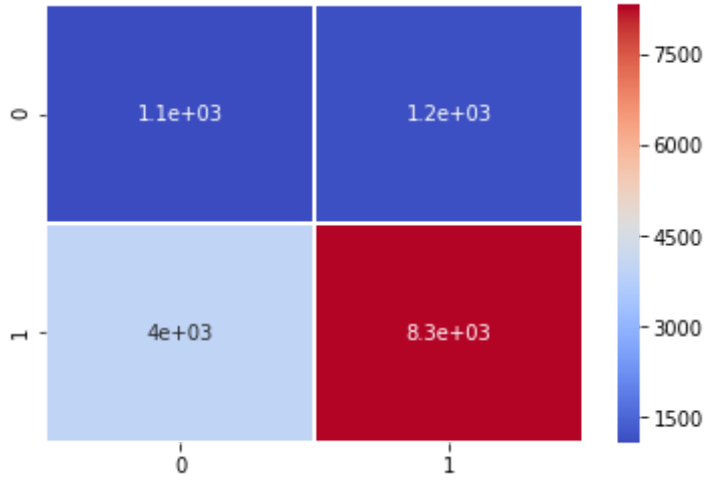
In [110]:
```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, tr
ain_fpr, train_fpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test
_fpr, test_fpr)))
```

```
====================================================================
============================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.249784122005 for threshold 0.82
[[1090 1156]
 [3984 8320]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.249906196319 for threshold 0.82
[[ 674  978]
 [3081 5964]]
```

In [119]:
```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html

# Train Confusion Matrix Heatmap
train_confusion_matrix = confusion_matrix(y_train, predict(y_train_pred
, tr_thresholds, train_fpr, train_fpr))

print("Train Confusion Matrix")
sns.heatmap(train_confusion_matrix,annot=True,linewidth = 0.5, cmap='co
olwarm')
```

```
the maximum value of tpr*(1-fpr) 0.249784122005 for threshold 0.82
Train Confusion Matrix
```

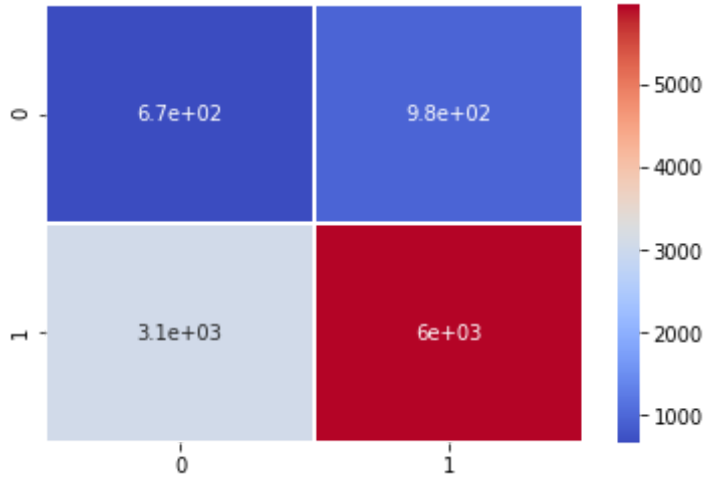
Out[119]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22071c0b358>
```



In [120]:
```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html

# Test Confusion Matrix Heatmap

test_confusion_matrix = confusion_matrix(y_test, predict(y_test_pred, t
r_thresholds, test_fpr, test_fpr))

print("Test Confusion Matrix")
sns.heatmap(test_confusion_matrix,annot=True,linewidth = 0.5, cmap='coo
lwarm')
```

```
the maximum value of tpr*(1-fpr) 0.249906196319 for threshold 0.82
Test Confusion Matrix
```

Out[120]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2206affaba8>
```



2.4.2 Applying KNN brute force on TFIDF, SET 2

In [121]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

# Train data stack
X_tr = hstack((X_Train_categories_one_hot,X_Train_sub_categories_one_hot,X_Train_school_state_one_hot,
               X_Train_teacher_prefix_one_hot,X_Train_grade_cat_one_hot,X_Train_essay_tfidf,X_Train_titles_tfidf,
               X_train_price_norm)).tocsr()

# CV data Stack
X_cr = hstack((X_CV_categories_one_hot,X_CV_sub_categories_one_hot,X_CV_school_state_one_hot,
               X_CV_teacher_prefix_one_hot,X_CV_grade_cat_one_hot,X_CV_essay_tfidf,X_CV_titles_tfidf,
               X_cv_price_norm)).tocsr()

# Test Data Stack
X_te = hstack((X_Test_categories_one_hot,X_Test_sub_categories_one_hot,
```

```
X_Test_school_state_one_hot,
               X_Test_teacher_prefix_one_hot,X_Test_grade_cat_one_hot,X
_Test_essay_tfidf,X_Test_titles_tfidf,
               X_test_price_norm)).tocsr()

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

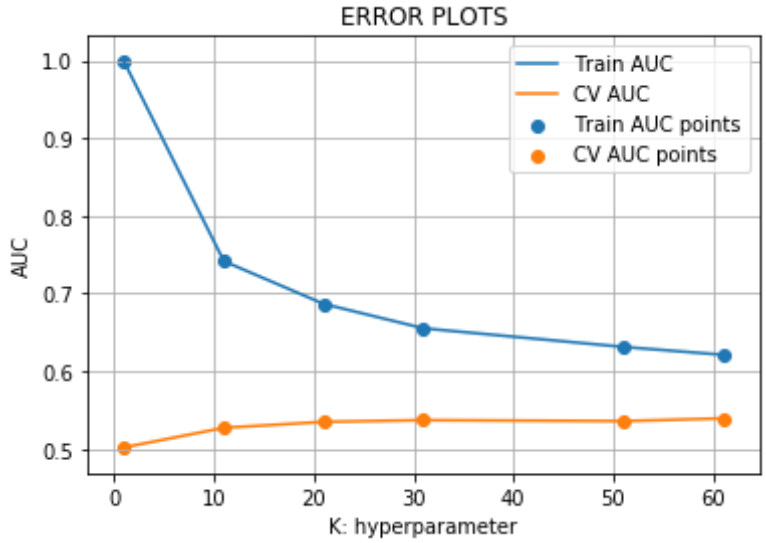
```
Final Data matrix
(14550, 1069) (14550,)
(7167, 1069) (7167,)
(10697, 1069) (10697,)
====================================================================
============================
```

In [122]:

```
# Plotting error plot, AUC vs K plot to get best K(Bias-Variance trade-
off)

train_auc = []
cv_auc = []

# Execute for different K values
K = [1, 11, 21, 31, 51, 61]

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_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probab
ility estimates of the positive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
```

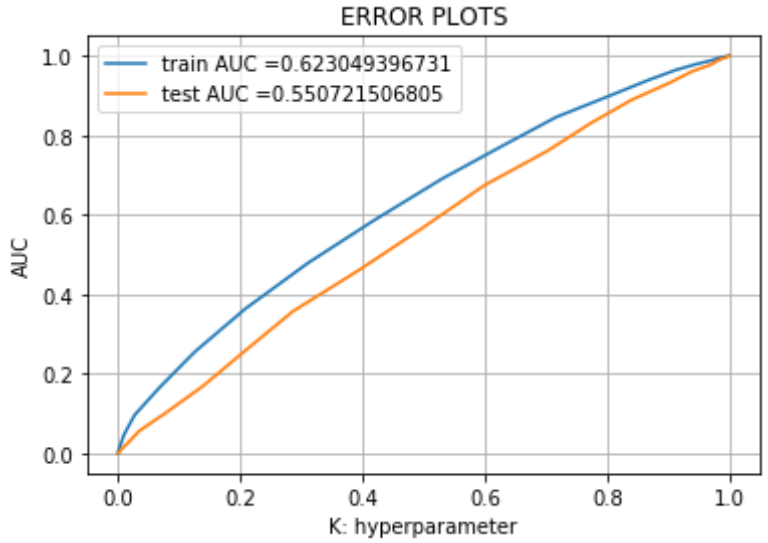
```
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

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

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

```
100%|██████████| 6/6 [12:20<00:00, 123.18s/it]
```



In [123]:
```
neigh = KNeighborsClassifier(n_neighbors=60)
neigh.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y 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, t
rain_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()
```



In [124]:
```
print("="*100)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, tr
```

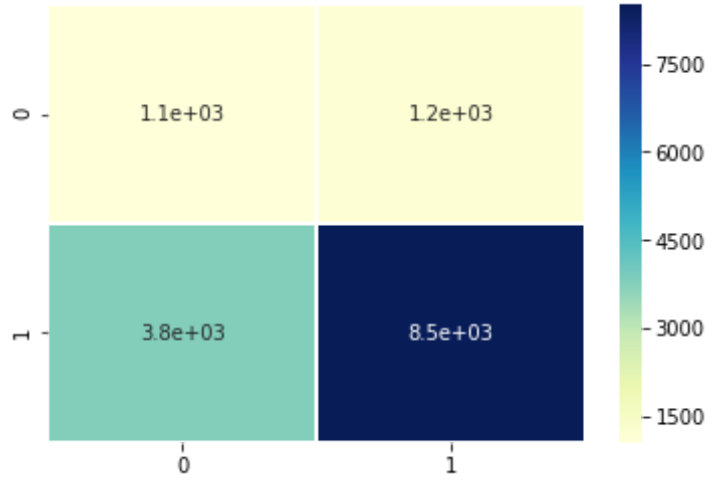
```
ain_fpr, train_fpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test
_fpr, test_fpr)))
```

```
==========================================================================
============================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24908336102 for threshold 0.817
[[1055 1191]
 [3805 8499]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.833
[[ 826  826]
 [3900 5145]]
```

In [126]:
```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html
# Train Confusion Matrix Heatmap

train_confusion_matrix = confusion_matrix(y_train, predict(y_train_pred
, tr_thresholds, train_fpr, train_fpr))

print("Train Confusion Matrix")
sns.heatmap(train_confusion_matrix,annot=True,linewidth = 0.5, cmap='Yl
GnBu')
```

```
the maximum value of tpr*(1-fpr) 0.24908336102 for threshold 0.817
Train Confusion Matrix
```

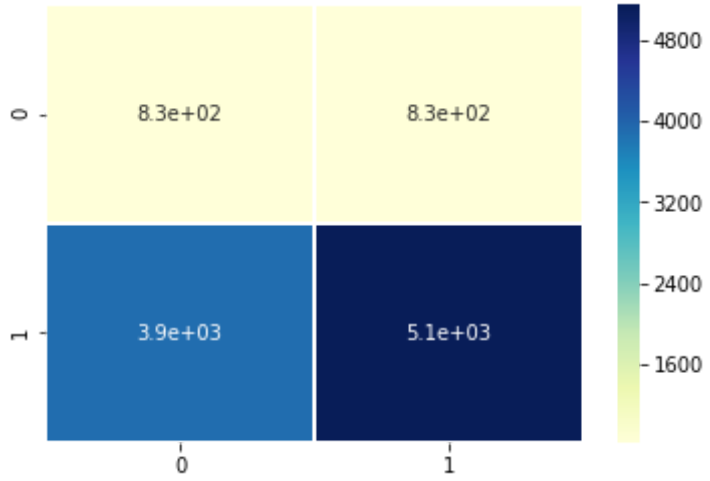
Out[126]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22071bf51d0>
```



In [127]:
```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html

# Test Confusion Matrix Heatmap

test_confusion_matrix = confusion_matrix(y_test, predict(y_test_pred, t
r_thresholds, test_fpr, test_fpr))

print("Test Confusion Matrix")
sns.heatmap(test_confusion_matrix,annot=True,linewidth = 0.5, cmap='YlG
nBu')
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.833
Test Confusion Matrix
```

Out[127]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x220719fc358>
```



2.4.3 Applying KNN brute force on AVG W2V, SET 3

In [128]:
```
# Train data stack
X_tr = hstack((X_Train_categories_one_hot,X_Train_sub_categories_one_ho
t,X_Train_school_state_one_hot,
               X_Train_teacher_prefix_one_hot,X_Train_grade_cat_one_hot
,X_Train_avg_w2v_vectors,X_Train_avg_w2v_titles_vectors,
               X_train_price_norm)).tocsr()

# CV data Stack
X_cr = hstack((X_CV_categories_one_hot,X_CV_sub_categories_one_hot,X_CV
_school_state_one_hot,
               X_CV_teacher_prefix_one_hot,X_CV_grade_cat_one_hot,X_CV_
avg_w2v_vectors,X_CV_avg_w2v_titles_vectors,
               X_cv_price_norm)).tocsr()

# Test Data Stack
X_te = hstack((X_Test_categories_one_hot,X_Test_sub_categories_one_hot,
X_Test_school_state_one_hot,
               X_Test_teacher_prefix_one_hot,X_Test_grade_cat_one_hot,X
_Test_avg_w2v_vectors,X_Test_avg_w2v_titles_vectors,
               X_test_price_norm)).tocsr()
```

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

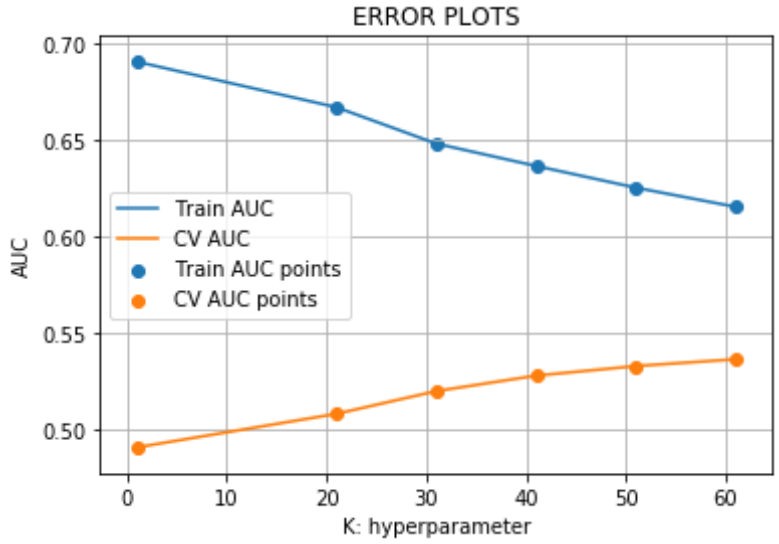
```
Final Data matrix
(14550, 699) (14550,)
(7167, 699) (7167,)
(10697, 699) (10697,)
====================================================================================
==============================
```

In [129]:

```
# Plotting error plot, AUC vs K plot to get best K(Bias-Variance trade-off)

train_auc = []
cv_auc = []

# Execute for different K values
K = [1, 21, 31, 41, 51, 61]

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_cr)

    # 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))

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

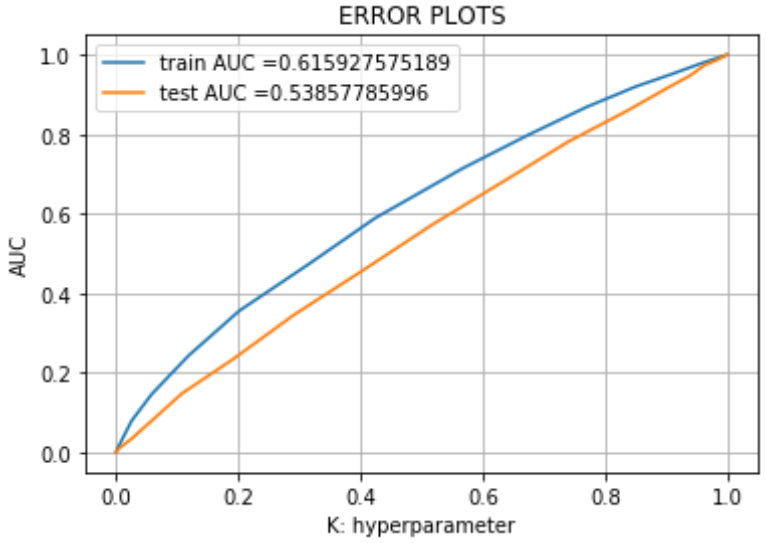
```
plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

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

```
100%|██████████████████████████████████████████████████████████████████████| 6/6 [41:19<00:00, 411.00s/it]
```



In [130]:
```
neigh = KNeighborsClassifier(n_neighbors=60)
neigh.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y 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, t
rain_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()
```



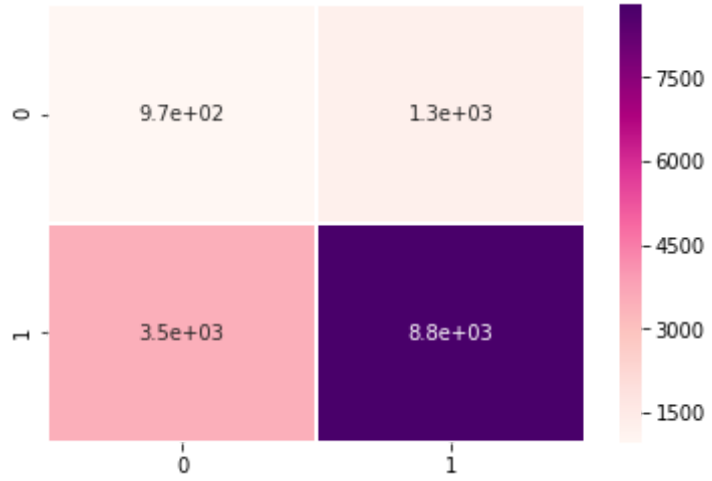
In [131]:
```
print("="*100)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, tr
ain_fpr, train_fpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test
_fpr, test_fpr)))
```

```
================================================================================
==============================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.245359515165 for threshold 0.833
[[ 970 1276]
 [3505 8799]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.249670221435 for threshold 0.833
[[ 590 1062]
 [2806 6239]]
```

In [133]:
```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html
# Train Confusion Matrix Heatmap

train_confusion_matrix = confusion_matrix(y_train, predict(y_train_pred
, tr_thresholds, train_fpr, train_fpr))

print("Train Confusion Matrix")
sns.heatmap(train_confusion_matrix,annot=True,linewidth = 0.5, cmap='Rd
Pu')
```

```
the maximum value of tpr*(1-fpr) 0.245359515165 for threshold 0.833
Train Confusion Matrix
```

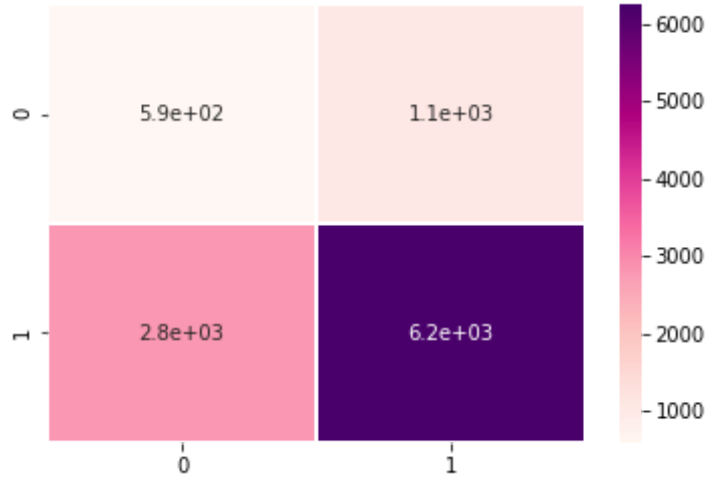
Out[133]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2207205f860>
```



In [134]:
```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html

# Test Confusion Matrix Heatmap

test_confusion_matrix = confusion_matrix(y_test, predict(y_test_pred, t
r_thresholds, test_fpr, test_fpr))

print("Test Confusion Matrix")
sns.heatmap(test_confusion_matrix,annot=True,linewidth = 0.5, cmap='RdP
u')
```

```
the maximum value of tpr*(1-fpr) 0.249670221435 for threshold 0.833
Test Confusion Matrix
```

Out[134]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22077f23d30>
```



2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

In [135]:
```
# Train data stack
X_tr = hstack((X_Train_categories_one_hot,X_Train_sub_categories_one_ho
t,X_Train_school_state_one_hot,
               X_Train_teacher_prefix_one_hot,X_Train_grade_cat_one_hot
,X_Train_tfidf_w2v_vectors,X_Train_titles_tfidf_w2v_vectors,
               X_train_price_norm)).tocsr()

# CV data Stack
X_cr = hstack((X_CV_categories_one_hot,X_CV_sub_categories_one_hot,X_CV
_school_state_one_hot,
               X_CV_teacher_prefix_one_hot,X_CV_grade_cat_one_hot,X_CV_
tfidf_w2v_vectors,X_CV_titles_tfidf_w2v_vectors,
               X_cv_price_norm)).tocsr()

# Test Data Stack
X_te = hstack((X_Test_categories_one_hot,X_Test_sub_categories_one_hot,
X_Test_school_state_one_hot,
               X_Test_teacher_prefix_one_hot,X_Test_grade_cat_one_hot,X
_Test_tfidf_w2v_vectors,X_Test_titles_tfidf_w2v_vectors,
               X_test_price_norm)).tocsr()
```

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

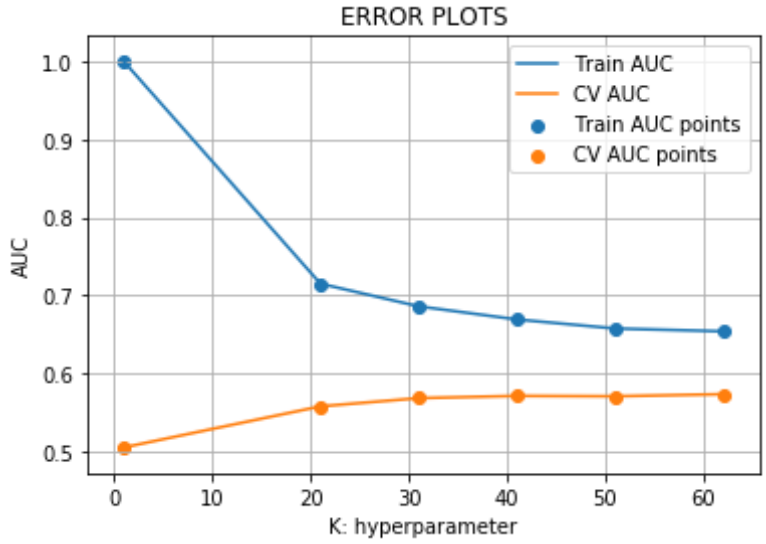
```
Final Data matrix
(14550, 699) (14550,)
(7167, 699) (7167,)
(10697, 699) (10697,)
====================================================================================
==============================
```

In [136]:

```
# Plotting error plot, AUC vs K plot to get best K(Bias-Variance trade-off)

train_auc = []
cv_auc = []

# Execute for different K values
K = [1, 21, 31, 41, 51, 62]

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_cr)

    # 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))

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

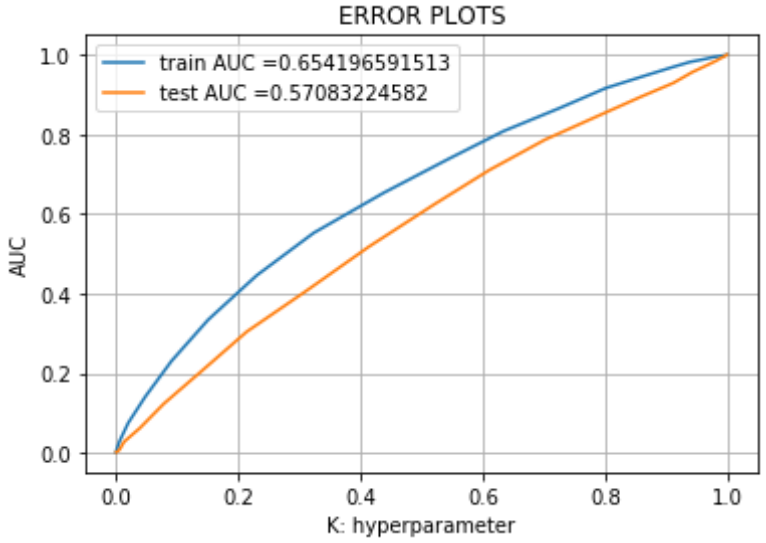
```
plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

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

```
100%|██████████████████████████████████████████████████████████████| 6/6 [59:06<00:00, 525.52s/it]
```



In [137]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve

neigh = KNeighborsClassifier(n_neighbors=60)
neigh.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilities 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, t
rain_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()
```



In [138]:
```
print("="*100)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, tr
ain_fpr, train_fpr)))
```

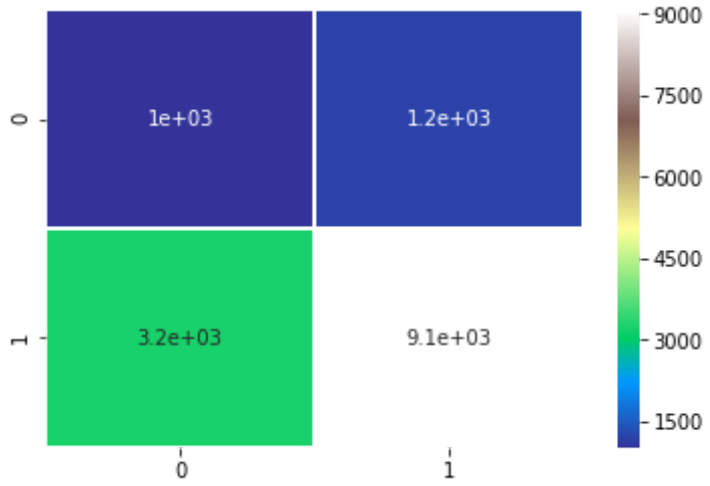
```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test
_fpr, test_fpr)))
```

```
==========================================================================
============================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.248173065563 for threshold 0.833
[[1027 1219]
 [3231 9073]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.249576417755 for threshold 0.85
[[ 792  860]
 [3399 5646]]
```

In [141]:
```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html
# Train Confusion Matrix Heatmap

train_confusion_matrix = confusion_matrix(y_train, predict(y_train_pred
, tr_thresholds, train_fpr, train_fpr))

print("Train Confusion Matrix")
sns.heatmap(train_confusion_matrix,annot=True,linewidth = 0.5, cmap='te
rrain')
```

```
the maximum value of tpr*(1-fpr) 0.248173065563 for threshold 0.833
Train Confusion Matrix
```

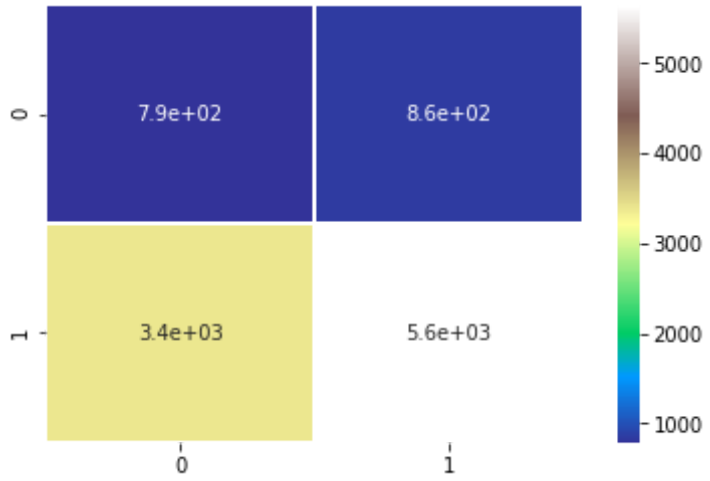
Out[141]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2206afe9048>
```



In [142]:
```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html

# Test Confusion Matrix Heatmap

test_confusion_matrix = confusion_matrix(y_test, predict(y_test_pred, t
r_thresholds, test_fpr, test_fpr))

print("Test Confusion Matrix")
sns.heatmap(test_confusion_matrix,annot=True,linewidth = 0.5, cmap='ter
rain')
```

```
the maximum value of tpr*(1-fpr) 0.249576417755 for threshold 0.85
Test Confusion Matrix
```

Out[142]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22071200c18>
```



2.5 Feature selection with `SelectKBest`

In [145]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html

from sklearn.feature_selection import SelectKBest, chi2

# Train data stack
X_tr = hstack((X_Train_categories_one_hot,X_Train_sub_categories_one_hot,X_Train_school_state_one_hot,
               X_Train_teacher_prefix_one_hot,X_Train_grade_cat_one_hot,X_Train_essay_bow,X_Train_titles_bow,
               X_train_price_norm)).tocsr()

# CV data Stack
X_cr = hstack((X_CV_categories_one_hot,X_CV_sub_categories_one_hot,X_CV_school_state_one_hot,
               X_CV_teacher_prefix_one_hot,X_CV_grade_cat_one_hot,X_CV_essay_bow,X_CV_titles_bow,
               X_cv_price_norm)).tocsr()

# Test Data Stack
```

```
X_te = hstack((X_Test_categories_one_hot,X_Test_sub_categories_one_hot,
X_Test_school_state_one_hot,
               X_Test_teacher_prefix_one_hot,X_Test_grade_cat_one_hot,X
_Test_essay_bow,X_Test_titles_bow,
               X_test_price_norm)).tocsr()

selectkbest = SelectKBest(chi2, k=500)
selectkbest.fit(X_tr, y_train)

X_tr_new = selectkbest.transform(X_tr)
X_cr_new = selectkbest.transform(X_cr)
X_te_new = selectkbest.transform(X_te)

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

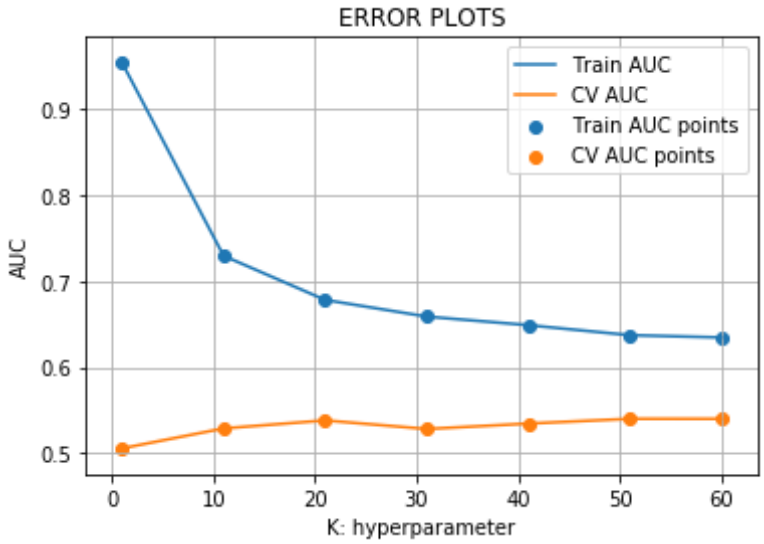
```
Final Data matrix
(14550, 500) (14550,)
(7167, 500) (7167,)
(10697, 500) (10697,)
====================================================================
============================
```

In [146]:

```
# Plotting error plot, AUC vs K plot to get best K(Bias-Variance trade-
off)

train_auc = []
cv_auc = []

# Execute for different K values
K = [1, 11, 21, 31, 41, 51, 60]

for i in tqdm(K):

    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_tr_new, y_train)
```

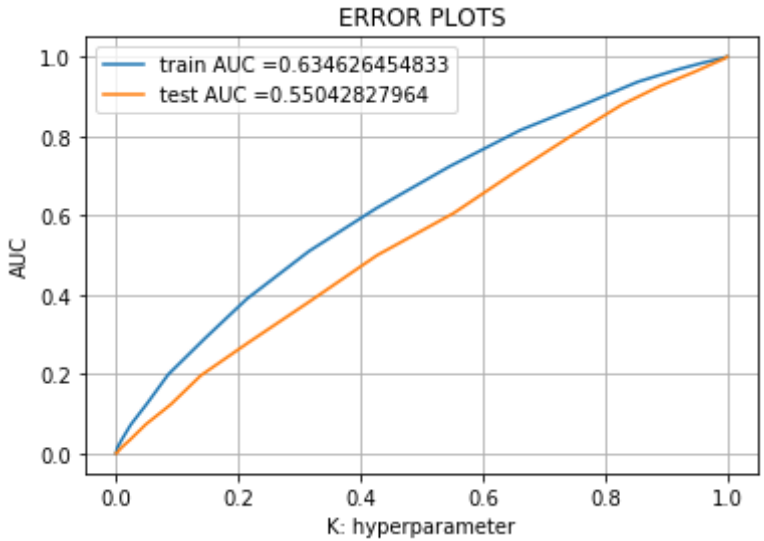
```
    y_train_pred = batch_predict(neigh, X_tr_new)
    y_cv_pred = batch_predict(neigh, X_cr_new)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probab
ility 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))

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

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

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

```
100%|██████████| 7/7 [01:14<00:00, 10.67s/it]
```



In [147]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve

neigh = KNeighborsClassifier(n_neighbors=60)
neigh.fit(X_tr_new, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilities estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr_new)
y_test_pred = batch_predict(neigh, X_te_new)

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

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

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



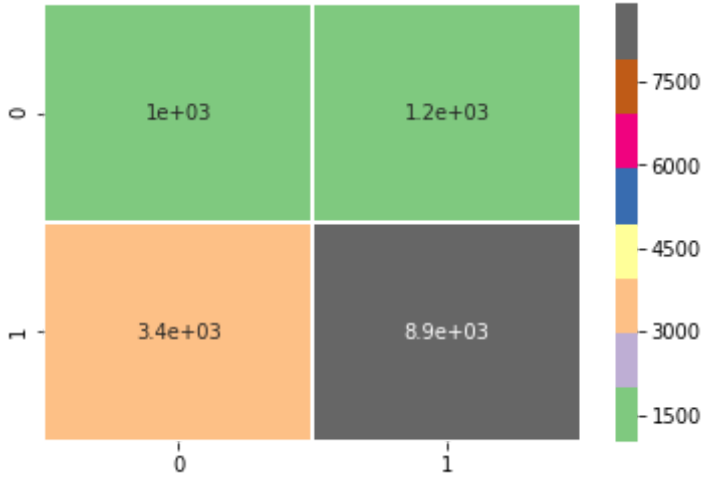
In [148]:
```
print("="*100)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, tr
ain_fpr, train_fpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test
_fpr, test_fpr)))
```

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.247977804015 for threshold 0.8
[[1022 1224]
 [3411 8893]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.247226562271 for threshold 0.8
[[ 573 1079]
 [2626 6419]]
```

In [154]:
```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html
# Train Confusion Matrix Heatmap

train_confusion_matrix = confusion_matrix(y_train, predict(y_train_pred
, tr_thresholds, train_fpr, train_fpr))

print("Train Confusion Matrix")
sns.heatmap(train_confusion_matrix,annot=True,linewidth = 0.5, cmap='Ac
cent')
```

```
the maximum value of tpr*(1-fpr) 0.247977804015 for threshold 0.8
Train Confusion Matrix
```

Out[154]: <matplotlib.axes._subplots.AxesSubplot at 0x220780819b0>

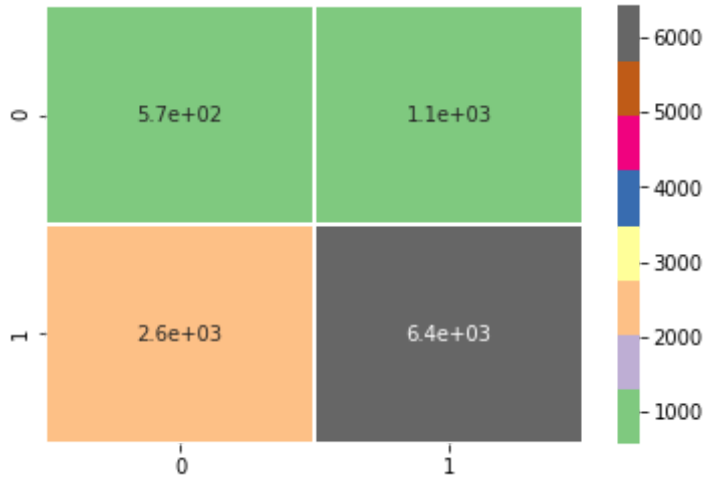


In [155]:
```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html

# Test Confusion Matrix Heatmap

test_confusion_matrix = confusion_matrix(y_test, predict(y_test_pred, t
r_thresholds, test_fpr, test_fpr))

print("Test Confusion Matrix")
```

```
sns.heatmap(test_confusion_matrix,annot=True,linewidth = 0.5, cmap='Acc
ent')
```

```
the maximum value of tpr*(1-fpr) 0.247226562271 for threshold 0.8
Test Confusion Matrix
```

Out[155]: <matplotlib.axes._subplots.AxesSubplot at 0x2207807b6d8>



3. Conclusions

In [158]:
```
# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Featurization", "Best K Value", "Train AUC", "Test AU
C"]

x.add_row(["KNN Brute force on BOW", 61, 0.62, 0.55])
x.add_row(["KNN Brute force on TFIDF", 60, 0.62, 0.55])
x.add_row(["KNN brute force on Avg W2V", 60, 0.62, 0.54])
```

```
x.add_row(["KNN brute force on TFIDF W2V", 60, 0.65, 0.57])
x.add_row(["Feature selection with SelectKBest", 60, 0.63, 0.55])

print(x)
```

```
+-----------------------------------+-------------+-----------+------
----+
|           Featurization           | Best K Value | Train AUC | Test
AUC |
+-----------------------------------+-------------+-----------+------
----+
|       KNN Brute force on BOW      |      61      |    0.62   |   0.5
5   |
|      KNN Brute force on TFIDF     |      60      |    0.62   |   0.5
5   |
|     KNN brute force on Avg W2V    |      60      |    0.62   |   0.5
4   |
|    KNN brute force on TFIDF W2V   |      60      |    0.65   |   0.5
7   |
| Feature selection with SelectKBest |      60      |    0.63   |   0.5
5   |
+-----------------------------------+-------------+-----------+------
----+
```