DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example**
`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 targete enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories f following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located (Two-letter U.S. postal (https://en.wikipedia.org/wiki/List_of_U.S._state_abbr **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategori **Examples:** • `Literacy` • `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources needed for the projec • `My students need hands on literacy mate sensory needs!`

Feature	Description
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*
`project_essay_4`	Fourth application essay*
`project_submitted_datetime`	Datetime when project application was submitted. **Exa** `12:43:56.245`
`teacher_id`	A unique identifier for the teacher of the proposed pro `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix`	Teacher's title. One of the following enumerated value • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted b **Example:** `2`

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [104]:

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

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

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

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

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

from tqdm import tqdm
import os

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

1.1 Reading Data

In [105]:

```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

In [106]:

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

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

In [107]:

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


#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/40840
39
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
project_data.drop('project_submitted_datetime', axis=1, inplace=True)
project_data.sort_values(by=['Date'], inplace=True)


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


project_data.head(2)
```

Out[107]:

	Unnamed: 0	id	teacher_id	teacher_prefix	scho
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT

In [108]:

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

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

Out[108]:

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

1.2 preprocessing of `project_subject_categories`

In [109]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47
301924/4084039

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

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [110]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47
301924/4084039

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

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", "Warmt
h", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "M
ath & 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:"M
ath & Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spa
ces
        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 [111]:

```
# 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 [112]:

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

Out[112]:

	Unnamed: 0	id	teacher_id	teacher_prefix	scho
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT

In [113]:

```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

In [114]:

```
#to drop a row having nan https://stackoverflow.com/questions/13413590
project_data=project_data.dropna(subset=['teacher_prefix'])
```

In [115]:

```
# 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)
print(project_data['essay'].values[99999])
print("="*50)
```

```
I have been fortunate enough to use the Fairy Tale STEM kits in my classro
om as well as the STEM journals, which my students really enjoyed.  I woul
d love to implement more of the Lakeshore STEM kits in my classroom for th
e next school year as they provide excellent and engaging STEM lessons.My
students come from a variety of backgrounds, including language and socioe
conomic status.  Many of them don't have a lot of experience in science an
d engineering and these kits give me the materials to provide these exciti
ng opportunities for my students.Each month I try to do several science or
STEM/STEAM projects.  I would use the kits and robot to help guide my scie
nce instruction in engaging and meaningful ways.  I can adapt the kits to
my current language arts pacing guide where we already teach some of the m
aterial in the kits like tall tales (Paul Bunyan) or Johnny Appleseed.  Th
e following units will be taught in the next school year where I will impl
ement these kits: magnets, motion, sink vs. float, robots.  I often get to
these units and don't know If I am teaching the right way or using the rig
ht materials.    The kits will give me additional ideas, strategies, and l
essons to prepare my students in science.It is challenging to develop high
quality science activities.  These kits give me the materials I need to pr
ovide my students with science activities that will go along with the curr
iculum in my classroom.  Although I have some things (like magnets) in my
classroom, I don't know how to use them effectively.  The kits will provid
e me with the right amount of materials and show me how to use them in an
appropriate way.
==================================================
I teach high school English to students with learning and behavioral disab
ilities. My students all vary in their ability level. However, the ultimat
e goal is to increase all students literacy levels. This includes their re
ading, writing, and communication levels.I teach a really dynamic group of
students. However, my students face a lot of challenges. My students all l
ive in poverty and in a dangerous neighborhood. Despite these challenges,
I have students who have the the desire to defeat these challenges. My stu
dents all have learning disabilities and currently all are performing belo
w grade level. My students are visual learners and will benefit from a cla
ssroom that fulfills their preferred learning style.The materials I am req
uesting will allow my students to be prepared for the classroom with the n
ecessary supplies.  Too often I am challenged with students who come to sc
hool unprepared for class due to economic challenges.  I want my students
to be able to focus on learning and not how they will be able to get schoo
l supplies.  The supplies will last all year.  Students will be able to co
mplete written assignments and maintain a classroom journal.  The chart pa
per will be used to make learning more visual in class and to create poste
rs to aid students in their learning.  The students have access to a class
room printer.  The toner will be used to print student work that is comple
ted on the classroom Chromebooks.I want to try and remove all barriers for
the students learning and create opportunities for learning. One of the bi
ggest barriers is the students not having the resources to get pens, pape
r, and folders. My students will be able to increase their literacy skills
because of this project.
==================================================
\"Life moves pretty fast. If you don't stop and look around once in awhil
e, you could miss it.\"  from the movie, Ferris Bueller's Day Off.  Think
back...what do you remember about your grandparents?  How amazing would it
be to be able to flip through a book to see a day in their lives?My second
graders are voracious readers! They love to read both fiction and nonficti
on books.  Their favorite characters include Pete the Cat, Fly Guy, Piggie
and Elephant, and Mercy Watson. They also love to read about insects, spac
e and plants. My students are hungry bookworms! My students are eager to l
earn and read about the world around them. My kids love to be at school an
d are like little sponges absorbing everything around them. Their parents
work long hours and usually do not see their children. My students are usu
ally cared for by their grandparents or a family friend. Most of my studen
```

```
ts do not have someone who speaks English at home. Thus it is difficult fo
r my students to acquire language.Now think forward... wouldn't it mean a
lot to your kids, nieces or nephews or grandchildren, to be able to see a
day in your life today 30 years from now? Memories are so precious to us a
nd being able to share these memories with future generations will be a re
warding experience.  As part of our social studies curriculum, students wi
ll be learning about changes over time.  Students will be studying photos
to learn about how their community has changed over time.  In particular,
we will look at photos to study how the land, buildings, clothing, and sch
ools have changed over time.  As a culminating activity, my students will
capture a slice of their history and preserve it through scrap booking. Ke
y important events in their young lives will be documented with the date,
location, and names.   Students will be using photos from home and from sc
hool to create their second grade memories.   Their scrap books will prese
rve their unique stories for future generations to enjoy.Your donation to
this project will provide my second graders with an opportunity to learn a
bout social studies in a fun and creative manner.  Through their scrapbook
s, children will share their story with others and have a historical docum
ent for the rest of their lives.
==================================================
Some of my students come from difficult family lives, but they don't let t
hat stop them. We have built a community in our classroom that allows each
student to be comfortable with who they are. Even though we are a diverse
school, everyone feels included. We have a high Hispanic population, and a
bout 90% of the students are on free or reduced-price lunch. Most students
are living with a single parent or both parents work full time, although m
any parents are eager to help in any way they can.\r\nWe all know how impo
rtant it is to get kids up and moving. I want my classroom to be a place w
here students can be active phyically and mentally. The requested items wi
ll allow my students to move all day. When they are sitting in a chair, th
eir movement is limited.\r\n       Kindergarten students have a hard time
sitting still for long periods of time. They would much rather bounce on a
stability ball or wiggle on a cushion than sit in a hard chair. Having the
se choices in my classroom will allow students to be active and learn at t
he same time. \r\n        Having these choices in my classroom will also b
uild a greater bond between the students.  They will learn to choose which
seat best fits their learning style, and hopefully they will be able to he
lp their classmates find a seat that works for them. As the students move
around the room, they will be able to work with everyone instead of being
with one group each day.nannan
==================================================
\"This is how mathematicians do it! Remember we are all mathematicians in
this classroom!\" A few simple words repeated regularly-words that instill
a sense of pride in each of my students!\r\n\r\nI am proud to teach math i
n rural Alabama where our Title I school has both administrator and parent
al support.\r\nThis sense of community pride has been instilled in the stu
dents. It is visible in the respect they give both faculty and visitors to
our campus. It is apparent in their love of learning. Our parents want the
best for their children, but many of them, due to their own economic conce
rns, can offer very little in the way of anything more than just the basic
s. Many of our students live in homes without computers or Internet acces
s. I feel that due to the socioeconomic status of many of my students it i
s my job to level the playing field as much as possible so that my student
s will have the same learning opportunities as other students and will be
able to compete in the global market.Many of my students come from low soc
ioeconomic homes. These homes have little in the way of hands on toys. In
addition, even those from affluent backgrounds, are more interested in ele
ctronics  than \"old fashion\" toys. As a result, more and more students,
find measurement- especially area, a very abstract concept!\r\n  Building
bricks would provide students a hands on opportunity to explore measuremen
t and area in an engaging way! Not only would students be able to explore
```

```
measurement, but also to learn more about the real world skills of designi
ng and building structures. It would also encourage teamwork and problem s
olving as students developed an idea and then worked to implement their id
ea.\r\n   How exciting would it be for students to have the opportunity to
not only learn so much, but to also have fun through such a simple items a
s building bricks!\r\nnannan
==================================================
```

In [116]:

```
# 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 [117]:

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

```
Some of my students come from difficult family lives, but they do not let
that stop them. We have built a community in our classroom that allows eac
h student to be comfortable with who they are. Even though we are a divers
e school, everyone feels included. We have a high Hispanic population, and
about 90% of the students are on free or reduced-price lunch. Most student
s are living with a single parent or both parents work full time, although
many parents are eager to help in any way they can.\r\nWe all know how imp
ortant it is to get kids up and moving. I want my classroom to be a place
where students can be active phyically and mentally. The requested items w
ill allow my students to move all day. When they are sitting in a chair, t
heir movement is limited.\r\n       Kindergarten students have a hard time
sitting still for long periods of time. They would much rather bounce on a
stability ball or wiggle on a cushion than sit in a hard chair. Having the
se choices in my classroom will allow students to be active and learn at t
he same time. \r\n        Having these choices in my classroom will also b
uild a greater bond between the students.  They will learn to choose which
seat best fits their learning style, and hopefully they will be able to he
lp their classmates find a seat that works for them. As the students move
around the room, they will be able to work with everyone instead of being
with one group each day.nannan
==================================================
```

In [118]:

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

```
Some of my students come from difficult family lives, but they do not let
that stop them. We have built a community in our classroom that allows eac
h student to be comfortable with who they are. Even though we are a divers
e school, everyone feels included. We have a high Hispanic population, and
about 90% of the students are on free or reduced-price lunch. Most student
s are living with a single parent or both parents work full time, although
many parents are eager to help in any way they can.  We all know how impor
tant it is to get kids up and moving. I want my classroom to be a place wh
ere students can be active phyically and mentally. The requested items wil
l allow my students to move all day. When they are sitting in a chair, the
ir movement is limited.        Kindergarten students have a hard time sit
ting still for long periods of time. They would much rather bounce on a st
ability ball or wiggle on a cushion than sit in a hard chair. Having these
choices in my classroom will allow students to be active and learn at the
same time.          Having these choices in my classroom will also build
a greater bond between the students.  They will learn to choose which seat
best fits their learning style, and hopefully they will be able to help th
eir classmates find a seat that works for them. As the students move aroun
d the room, they will be able to work with everyone instead of being with
one group each day.nannan
```

In [119]:

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

```
Some of my students come from difficult family lives but they do not let t
hat stop them We have built a community in our classroom that allows each
student to be comfortable with who they are Even though we are a diverse s
chool everyone feels included We have a high Hispanic population and about
90 of the students are on free or reduced price lunch Most students are li
ving with a single parent or both parents work full time although many par
ents are eager to help in any way they can We all know how important it is
to get kids up and moving I want my classroom to be a place where students
can be active phyically and mentally The requested items will allow my stu
dents to move all day When they are sitting in a chair their movement is l
imited Kindergarten students have a hard time sitting still for long perio
ds of time They would much rather bounce on a stability ball or wiggle on
a cushion than sit in a hard chair Having these choices in my classroom wi
ll allow students to be active and learn at the same time Having these cho
ices in my classroom will also build a greater bond between the students T
hey will learn to choose which seat best fits their learning style and hop
efully they will be able to help their classmates find a seat that works f
or them As the students move around the room they will be able to work wit
h everyone instead of being with one group each day nannan
```

In [120]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you'r
e", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 't
hey', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
at'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
d', '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', 'ov
er', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
y', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too'
, 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
w', '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', "isn't", 'ma', 'migh
tn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
asn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [121]:

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

```
100%|█████████████████████████████████████████████████████████████████████
███| 109245/109245 [01:19<00:00, 1377.42it/s]
```

In [122]:

```
# after preprocesing
preprocessed_essays[20000]
```

Out[122]:

```
'students come difficult family lives not let stop built community classro
om allows student comfortable even though diverse school everyone feels in
cluded high hispanic population 90 students free reduced price lunch stude
nts living single parent parents work full time although many parents eage
r help way know important get kids moving want classroom place students ac
tive phyically mentally requested items allow students move day sitting ch
air movement limited kindergarten students hard time sitting still long pe
riods time would much rather bounce stability ball wiggle cushion sit hard
chair choices classroom allow students active learn time choices classroom
also build greater bond students learn choose seat best fits learning styl
e hopefully able help classmates find seat works students move around room
able work everyone instead one group day nannan'
```

1.4 Preprocessing of `project_title`

In [123]:

```
# printing some random reviews
print(project_data['project_title'].values[0])
print("="*50)
print(project_data['project_title'].values[150])
print("="*50)
print(project_data['project_title'].values[1000])
print("="*50)
print(project_data['project_title'].values[20000])
print("="*50)
print(project_data['project_title'].values[99999])
print("="*50)
```

```
Engineering STEAM into the Primary Classroom
==================================================
Building Blocks for Learning
==================================================
Empowering Students Through Art:Learning About Then and Now
==================================================
Wiggle While We Learn
==================================================
Area Architect!
==================================================
```

In [124]:

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

```
100%|██████████████████████████████████████████████████████████████████████
██| 109245/109245 [00:03<00:00, 30578.98it/s]
```

In [125]:

```
# after preprocesing
preprocessed_titles[10000]
```

Out[125]:

```
'mobile seating center alternative seating classroom'
```

1.5 Preparing data for models

In [126]:

```
project_data.columns
```

Out[126]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', 'project_title', 'project_essay_
1',
       'project_essay_2', 'project_essay_3', 'project_essay_4',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approve
d',
       '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/ (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/)

In [127]:

```
# 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', 'AppliedLearnin
g', 'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig  (109245, 9)
```

In [128]:

```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=Fal
se, binary=True)
sub_categories_one_hot = vectorizer.fit_transform(project_data['clean_subcategories'].v
alues)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",sub_categories_one_hot.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvemen
t', 'Extracurricular', 'Civics_Government', 'ForeignLanguages', 'Nutrition
Education', 'Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingArts',
'CharacterEducation', 'TeamSports', 'Other', 'College_CareerPrep', 'Musi
c', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL',
'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'A
ppliedSciences', 'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Lit
eracy']
Shape of matrix after one hot encodig  (109245, 30)
```

In [129]:

```
#Vectorizing Categorical data:State

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


# dict sort by value python: https://stackoverflow.com/a/613218/4084039
state_dict = dict(my_counter)
sorted_state_dict = dict(sorted(state_dict.items(), key=lambda kv: kv[1]))

# we use count vectorizer to convert the values into one hot encoded features
from sklearn.feature_extraction.text import CountVectorizer
vectorizer1 = CountVectorizer(vocabulary=list(sorted_state_dict.keys()), lowercase=Fals
e, binary=True)
vectorizer1.fit(project_data['school_state'].values)
print(vectorizer1.get_feature_names())


state_one_hot = vectorizer1.transform(project_data['school_state'].values)

print("Shape of matrix after one hot encodig ",state_one_hot.shape)
```

```
['VT', 'WY', 'ND', 'MT', 'RI', 'SD', 'NE', 'DE', 'AK', 'NH', 'WV', 'ME',
'HI', 'DC', 'NM', 'KS', 'IA', 'ID', 'AR', 'CO', 'MN', 'OR', 'KY', 'MS', 'N
V', 'MD', 'CT', 'TN', 'UT', 'AL', 'WI', 'VA', 'AZ', 'NJ', 'OK', 'WA', 'M
A', 'LA', 'OH', 'MO', 'IN', 'PA', 'MI', 'SC', 'GA', 'IL', 'NC', 'FL', 'N
Y', 'TX', 'CA']
Shape of matrix after one hot encodig  (109245, 51)
```

In [130]:

```
#Vectorizing Categorical data:teacher_prefix

def partition(i):
    return i.replace('.', '')

prefix = project_data['teacher_prefix']
actual_prefix = prefix.map(partition)
project_data['teacher_prefix'] = actual_prefix

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

#dict sort by value python: https://stackoverflow.com/a/613218/4084039
teacher_dict = dict(my_counter)
sorted_teacher_dict = dict(sorted(teacher_dict.items(), key=lambda kv: kv[1]))

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

vectorizer1 = CountVectorizer(vocabulary=list(sorted_teacher_dict.keys()), lowercase=Fa
lse, binary=True)
vectorizer1.fit(project_data['teacher_prefix'].values)
print(vectorizer1.get_feature_names())


prefix_one_hot = vectorizer1.transform(project_data['teacher_prefix'].values)

print("Shape of matrix after one hot encodig ",prefix_one_hot.shape)
```

```
['Dr', 'Teacher', 'Mr', 'Ms', 'Mrs']
Shape of matrix after one hot encodig  (109245, 5)
```

In [131]:

```
def partition(i):
    return i.replace('-', '_')

prefix = project_data['project_grade_category']
actual_prefix = prefix.map(partition)
project_data['project_grade_category'] = actual_prefix

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


# dict sort by value python: https://stackoverflow.com/a/613218/4084039
grade_dict = dict(my_counter)
sorted_grade_dict = dict(sorted(grade_dict.items(), key=lambda kv: kv[1]))

#https://thispointer.com/different-ways-to-remove-a-key-from-dictionary-in-python/
if "Grades" in sorted_grade_dict:
    del sorted_grade_dict["Grades"]


#Vectorizing Categorical data:project_grade_category

# we use count vectorizer to convert the values into one hot encoded features
vectorizer3 = CountVectorizer(vocabulary=list(sorted_grade_dict.keys()), lowercase=Fals
e, binary=True)
vectorizer3.fit(project_data['project_grade_category'].values)
print(vectorizer3.get_feature_names())


grade_one_hot = vectorizer3.transform(project_data['project_grade_category'].values)
print("Shape of matrix after one hot encodig ",grade_one_hot.shape)
```

```
['9_12', '6_8', '3_5', 'PreK_2']
Shape of matrix after one hot encodig  (109245, 4)
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [132]:

```
# We are considering only the words which appeared in at least 10 documents(rows or pro
jects).
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  (109245, 16512)
```

In [133]:

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

```
Shape of matrix after one hot encodig  (109245, 3222)
```

1.5.2.2 TFIDF vectorizer

In [134]:

```
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  (109245, 16512)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [135]:

```
'''
# 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 our 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-p
ickle-to-save-and-load-variables-in-python/

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


'''
```

Out[135]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/3823034
9/4084039\ndef loadGloveModel(gloveFile):\n    print ("Loading Glove Mode
l")\n    f = open(gloveFile,\'r\', encoding="utf8")\n    model = {}\n    f
or line in tqdm(f):\n        splitLine = line.split()\n        word = spli
tLine[0]\n        embedding = np.array([float(val) for val in splitLine
[1:]])\n        model[word] = embedding\n    print ("Done.",len(model)," w
ords loaded!")\n    return model\nmodel = loadGloveModel(\'glove.42B.300d.
txt\')\n\n# ============================\nOutput:\n    \nLoading Glove Mod
el\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n# ====
======================\n\nwords = []\nfor i in preproced_texts:\n    wor
ds.extend(i.split(\' \'))\n\nfor i in preproced_titles:\n    words.extend
(i.split(\' \'))\nprint("all the words in the coupus", len(words))\nwords
= set(words)\nprint("the unique words in the coupus", len(words))\n\ninter
_words = set(model.keys()).intersection(words)\nprint("The number of words
that are present in both glove vectors and our coupus",       len(inter_wo
rds),"(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_courpu
s = {}\nwords_glove = set(model.keys())\nfor i in words:\n    if i in word
s_glove:\n        words_courpus[i] = model[i]\nprint("word 2 vec length",
len(words_courpus))\n\n\n# stronging variables into pickle files python: h
ttp://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-
python/\n\nimport pickle\nwith open(\'glove_vectors\', \'wb\') as f:\n
pickle.dump(words_courpus, f)\n\n\n'
```

In [136]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-p
ickle-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 [137]:

```
# 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/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors.append(vector)

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

```
100%|█████████████████████████████████████████████████████████████████████
███| 109245/109245 [00:43<00:00, 2534.71it/s]

109245
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [138]:

```
# 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 value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [139]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_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
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting 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%|██████████████████████████████████████████████████████████████████████
████| 109245/109245 [04:58<00:00, 366.57it/s]

109245
300
```

In [140]:

```
#TFIDF Vectorizer on project_title
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
title_tfidf = vectorizer.fit_transform(preprocessed_titles)
print("Shape of matrix after one hot encodig ",title_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (109245, 3222)
```

In [141]:

```
#Using Pretrained Models: AVG W2V on project_title

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

# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_title = []; # the avg-w2v for each sentence/review is stored in this li
st
for sentence in tqdm(preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_title.append(vector)

print(len(avg_w2v_vectors_title))
print(len(avg_w2v_vectors_title[0]))
```

```
100%|█████████████████████████████████████████████████████████████████████
██| 109245/109245 [00:01<00:00, 60503.78it/s]

109245
300
```

In [142]:

```
#Using Pretrained Models: TFIDF weighted W2V on project_title

# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_titles)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words_title = set(tfidf_model.get_feature_names())

# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_title = []; # the avg-w2v for each sentence/review is stored in this
 list
for sentence in tqdm(preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_title):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting 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_title.append(vector)

print(len(tfidf_w2v_vectors_title))
print(len(tfidf_w2v_vectors_title[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
██| 109245/109245 [00:04<00:00, 24914.71it/s]

109245
300
```

1.5.3 Vectorizing Numerical features

In [143]:

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

In [144]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.pr
eprocessing.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=[725.05 213.03 329.   ...
 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 sta
ndard deviation of this data
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_
[0])}")

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

```
Mean : 298.1152448166964, Standard deviation : 367.49642545627506
```

In [145]:

```
price_standardized
```

Out[145]:

```
array([[ 1.16173852],
       [-0.23152673],
       [ 0.08404097],
       ...,
       [ 0.27451901],
       [-0.02825944],
       [-0.7962397 ]])
```

1.5.4 Merging all the above features

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

In [146]:

```
print(categories_one_hot.shape)
print(sub_categories_one_hot.shape)
print(text_bow.shape)
print(price_standardized.shape)
```

```
(109245, 9)
(109245, 30)
(109245, 16512)
(109245, 1)
```

In [147]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
:)
X = hstack((categories_one_hot, sub_categories_one_hot, text_bow, price_standardized))
X.shape
```

Out[147]:

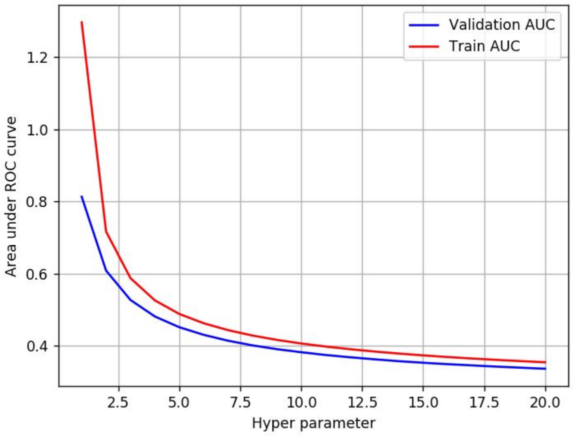
```
(109245, 16552)
```

Assignment 3: Apply KNN

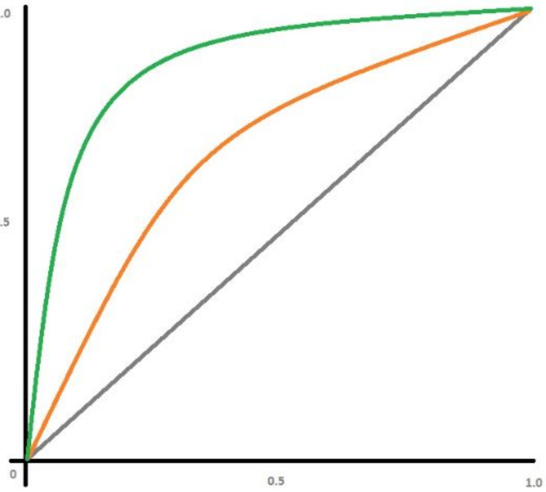
1. **[Task-1] Apply KNN(brute force version) on these feature sets**
   - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)
   - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF)
   - Set 3: categorical, numerical features + 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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) 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

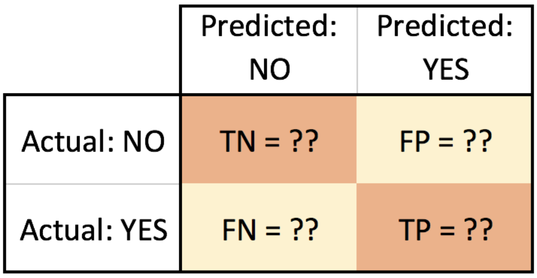
3. **Representation of results**
   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, as shown in the figure



   - 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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points



	Predicted: NO	Predicted: YES
Actual: NO	TN = ??	FP = ??
Actual: YES	FN = ??	TP = ??

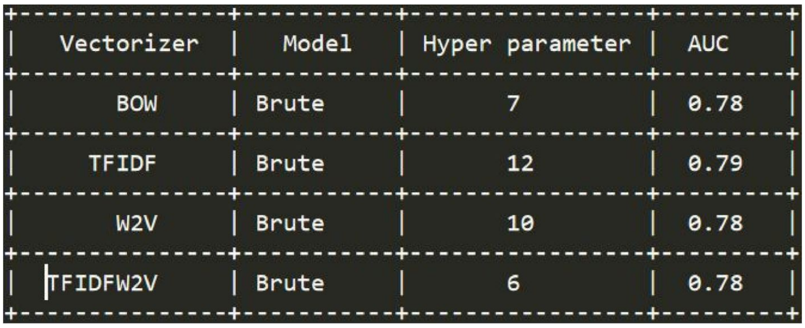
4. **[Task-2]**

- Select top 2000 features from feature Set 2 using `SelectKBest` (https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html) and then apply KNN on top of these features

-
  ```
  from sklearn.datasets import load_digits
  from sklearn.feature_selection import SelectKBest, chi2
  X, y = load_digits(return_X_y=True)
  X.shape
  X_new = SelectKBest(chi2, k=20).fit_transform(X, y)
  X_new.shape
  ========
  output:
  (1797, 64)
  (1797, 20)
  ```
- Repeat the steps 2 and 3 on the data matrix after feature selection

5. **Conclusion**

- You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link (http://zetcode.com/python/prettytable/)



```
+---------------+-----------+-----------------+---------+
|   Vectorizer  |   Model   | Hyper parameter |  AUC    |
+---------------+-----------+-----------------+---------+
|      BOW      | Brute     |        7        |  0.78   |
+---------------+-----------+-----------------+---------+
|     TFIDF     | Brute     |       12        |  0.79   |
+---------------+-----------+-----------------+---------+
|      W2V      | Brute     |       10        |  0.78   |
+---------------+-----------+-----------------+---------+
|   TFIDFW2V    | Brute     |        6        |  0.78   |
+---------------+-----------+-----------------+---------+
```

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. (https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf)

2. K Nearest Neighbor

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

In [148]:

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

Out[148]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_s
0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs	CA
1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms	UT
2	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs	CA

In [149]:

```
project_data1 = project_data
y = project_data1['project_is_approved'].values
project_data1.drop(['project_is_approved'], axis=1, inplace=True)
project_data1.head(1)
```

Out[149]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_st
0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs	CA

In [150]:

```
X = project_data1
# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, strat
ify=y_train)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

2.2.1 Normalizing the numerical features: Price

In [151]:

```
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.reshape(-1,1))
X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(-1,1))

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

```
After vectorizations
(49039, 1) (49039,)
(24155, 1) (24155,)
(36051, 1) (36051,)
==========================================================================
==========================
```

2.2.2 Normalizing the numerical features: teacher_number_of_previously_posted_projects

In [152]:

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

X_train_tnppp_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_
projects'].values.reshape(-1,1))
X_cv_tnppp_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projec
ts'].values.reshape(-1,1))
X_test_tnppp_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_pr
ojects'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_tnppp_norm.shape, y_train.shape)
print(X_cv_tnppp_norm.shape, y_cv.shape)
print(X_test_tnppp_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49039, 1) (49039,)
(24155, 1) (24155,)
(36051, 1) (36051,)
==========================================================================
==========================
```

2.2.3 one hot encoding the catogorical features:clean_categories

In [153]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False,
binary=True)
vectorizer.fit(X_train['clean_categories'].values)

X_train_categories_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_cv_categories_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_categories_ohe = vectorizer.transform(X_test['clean_categories'].values)
print("After vectorizations")
print(X_train_categories_ohe.shape, y_train.shape)
print(X_cv_categories_ohe.shape, y_cv.shape)
print(X_test_categories_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(49039, 9) (49039,)
(24155, 9) (24155,)
(36051, 9) (36051,)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearnin
g', 'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
===========================================================================
=========================
```

2.2.4 one hot encoding the catogorical features: clean_subcategories

In [154]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=True)
vectorizer.fit(X_train['clean_subcategories'].values)
X_train_sub_categories_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
X_cv_sub_categories_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_sub_categories_ohe = vectorizer.transform(X_test['clean_subcategories'].values)
print("After vectorizations")
print(X_train_sub_categories_ohe.shape, y_train.shape)
print(X_cv_sub_categories_ohe.shape, y_cv.shape)
print(X_test_sub_categories_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(49039, 30) (49039,)
(24155, 30) (24155,)
(36051, 30) (36051,)
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvemen
t', 'Extracurricular', 'Civics_Government', 'ForeignLanguages', 'Nutrition
Education', 'Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingArts',
'CharacterEducation', 'TeamSports', 'Other', 'College_CareerPrep', 'Musi
c', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL',
'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'A
ppliedSciences', 'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Lit
eracy']
==========================================================================
==========================
```

2.2.5 one hot encoding the catogorical features: school_state

In [155]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(49039, 51) (49039,)
(24155, 51) (24155,)
(36051, 51) (36051,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi',
'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'm
o', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'o
k', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'w
i', 'wv', 'wy']
===========================================================================
=========================
```

2.2.6 one hot encoding the catogorical features: teacher_prefix

In [156]:

```
vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_cv_teacher_ohe.shape, y_cv.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(49039, 5) (49039,)
(24155, 5) (24155,)
(36051, 5) (36051,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
===========================================================================
=========================
```

2.2.7 one hot encoding the catogorical features: project_grade_category

In [157]:

```
#one hot encoding of project_grade_category for X_train
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
from collections import Counter
my_counter = Counter()
for word in X_train['project_grade_category'].values:
    my_counter.update(word.split())


# dict sort by value python: https://stackoverflow.com/a/613218/4084039
grade_dict = dict(my_counter)
sorted_grade_dict = dict(sorted(grade_dict.items(), key=lambda kv: kv[1]))

#https://thispointer.com/different-ways-to-remove-a-key-from-dictionary-in-python/
if "Grades" in sorted_grade_dict:
    del sorted_grade_dict["Grades"]


#Vectorizing Categorical data:project_grade_category

# we use count vectorizer to convert the values into one hot encoded features
vectorizer = CountVectorizer(vocabulary=list(sorted_grade_dict.keys()), lowercase=False
, binary=True)
vectorizer.fit(X_train['project_grade_category'].values)
print(vectorizer.get_feature_names())


X_train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].values)
print(X_train_grade_ohe.shape, y_train.shape)
```

```
['9_12', '6_8', '3_5', 'PreK_2']
(49039, 4) (49039,)
```

In [158]:

```
#one hot encoding of project_grade_category for X_cv
my_counter = Counter()
for word in X_cv['project_grade_category'].values:
    my_counter.update(word.split())


# dict sort by value python: https://stackoverflow.com/a/613218/4084039
grade_dict = dict(my_counter)
sorted_grade_dict = dict(sorted(grade_dict.items(), key=lambda kv: kv[1]))

#https://thispointer.com/different-ways-to-remove-a-key-from-dictionary-in-python/
if "Grades" in sorted_grade_dict:
    del sorted_grade_dict["Grades"]


#Vectorizing Categorical data:project_grade_category

# we use count vectorizer to convert the values into one hot encoded features
vectorizer = CountVectorizer(vocabulary=list(sorted_grade_dict.keys()), lowercase=False
, binary=True)
vectorizer.fit(X_cv['project_grade_category'].values)
print(vectorizer.get_feature_names())


X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
print(X_cv_grade_ohe.shape, y_cv.shape)
```

```
['9_12', '6_8', '3_5', 'PreK_2']
(24155, 4) (24155,)
```

In [159]:

```
#one hot encoding of project_grade_category for X_test
my_counter = Counter()
for word in X_test['project_grade_category'].values:
    my_counter.update(word.split())



# dict sort by value python: https://stackoverflow.com/a/613218/4084039
grade_dict = dict(my_counter)
sorted_grade_dict = dict(sorted(grade_dict.items(), key=lambda kv: kv[1]))

#https://thispointer.com/different-ways-to-remove-a-key-from-dictionary-in-python/
if "Grades" in sorted_grade_dict:
    del sorted_grade_dict["Grades"]



#Vectorizing Categorical data:project_grade_category

# we use count vectorizer to convert the values into one hot encoded features
vectorizer = CountVectorizer(vocabulary=list(sorted_grade_dict.keys()), lowercase=False
, binary=True)
vectorizer.fit(X_test['project_grade_category'].values)
print(vectorizer3.get_feature_names())



X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].values)
print(X_test_grade_ohe.shape, y_test.shape)
```

```
['9_12', '6_8', '3_5', 'PreK_2']
(36051, 4) (36051,)
```

In [160]:

```
print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(49039, 4) (49039,)
(24155, 4) (24155,)
(36051, 4) (36051,)
['9_12', '6_8', '3_5', 'PreK_2']
==========================================================================
==========================
```

2.3 Make Data Model Ready: encoding eassay, and project_title

2.3.1.1 Text preprocessing

In [161]:

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

```
100%|████████████████████████████████████████████████████████████████████
█████| 49039/49039 [00:34<00:00, 1439.54it/s]
```

In [162]:

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

```
100%|████████████████████████████████████████████████████████████████████
█████| 24155/24155 [00:16<00:00, 1479.94it/s]
```

In [163]:

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

```
100%|████████████████████████████████████████████████████████████████████
█████| 36051/36051 [00:26<00:00, 1385.41it/s]
```

2.3.1.2 Vectorizing Text data:Bag of words

In [164]:

```
# We are considering only the words which appeared in at least 10 documents(rows or pro
jects).
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(X_train_preprocessed_essays) # fit has to happen only on train data

X_train_text_bow = vectorizer.transform(X_train_preprocessed_essays)
X_cv_text_bow = vectorizer.transform(X_cv_preprocessed_essays)
X_test_text_bow = vectorizer.transform(X_test_preprocessed_essays)

print("After vectorizations")
print(X_train_text_bow.shape, y_train.shape)
print(X_cv_text_bow.shape, y_cv.shape)
print(X_test_text_bow.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49039, 12023) (49039,)
(24155, 12023) (24155,)
(36051, 12023) (36051,)
==========================================================================
==========================
```

2.3.1.3 Vectorizing Text data: tfidf

In [165]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train_preprocessed_essays) # fit has to happen only on train data
X_train_text_tfidf = vectorizer.transform(X_train_preprocessed_essays)
X_cv_text_tfidf = vectorizer.transform(X_cv_preprocessed_essays)
X_test_text_tfidf = vectorizer.transform(X_test_preprocessed_essays)

print("After vectorizations")
print(X_train_text_tfidf.shape, y_train.shape)
print(X_cv_text_tfidf.shape, y_cv.shape)
print(X_test_text_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49039, 12023) (49039,)
(24155, 12023) (24155,)
(36051, 12023) (36051,)
==========================================================================
==========================
```

2.3.1.4 Vectorizing Text data: avg w2v

In [166]:

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

In [167]:

```
# average Word2Vec for X_train
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/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_train_avg_w2v_vectors.append(vector)

print(len(X_train_avg_w2v_vectors))
print(len(X_train_avg_w2v_vectors[0]))
```

```
100%|█████████████████████████████████████████████████████████████████████
█████| 49039/49039 [00:15<00:00, 3069.46it/s]

49039
300
```

In [168]:

```
# average Word2Vec for X_cv
X_cv_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this lis
t
for sentence in tqdm(X_cv_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_cv_avg_w2v_vectors.append(vector)

print(len(X_cv_avg_w2v_vectors))
print(len(X_cv_avg_w2v_vectors[0]))
```

```
100%|█████████████████████████████████████████████████████████████████████
█████| 24155/24155 [00:07<00:00, 3257.55it/s]

24155
300
```

In [169]:

```
# average Word2Vec for X_test
X_test_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this l
ist
for sentence in tqdm(X_test_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_test_avg_w2v_vectors.append(vector)

print(len(X_test_avg_w2v_vectors))
print(len(X_test_avg_w2v_vectors[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
█████| 36051/36051 [00:11<00:00, 3137.93it/s]

36051
300
```

2.3.1.5 Vectorizing Text data: tfidf weighted w2v

In [170]:

```
#tfidf w2v for X_train

# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit_transform(X_train_preprocessed_essays)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())

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/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the 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%|██████████████████████████████████████████████████████████████████████
██████| 49039/49039 [02:08<00:00, 382.58it/s]

49039
300
```

In [171]:

```
#tfidf w2v for X_cv

# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model.transform(X_cv_preprocessed_essays)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())

X_cv_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this l
ist
for sentence in tqdm(X_cv_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting 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%|██████████████████████████████████████████████████████████████████████
██████| 24155/24155 [01:04<00:00, 374.14it/s]

24155
300
```

In [172]:

```
#tfidf w2v for X_test

# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model.transform(X_test_preprocessed_essays)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())

X_test_tfidf_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/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting 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%|██████████████████████████████████████████████████████████████████████
██████| 36051/36051 [01:29<00:00, 402.60it/s]

36051
300
```

2.3.2.1 Title preprocessing

In [173]:

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

```
100%|██████████████████████████████████████████████████████████████████████
████| 49039/49039 [00:01<00:00, 30221.37it/s]
```

In [174]:

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

```
100%|█████████████████████████████████████████████████████████████████████
████| 24155/24155 [00:00<00:00, 29717.55it/s]
```

In [175]:

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

```
100%|█████████████████████████████████████████████████████████████████████
████| 36051/36051 [00:01<00:00, 30606.46it/s]
```

2.3.2.2 Vectorizing project_titles data:Bag of words

In [176]:

```
# We are considering only the words which appeared in at least 10 documents(rows or pro
jects).
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(X_train_preprocessed_titles) # fit has to happen only on train data

X_train_titles_bow = vectorizer.transform(X_train_preprocessed_titles)
X_cv_titles_bow = vectorizer.transform(X_cv_preprocessed_titles)
X_test_titles_bow = vectorizer.transform(X_test_preprocessed_titles)

print("After vectorizations")
print(X_train_titles_bow.shape, y_train.shape)
print(X_cv_titles_bow.shape, y_cv.shape)
print(X_test_titles_bow.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49039, 1988) (49039,)
(24155, 1988) (24155,)
(36051, 1988) (36051,)
==========================================================================
==========================
```

2.3.2.3 Vectorizing Text data: tfidf

In [177]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train_preprocessed_titles) # fit has to happen only on train data
X_train_titles_tfidf = vectorizer.transform(X_train_preprocessed_titles)
X_cv_titles_tfidf = vectorizer.transform(X_cv_preprocessed_titles)
X_test_titles_tfidf = vectorizer.transform(X_test_preprocessed_titles)

print("After vectorizations")
print(X_train_titles_tfidf.shape, y_train.shape)
print(X_cv_titles_tfidf.shape, y_cv.shape)
print(X_test_titles_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49039, 1988) (49039,)
(24155, 1988) (24155,)
(36051, 1988) (36051,)
==========================================================================
==========================
```

2.3.2.4 Vectorizing project_title data: avg w2v

In [178]:

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

In [179]:

```
# average Word2Vec for X_train
X_train_avg_w2v_vectors_titles = []; # the avg-w2v for each sentence/review is stored i
n this list
for sentence in tqdm(X_train_preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_train_avg_w2v_vectors_titles.append(vector)

print(len(X_train_avg_w2v_vectors_titles))
print(len(X_train_avg_w2v_vectors_titles[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
████| 49039/49039 [00:00<00:00, 57915.52it/s]

49039
300
```

In [180]:

```
# average Word2Vec for X_cv
X_cv_avg_w2v_vectors_titles = []; # the avg-w2v for each sentence/review is stored in t
his list
for sentence in tqdm(X_cv_preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_cv_avg_w2v_vectors_titles.append(vector)

print(len(X_cv_avg_w2v_vectors_titles))
print(len(X_cv_avg_w2v_vectors_titles[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
████| 24155/24155 [00:00<00:00, 51313.10it/s]

24155
300
```

In [181]:

```
# average Word2Vec for X_test
X_test_avg_w2v_vectors_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test_preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_test_avg_w2v_vectors_titles.append(vector)

print(len(X_test_avg_w2v_vectors_titles))
print(len(X_test_avg_w2v_vectors_titles[0]))
```

```
100%|█████████████████████████████████████████████████████████████████████
████| 36051/36051 [00:00<00:00, 56919.86it/s]

36051
300
```

2.3.2.5 Vectorizing project_title data: tfidf weighted w2v

In [182]:

```
#tfidf w2v for X_train

# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train_preprocessed_titles)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())

X_train_tfidf_w2v_vectors_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train_preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the 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_titles.append(vector)

print(len(X_train_tfidf_w2v_vectors_titles))
print(len(X_train_tfidf_w2v_vectors_titles[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
████| 49039/49039 [00:01<00:00, 26026.43it/s]


49039
300
```

In [183]:

```
#tfidf w2v for X_cv

# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_cv_preprocessed_titles)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())

X_cv_tfidf_w2v_vectors_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv_preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the 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_titles.append(vector)

print(len(X_cv_tfidf_w2v_vectors_titles))
print(len(X_cv_tfidf_w2v_vectors_titles[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
███| 24155/24155 [00:00<00:00, 25955.27it/s]

24155
300
```

In [184]:

```
#tfidf w2v for X_test

# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_test_preprocessed_titles)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())

X_test_tfidf_w2v_vectors_titles = []; # the avg-w2v for each sentence/review is stored
 in this list
for sentence in tqdm(X_test_preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting 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_titles.append(vector)

print(len(X_test_tfidf_w2v_vectors_titles))
print(len(X_test_tfidf_w2v_vectors_titles[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
████| 36051/36051 [00:01<00:00, 23791.88it/s]

36051
300
```

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

Concatinating all the features:model1-BOW

In [185]:

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

X_tr_m1 = hstack((X_train_categories_ohe, X_train_sub_categories_ohe, X_train_teacher_o
he, X_train_state_ohe, X_train_grade_ohe, X_train_price_norm, X_train_tnppp_norm, X_tra
in_text_bow, X_train_titles_bow))

X_cv_m1 = hstack((X_cv_categories_ohe, X_cv_sub_categories_ohe, X_cv_teacher_ohe, X_cv_
state_ohe, X_cv_grade_ohe, X_cv_price_norm, X_cv_tnppp_norm, X_cv_text_bow, X_cv_titles
_bow))

X_te_m1 = hstack((X_test_categories_ohe, X_test_sub_categories_ohe, X_test_teacher_ohe,
X_test_state_ohe, X_test_grade_ohe, X_test_price_norm, X_test_tnppp_norm, X_test_text_b
ow, X_test_titles_bow))

print("Final Data matrix")
print(X_tr_m1.shape, y_train.shape)
print(X_cv_m1.shape, y_cv.shape)
print(X_te_m1.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(49039, 14112) (49039,)
(24155, 14112) (24155,)
(36051, 14112) (36051,)
==========================================================================
==========================
```

Concatinating all the features:model2-TFIDF

In [186]:

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

X_tr_m2 = hstack((X_train_categories_ohe, X_train_sub_categories_ohe, X_train_teacher_o
he, X_train_state_ohe, X_train_grade_ohe, X_train_price_norm, X_train_tnppp_norm, X_tra
in_text_tfidf, X_train_titles_tfidf))

X_cv_m2 = hstack((X_cv_categories_ohe, X_cv_sub_categories_ohe, X_cv_teacher_ohe, X_cv_
state_ohe, X_cv_grade_ohe, X_cv_price_norm, X_cv_tnppp_norm, X_cv_text_tfidf, X_cv_titl
es_tfidf))

X_te_m2 = hstack((X_test_categories_ohe, X_test_sub_categories_ohe, X_test_teacher_ohe,
X_test_state_ohe, X_test_grade_ohe, X_test_price_norm, X_test_tnppp_norm, X_test_text_t
fidf, X_test_titles_tfidf))

print("Final Data matrix")
print(X_tr_m2.shape, y_train.shape)
print(X_cv_m2.shape, y_cv.shape)
print(X_te_m2.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(49039, 14112) (49039,)
(24155, 14112) (24155,)
(36051, 14112) (36051,)
==========================================================================
==========================
```

Concatinating all the features:model3-AVG W2V

In [187]:

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

X_tr_m3 = hstack((X_train_categories_ohe, X_train_sub_categories_ohe, X_train_teacher_o
he, X_train_state_ohe, X_train_grade_ohe, X_train_price_norm, X_train_tnppp_norm, X_tra
in_avg_w2v_vectors, X_train_avg_w2v_vectors_titles))

X_cv_m3 = hstack((X_cv_categories_ohe, X_cv_sub_categories_ohe, X_cv_teacher_ohe, X_cv_
state_ohe, X_cv_grade_ohe, X_cv_price_norm, X_cv_tnppp_norm, X_cv_avg_w2v_vectors, X_cv
_avg_w2v_vectors_titles))

X_te_m3 = hstack((X_test_categories_ohe, X_test_sub_categories_ohe, X_test_teacher_ohe,
X_test_state_ohe, X_test_grade_ohe, X_test_price_norm, X_test_tnppp_norm, X_test_avg_w2
v_vectors, X_test_avg_w2v_vectors_titles))

print("Final Data matrix")
print(X_tr_m3.shape, y_train.shape)
print(X_cv_m3.shape, y_cv.shape)
print(X_te_m3.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(49039, 701) (49039,)
(24155, 701) (24155,)
(36051, 701) (36051,)
==========================================================================
==========================
```

Concatinating all the features:model4-TFIDF WEIGHTED W2V

In [188]:

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

X_tr_m4 = hstack((X_train_categories_ohe, X_train_sub_categories_ohe, X_train_teacher_ohe, X_train_state_ohe, X_train_grade_ohe, X_train_price_norm, X_train_tnppp_norm, X_train_tfidf_w2v_vectors, X_train_tfidf_w2v_vectors_titles))

X_cv_m4 = hstack((X_cv_categories_ohe, X_cv_sub_categories_ohe, X_cv_teacher_ohe, X_cv_state_ohe, X_cv_grade_ohe, X_cv_price_norm, X_cv_tnppp_norm, X_cv_tfidf_w2v_vectors, X_cv_tfidf_w2v_vectors_titles))

X_te_m4 = hstack((X_test_categories_ohe, X_test_sub_categories_ohe, X_test_teacher_ohe, X_test_state_ohe, X_test_grade_ohe, X_test_price_norm, X_test_tnppp_norm, X_test_tfidf_w2v_vectors, X_test_tfidf_w2v_vectors_titles))

print("Final Data matrix")
print(X_tr_m4.shape, y_train.shape)
print(X_cv_m4.shape, y_cv.shape)
print(X_te_m4.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(49039, 701) (49039,)
(24155, 701) (24155,)
(36051, 701) (36051,)
====================================================================================================
```

2.4.1 Applying KNN brute force on BOW, SET 1

In [189]:

```
from scipy.sparse import hstack
from scipy.sparse import coo_matrix
from scipy.sparse import csr_matrix

X_tr_m1 = csr_matrix(X_tr_m1)
X_cv_m1 = csr_matrix(X_cv_m1)
X_te_m1 = csr_matrix(X_te_m1)

X_tr1_m1, X_tr2_m1, y_tr1, y_tr2 = train_test_split(X_tr_m1, y_train, test_size=0.33, s
tratify=y_train)
X_tr1_m1, X_tr3_m1, y_tr1, y_tr3 = train_test_split(X_tr1_m1, y_tr1, test_size=0.50, st
ratify=y_tr1)

X_cv1_m1, X_cv2_m1, y_cv1, y_cv2 = train_test_split(X_cv_m1, y_cv, test_size=0.33, stra
tify=y_cv)
X_cv1_m1, X_cv3_m1, y_cv1, y_cv3 = train_test_split(X_cv1_m1, y_cv1, test_size=0.50, st
ratify=y_cv1)

X_te1_m1, X_te2_m1, y_te1, y_te2 = train_test_split(X_te_m1, y_test, test_size=0.33, st
ratify=y_test)
X_te1_m1, X_te3_m1, y_te1, y_te3 = train_test_split(X_te1_m1, y_te1, test_size=0.50, st
ratify=y_te1)

print("Final Data1 matrix")
print(X_tr1_m1.shape, y_tr1.shape)
print(X_cv1_m1.shape, y_cv1.shape)
print(X_te1_m1.shape, y_te1.shape)
print("="*100)

print("Final Data2 matrix")
print(X_tr2_m1.shape, y_tr2.shape)
print(X_cv2_m1.shape, y_cv2.shape)
print(X_te2_m1.shape, y_te2.shape)
print("="*100)

print("Final Data3 matrix")
print(X_tr3_m1.shape, y_tr3.shape)
print(X_cv3_m1.shape, y_cv3.shape)
print(X_te3_m1.shape, y_te3.shape)
print("="*100)
```

```
Final Data1 matrix
(16428, 14112) (16428,)
(8091, 14112) (8091,)
(12077, 14112) (12077,)
==========================================================================
==========================
Final Data2 matrix
(16183, 14112) (16183,)
(7972, 14112) (7972,)
(11897, 14112) (11897,)
==========================================================================
==========================
Final Data3 matrix
(16428, 14112) (16428,)
(8092, 14112) (8092,)
(12077, 14112) (12077,)
==========================================================================
==========================
```

In [191]:

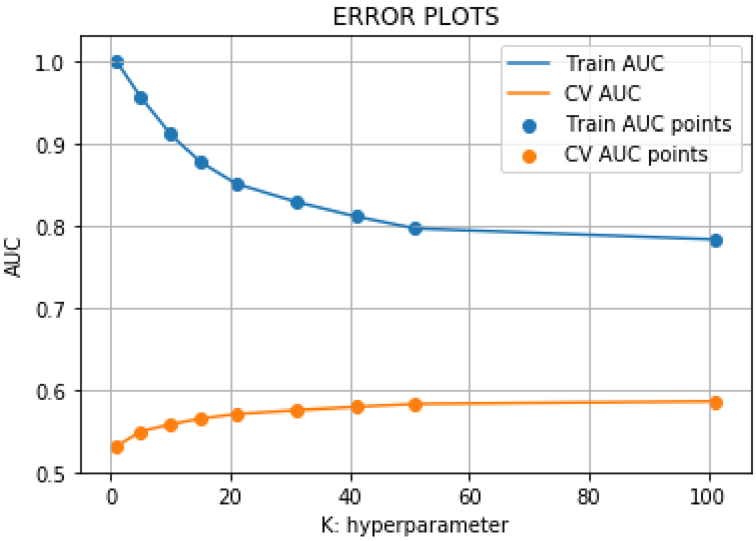
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
 of the positive class
    # not the predicted outputs

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

    return y_data_pred
```

In [192]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence va
lues, or non-thresholded measure of
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""
y_train_pred = []
y_tr = []
y_cv_pred = []
y_cr = []

train_auc = []
cv_auc = []

K = [1, 5, 10, 15, 21, 31, 41, 51, 101]
for i in K:
    neigh1 = KNeighborsClassifier(n_neighbors=i)
    neigh1.fit(X_tr1_m1, y_tr1)
    y_train_pred1 = batch_predict(neigh1, X_tr1_m1)
    y_cv_pred1 = batch_predict(neigh1, X_cv1_m1)

    neigh2 = KNeighborsClassifier(n_neighbors=i)
    neigh2.fit(X_tr2_m1, y_tr2)
    y_train_pred2 = batch_predict(neigh2, X_tr2_m1)
    y_cv_pred2 = batch_predict(neigh2, X_cv2_m1)

    neigh3 = KNeighborsClassifier(n_neighbors=i)
    neigh3.fit(X_tr3_m1, y_tr3)
    y_train_pred3 = batch_predict(neigh3, X_tr3_m1)
    y_cv_pred3 = batch_predict(neigh3, X_cv3_m1)

    y_train_pred.extend(y_train_pred1)
    y_train_pred.extend(y_train_pred2)
    y_train_pred.extend(y_train_pred3)
    y_tr.extend(y_tr1)
    y_tr.extend(y_tr2)
    y_tr.extend(y_tr3)

    y_cv_pred.extend(y_cv_pred1)
    y_cv_pred.extend(y_cv_pred2)
    y_cv_pred.extend(y_cv_pred3)
    y_cr.extend(y_cv1)
    y_cr.extend(y_cv2)
    y_cr.extend(y_cv3)

    # 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_tr,y_train_pred))
    cv_auc.append(roc_auc_score(y_cr, y_cv_pred))
```

In [193]:

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

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

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

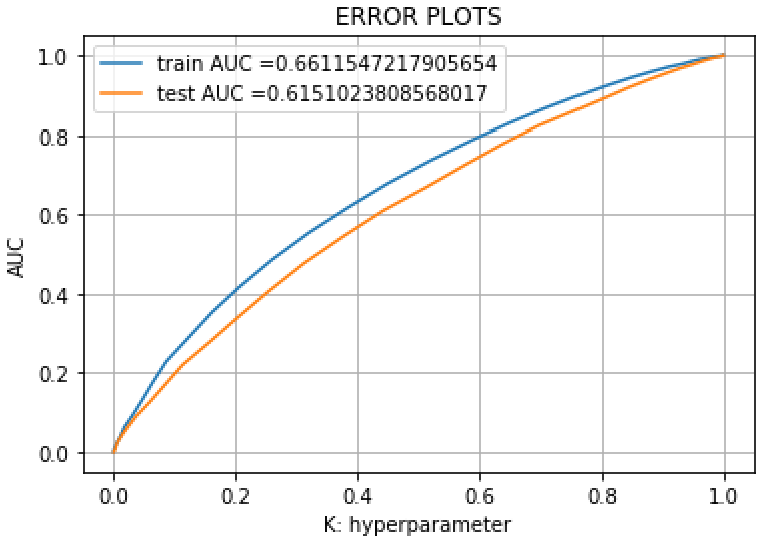


In [194]:

```
# from the error plot we choose K such that, we will have maximum AUC on cv data and ga
p between the train and cv is less
# Note: based on the method you use you might get different hyperparameter values as be
st one
# so, you choose according to the method you choose, you use gridsearch if you are havi
ng more computing power and note it will take more time
# if you increase the cv values in the GridSearchCV you will get more rebust results.

#here we are choosing the best_k based on forloop results
best_k = 101
```

In [195]:

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

y_train_pred = []
y_test_pred = []
y_te = []
y_tr = []
neigh1 = KNeighborsClassifier(n_neighbors=best_k)
neigh1.fit(X_tr1_m1, y_tr1)
y_train_pred1 = batch_predict(neigh1, X_tr1_m1)
y_test_pred1 = batch_predict(neigh1, X_te1_m1)

neigh2 = KNeighborsClassifier(n_neighbors=best_k)
neigh2.fit(X_tr2_m1, y_tr2)
y_train_pred2 = batch_predict(neigh2, X_tr2_m1)
y_test_pred2 = batch_predict(neigh2, X_te2_m1)

neigh3 = KNeighborsClassifier(n_neighbors=best_k)
neigh3.fit(X_tr3_m1, y_tr3)
y_train_pred3 = batch_predict(neigh3, X_tr3_m1)
y_test_pred3 = batch_predict(neigh3, X_te3_m1)

y_train_pred.extend(y_train_pred1)
y_train_pred.extend(y_train_pred2)
y_train_pred.extend(y_train_pred3)

y_test_pred.extend(y_test_pred1)
y_test_pred.extend(y_test_pred2)
y_test_pred.extend(y_test_pred3)

y_tr.extend(y_tr1)
y_tr.extend(y_tr2)
y_tr.extend(y_tr3)

y_te.extend(y_te1)
y_te.extend(y_te2)
y_te.extend(y_te3)


train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_te, 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()
```



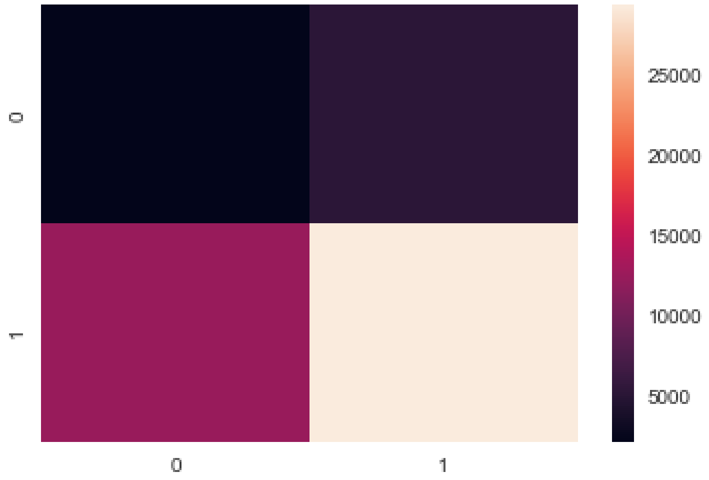
In [196]:

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

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

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

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

In [197]:

```
import seaborn as sns; sns.set()
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
con_mat_tr = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))
ax = sns.heatmap(con_mat_tr)
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

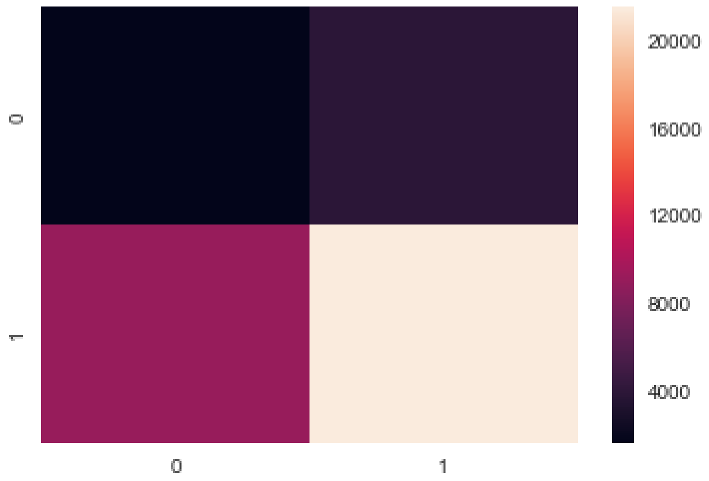
```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24962125860172998 for threshold 0.782
the maximum value of tpr*(1-fpr) 0.24962125860172998 for threshold 0.782
[[ 2166  5259]
 [12411 29203]]
```



In [198]:

```
print("Test confusion matrix")
con_mat_te = confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, tes
t_fpr))
ax = sns.heatmap(con_mat_te)
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr
)))
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24984254554451288 for threshold 0.782
the maximum value of tpr*(1-fpr) 0.24984254554451288 for threshold 0.782
[[ 1590  3869]
 [ 9086 21506]]
```



2.4.2 Applying KNN brute force on TFIDF, SET 2

In [199]:

```
from scipy.sparse import hstack
from scipy.sparse import coo_matrix
from scipy.sparse import csr_matrix

X_tr_m2 = csr_matrix(X_tr_m2)
X_cv_m2 = csr_matrix(X_cv_m2)
X_te_m2 = csr_matrix(X_te_m2)

X_tr1_m2, X_tr2_m2, y_tr1, y_tr2 = train_test_split(X_tr_m2, y_train, test_size=0.33, s
tratify=y_train)
X_tr1_m2, X_tr3_m2, y_tr1, y_tr3 = train_test_split(X_tr1_m2, y_tr1, test_size=0.50, st
ratify=y_tr1)

X_cv1_m2, X_cv2_m2, y_cv1, y_cv2 = train_test_split(X_cv_m2, y_cv, test_size=0.33, stra
tify=y_cv)
X_cv1_m2, X_cv3_m2, y_cv1, y_cv3 = train_test_split(X_cv1_m2, y_cv1, test_size=0.50, st
ratify=y_cv1)

X_te1_m2, X_te2_m2, y_te1, y_te2 = train_test_split(X_te_m2, y_test, test_size=0.33, st
ratify=y_test)
X_te1_m2, X_te3_m2, y_te1, y_te3 = train_test_split(X_te1_m2, y_te1, test_size=0.50, st
ratify=y_te1)

print("Final Data1 matrix")
print(X_tr1_m2.shape, y_tr1.shape)
print(X_cv1_m2.shape, y_cv1.shape)
print(X_te1_m2.shape, y_te1.shape)
print("="*100)

print("Final Data2 matrix")
print(X_tr2_m2.shape, y_tr2.shape)
print(X_cv2_m2.shape, y_cv2.shape)
print(X_te2_m2.shape, y_te2.shape)
print("="*100)

print("Final Data3 matrix")
print(X_tr3_m2.shape, y_tr3.shape)
print(X_cv3_m2.shape, y_cv3.shape)
print(X_te3_m2.shape, y_te3.shape)
print("="*100)
```

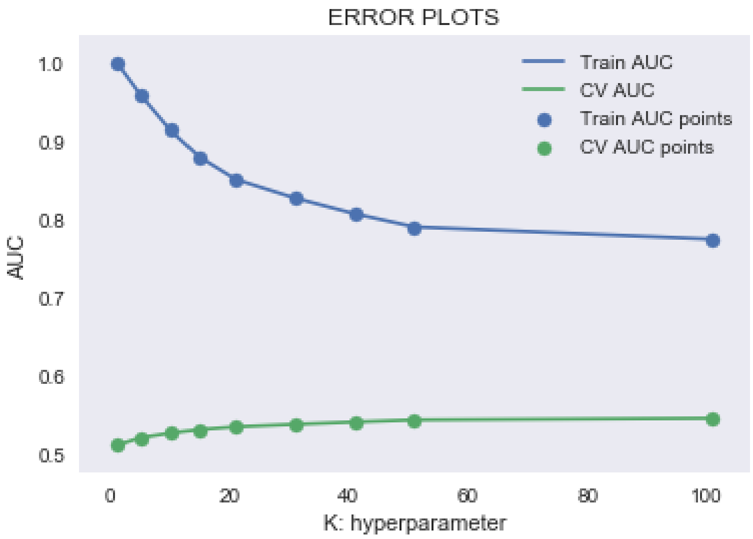
```
Final Data1 matrix
(16428, 14112) (16428,)
(8091, 14112) (8091,)
(12077, 14112) (12077,)
==========================================================================
==========================
Final Data2 matrix
(16183, 14112) (16183,)
(7972, 14112) (7972,)
(11897, 14112) (11897,)
==========================================================================
==========================
Final Data3 matrix
(16428, 14112) (16428,)
(8092, 14112) (8092,)
(12077, 14112) (12077,)
==========================================================================
==========================
```

In [200]:

```
y_train_pred = []
y_tr = []
y_cv_pred = []
y_cr = []

train_auc = []
cv_auc = []

K = [1, 5, 10, 15, 21, 31, 41, 51, 101]
for i in K:
    neigh1 = KNeighborsClassifier(n_neighbors=i)
    neigh1.fit(X_tr1_m2, y_tr1)
    y_train_pred1 = batch_predict(neigh1, X_tr1_m2)
    y_cv_pred1 = batch_predict(neigh1, X_cv1_m2)

    neigh2 = KNeighborsClassifier(n_neighbors=i)
    neigh2.fit(X_tr2_m2, y_tr2)
    y_train_pred2 = batch_predict(neigh2, X_tr2_m2)
    y_cv_pred2 = batch_predict(neigh2, X_cv2_m2)

    neigh3 = KNeighborsClassifier(n_neighbors=i)
    neigh3.fit(X_tr3_m2, y_tr3)
    y_train_pred3 = batch_predict(neigh3, X_tr3_m2)
    y_cv_pred3 = batch_predict(neigh3, X_cv3_m2)

    y_train_pred.extend(y_train_pred1)
    y_train_pred.extend(y_train_pred2)
    y_train_pred.extend(y_train_pred3)
    y_tr.extend(y_tr1)
    y_tr.extend(y_tr2)
    y_tr.extend(y_tr3)

    y_cv_pred.extend(y_cv_pred1)
    y_cv_pred.extend(y_cv_pred2)
    y_cv_pred.extend(y_cv_pred3)
    y_cr.extend(y_cv1)
    y_cr.extend(y_cv2)
    y_cr.extend(y_cv3)

    # 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_tr,y_train_pred))
    cv_auc.append(roc_auc_score(y_cr, y_cv_pred))
```

In [201]:

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

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

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

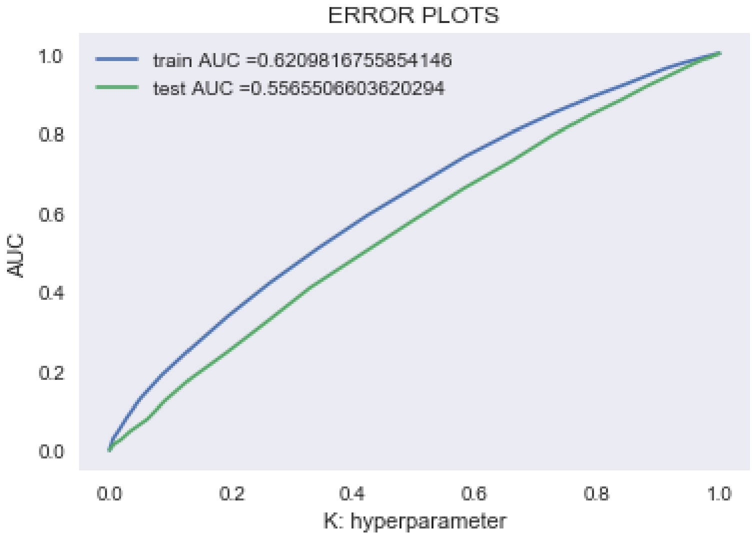


In [203]:

```
#here we are choosing the best_k based on forloop results
best_k = 101
```

In [204]:

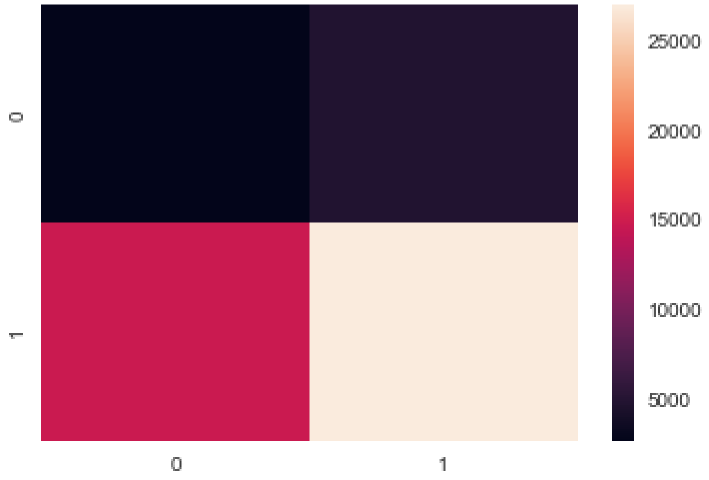
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#skle
arn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

y_train_pred = []
y_test_pred = []
y_te = []
y_tr = []
neigh1 = KNeighborsClassifier(n_neighbors=best_k)
neigh1.fit(X_tr1_m2, y_tr1)
y_train_pred1 = batch_predict(neigh1, X_tr1_m2)
y_test_pred1 = batch_predict(neigh1, X_te1_m2)

neigh2 = KNeighborsClassifier(n_neighbors=best_k)
neigh2.fit(X_tr2_m2, y_tr2)
y_train_pred2 = batch_predict(neigh2, X_tr2_m2)
y_test_pred2 = batch_predict(neigh2, X_te2_m2)

neigh3 = KNeighborsClassifier(n_neighbors=best_k)
neigh3.fit(X_tr3_m2, y_tr3)
y_train_pred3 = batch_predict(neigh3, X_tr3_m2)
y_test_pred3 = batch_predict(neigh3, X_te3_m2)

y_train_pred.extend(y_train_pred1)
y_train_pred.extend(y_train_pred2)
y_train_pred.extend(y_train_pred3)

y_test_pred.extend(y_test_pred1)
y_test_pred.extend(y_test_pred2)
y_test_pred.extend(y_test_pred3)

y_tr.extend(y_tr1)
y_tr.extend(y_tr2)
y_tr.extend(y_tr3)

y_te.extend(y_te1)
y_te.extend(y_te2)
y_te.extend(y_te3)


train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_te, 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 [205]:

```
import seaborn as sns; sns.set()
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
con_mat_tr = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr,
train_fpr))
ax = sns.heatmap(con_mat_tr)
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_f
pr)))
```

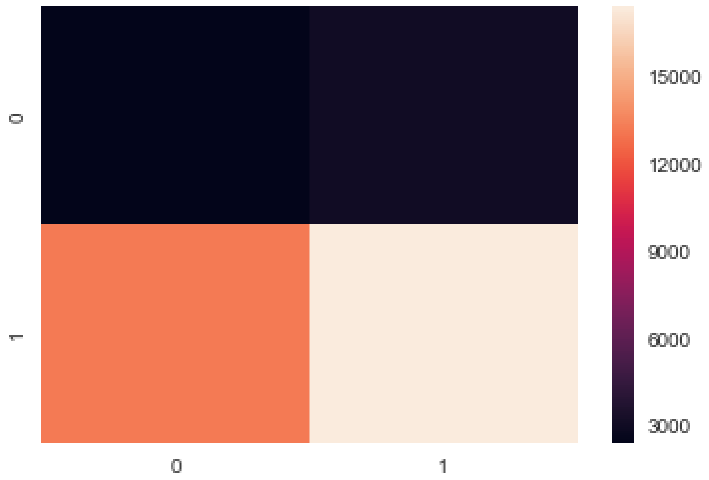
```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24994412815018877 for threshold 0.842
the maximum value of tpr*(1-fpr) 0.24994412815018877 for threshold 0.842
[[ 2648  4777]
 [14722 26892]]
```



In [206]:

```
print("Test confusion matrix")
con_mat_te = confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, tes
t_fpr))
ax = sns.heatmap(con_mat_te)
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr
)))
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999757555875987 for threshold 0.851
the maximum value of tpr*(1-fpr) 0.24999757555875987 for threshold 0.851
[[ 2382  3077]
 [13206 17386]]
```



2.4.3 Applying KNN brute force on AVG W2V, SET 3

In [88]:

```
#KNN brute force on AVG W2V(SET 3) are performed on 30k datapoints because computation
 time is very high om 100k datapoints
from scipy.sparse import coo_matrix
from scipy.sparse import csr_matrix

X_tr_m3 = csr_matrix(X_tr_m3)
X_cv_m3 = csr_matrix(X_cv_m3)
X_te_m3 = csr_matrix(X_te_m3)

X_tr1_m3, X_tr2_m3, y_tr1, y_tr2 = train_test_split(X_tr_m3, y_train, test_size=0.33, s
tratify=y_train)
X_tr1_m3, X_tr3_m3, y_tr1, y_tr3 = train_test_split(X_tr1_m3, y_tr1, test_size=0.50, st
ratify=y_tr1)

X_cv1_m3, X_cv2_m3, y_cv1, y_cv2 = train_test_split(X_cv_m3, y_cv, test_size=0.33, stra
tify=y_cv)
X_cv1_m3, X_cv3_m3, y_cv1, y_cv3 = train_test_split(X_cv1_m3, y_cv1, test_size=0.50, st
ratify=y_cv1)

X_te1_m3, X_te2_m3, y_te1, y_te2 = train_test_split(X_te_m3, y_test, test_size=0.33, st
ratify=y_test)
X_te1_m3, X_te3_m3, y_te1, y_te3 = train_test_split(X_te1_m3, y_te1, test_size=0.50, st
ratify=y_te1)

print("Final Data1 matrix")
print(X_tr1_m3.shape, y_tr1.shape)
print(X_cv1_m3.shape, y_cv1.shape)
print(X_te1_m3.shape, y_te1.shape)
print("="*100)

print("Final Data2 matrix")
print(X_tr2_m3.shape, y_tr2.shape)
print(X_cv2_m3.shape, y_cv2.shape)
print(X_te2_m3.shape, y_te2.shape)
print("="*100)

print("Final Data3 matrix")
print(X_tr3_m3.shape, y_tr3.shape)
print(X_cv3_m3.shape, y_cv3.shape)
print(X_te3_m3.shape, y_te3.shape)
print("="*100)
```

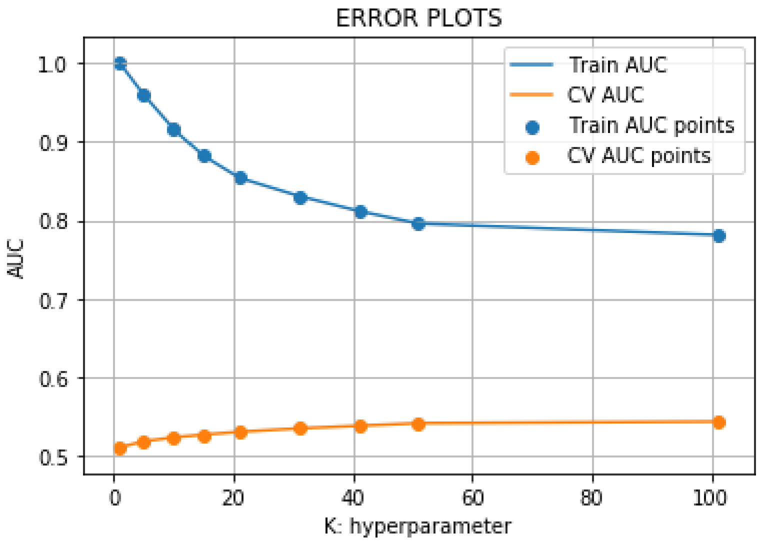
```
Final Data1 matrix
(4511, 700) (4511,)
(2222, 700) (2222,)
(3316, 700) (3316,)
==========================================================================
==========================
Final Data2 matrix
(4445, 700) (4445,)
(2189, 700) (2189,)
(3267, 700) (3267,)
==========================================================================
==========================
Final Data3 matrix
(4511, 700) (4511,)
(2222, 700) (2222,)
(3317, 700) (3317,)
==========================================================================
==========================
```

In [89]:

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

y_train_pred = []
y_tr = []
y_cv_pred = []
y_cr = []

y_train_pred1 = []
y_train_pred2 = []
y_train_pred3 = []

y_cv_pred1 = []
y_cv_pred2 = []
y_cv_pred3 = []

train_auc = []
cv_auc = []

K = [1, 5, 10, 15, 21, 31, 41, 51, 101]
for i in K:
    neigh1 = KNeighborsClassifier(n_neighbors=i)
    neigh1.fit(X_tr1_m3, y_tr1)
    y_train_pred1 = batch_predict(neigh1, X_tr1_m3)
    y_cv_pred1 = batch_predict(neigh1, X_cv1_m3)

    neigh2 = KNeighborsClassifier(n_neighbors=i)
    neigh2.fit(X_tr2_m3, y_tr2)
    y_train_pred2 = batch_predict(neigh2, X_tr2_m3)
    y_cv_pred2 = batch_predict(neigh2, X_cv2_m3)

    neigh3 = KNeighborsClassifier(n_neighbors=i)
    neigh3.fit(X_tr3_m3, y_tr3)
    y_train_pred3 = batch_predict(neigh3, X_tr3_m3)
    y_cv_pred3 = batch_predict(neigh3, X_cv3_m3)

    y_train_pred.extend(y_train_pred1)
    y_train_pred.extend(y_train_pred2)
    y_train_pred.extend(y_train_pred3)
    y_tr.extend(y_tr1)
    y_tr.extend(y_tr2)
    y_tr.extend(y_tr3)

    y_cv_pred.extend(y_cv_pred1)
    y_cv_pred.extend(y_cv_pred2)
    y_cv_pred.extend(y_cv_pred3)
    y_cr.extend(y_cv1)
    y_cr.extend(y_cv2)
    y_cr.extend(y_cv3)

    # 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_tr,y_train_pred))
    cv_auc.append(roc_auc_score(y_cr, y_cv_pred))
```

In [90]:

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

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

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

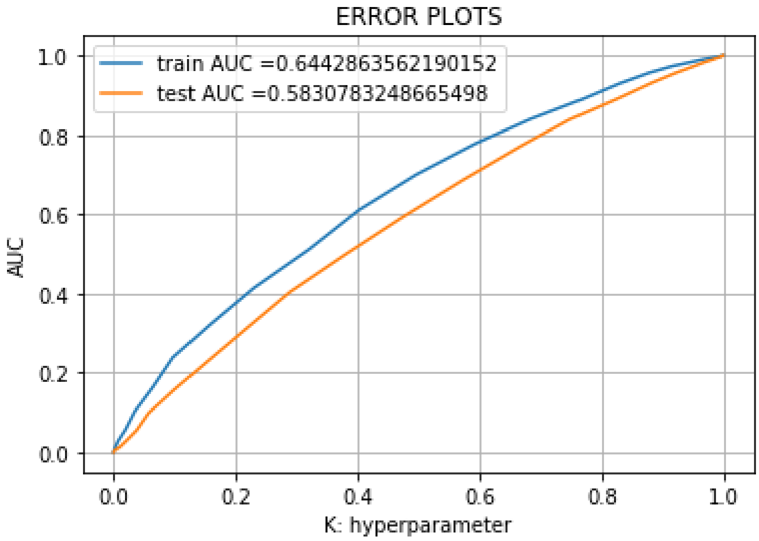


In [91]:

```
#here we are choosing the best_k based on forloop results
best_k = 101
```

In [92]:

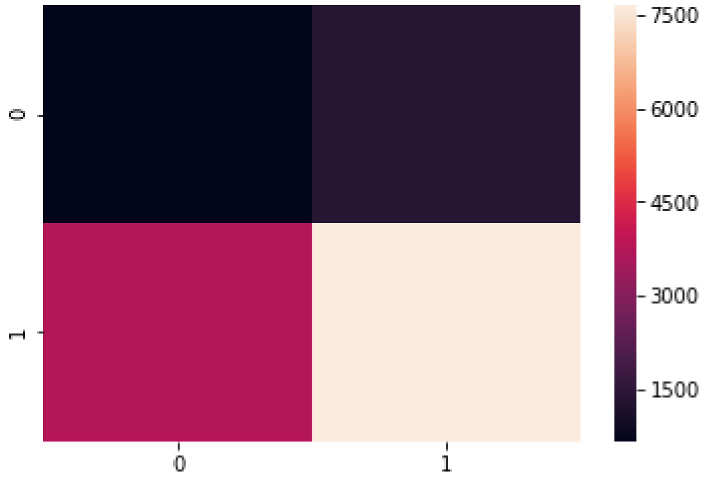
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#skle
arn.metrics.roc_curve

y_train_pred = []
y_test_pred = []
y_te = []
y_tr = []
neigh1 = KNeighborsClassifier(n_neighbors=best_k)
neigh1.fit(X_tr1_m3, y_tr1)
y_train_pred1 = batch_predict(neigh1, X_tr1_m3)
y_test_pred1 = batch_predict(neigh1, X_te1_m3)

neigh2 = KNeighborsClassifier(n_neighbors=best_k)
neigh2.fit(X_tr2_m3, y_tr2)
y_train_pred2 = batch_predict(neigh2, X_tr2_m3)
y_test_pred2 = batch_predict(neigh2, X_te2_m3)

neigh3 = KNeighborsClassifier(n_neighbors=best_k)
neigh3.fit(X_tr3_m3, y_tr3)
y_train_pred3 = batch_predict(neigh3, X_tr3_m3)
y_test_pred3 = batch_predict(neigh3, X_te3_m3)

y_train_pred.extend(y_train_pred1)
y_train_pred.extend(y_train_pred2)
y_train_pred.extend(y_train_pred3)

y_test_pred.extend(y_test_pred1)
y_test_pred.extend(y_test_pred2)
y_test_pred.extend(y_test_pred3)

y_tr.extend(y_tr1)
y_tr.extend(y_tr2)
y_tr.extend(y_tr3)

y_te.extend(y_te1)
y_te.extend(y_te2)
y_te.extend(y_te3)


train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_te, 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 [95]:

```
import seaborn as sns
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
con_mat_tr = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr,
train_fpr))
ax = sns.heatmap(con_mat_tr)
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_f
pr)))
```

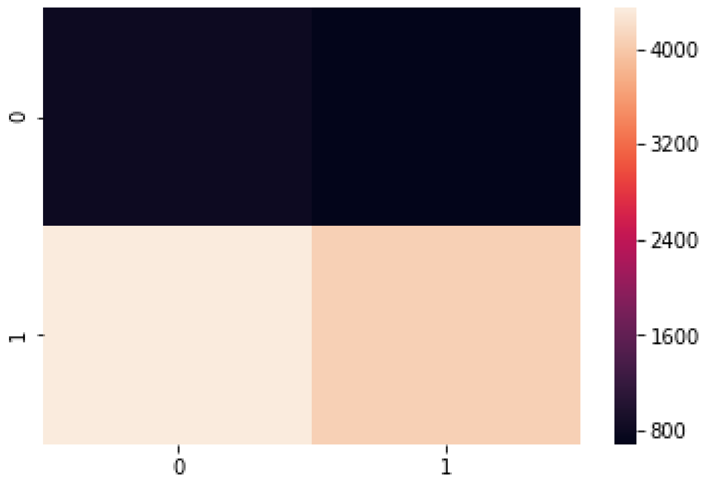
```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24998630963492605 for threshold 0.851
the maximum value of tpr*(1-fpr) 0.24998630963492605 for threshold 0.851
[[ 656 1371]
 [3795 7645]]
```



In [96]:

```
print("Test confusion matrix")
con_mat_te = confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr))
ax = sns.heatmap(con_mat_te)
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24940071909210682 for threshold 0.871
the maximum value of tpr*(1-fpr) 0.24940071909210682 for threshold 0.871
[[ 806  685]
 [4339 4070]]
```



2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

In [97]:

```
#KNN brute force on TFIDF W2V(SET 4) are performed on 30k datapoints because computation time is very high om 100k datapoints
from scipy.sparse import hstack
from scipy.sparse import coo_matrix
from scipy.sparse import csr_matrix

X_tr_m4 = csr_matrix(X_tr_m4)
X_cv_m4 = csr_matrix(X_cv_m4)
X_te_m4 = csr_matrix(X_te_m4)

X_tr1_m1, X_tr2_m1, y_tr1, y_tr2 = train_test_split(X_tr_m4, y_train, test_size=0.33, stratify=y_train)
X_tr1_m1, X_tr3_m1, y_tr1, y_tr3 = train_test_split(X_tr1_m1, y_tr1, test_size=0.50, stratify=y_tr1)

X_cv1_m1, X_cv2_m1, y_cv1, y_cv2 = train_test_split(X_cv_m4, y_cv, test_size=0.33, stratify=y_cv)
X_cv1_m1, X_cv3_m1, y_cv1, y_cv3 = train_test_split(X_cv1_m1, y_cv1, test_size=0.50, stratify=y_cv1)

X_te1_m1, X_te2_m1, y_te1, y_te2 = train_test_split(X_te_m4, y_test, test_size=0.33, stratify=y_test)
X_te1_m1, X_te3_m1, y_te1, y_te3 = train_test_split(X_te1_m1, y_te1, test_size=0.50, stratify=y_te1)

print("Final Data1 matrix")
print(X_tr1_m1.shape, y_tr1.shape)
print(X_cv1_m1.shape, y_cv1.shape)
print(X_te1_m1.shape, y_te1.shape)
print("="*100)

print("Final Data2 matrix")
print(X_tr2_m1.shape, y_tr2.shape)
print(X_cv2_m1.shape, y_cv2.shape)
print(X_te2_m1.shape, y_te2.shape)
print("="*100)

print("Final Data3 matrix")
print(X_tr3_m1.shape, y_tr3.shape)
print(X_cv3_m1.shape, y_cv3.shape)
print(X_te3_m1.shape, y_te3.shape)
print("="*100)
```

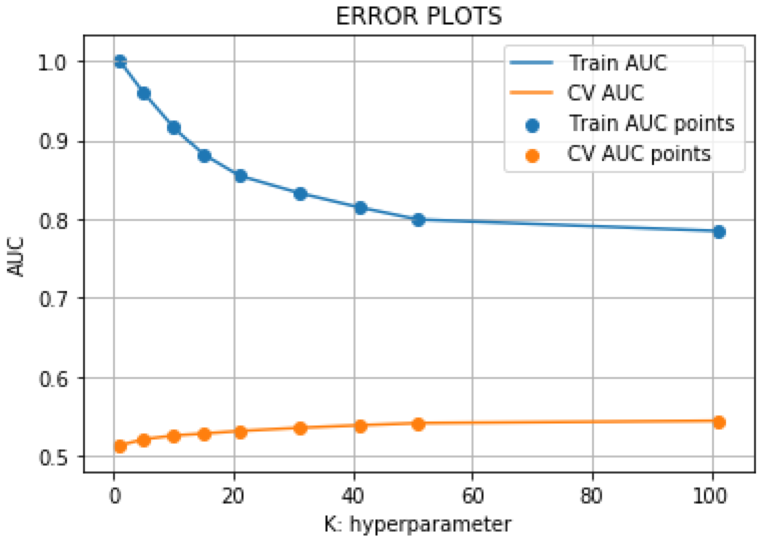
```
Final Data1 matrix
(4511, 700) (4511,)
(2222, 700) (2222,)
(3316, 700) (3316,)
==========================================================================
==========================
Final Data2 matrix
(4445, 700) (4445,)
(2189, 700) (2189,)
(3267, 700) (3267,)
==========================================================================
==========================
Final Data3 matrix
(4511, 700) (4511,)
(2222, 700) (2222,)
(3317, 700) (3317,)
==========================================================================
==========================
```

In [98]:

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

y_train_pred = []
y_tr = []
y_cv_pred = []
y_cr = []

train_auc = []
cv_auc = []

K = [1, 5, 10, 15, 21, 31, 41, 51, 101]
for i in K:
    neigh1 = KNeighborsClassifier(n_neighbors=i)
    neigh1.fit(X_tr1_m1, y_tr1)
    y_train_pred1 = batch_predict(neigh1, X_tr1_m1)
    y_cv_pred1 = batch_predict(neigh1, X_cv1_m1)

    neigh2 = KNeighborsClassifier(n_neighbors=i)
    neigh2.fit(X_tr2_m1, y_tr2)
    y_train_pred2 = batch_predict(neigh2, X_tr2_m1)
    y_cv_pred2 = batch_predict(neigh2, X_cv2_m1)

    neigh3 = KNeighborsClassifier(n_neighbors=i)
    neigh3.fit(X_tr3_m1, y_tr3)
    y_train_pred3 = batch_predict(neigh3, X_tr3_m1)
    y_cv_pred3 = batch_predict(neigh3, X_cv3_m1)

    y_train_pred.extend(y_train_pred1)
    y_train_pred.extend(y_train_pred2)
    y_train_pred.extend(y_train_pred3)
    y_tr.extend(y_tr1)
    y_tr.extend(y_tr2)
    y_tr.extend(y_tr3)

    y_cv_pred.extend(y_cv_pred1)
    y_cv_pred.extend(y_cv_pred2)
    y_cv_pred.extend(y_cv_pred3)
    y_cr.extend(y_cv1)
    y_cr.extend(y_cv2)
    y_cr.extend(y_cv3)

    # 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_tr,y_train_pred))
    cv_auc.append(roc_auc_score(y_cr, y_cv_pred))
```

In [99]:

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

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

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

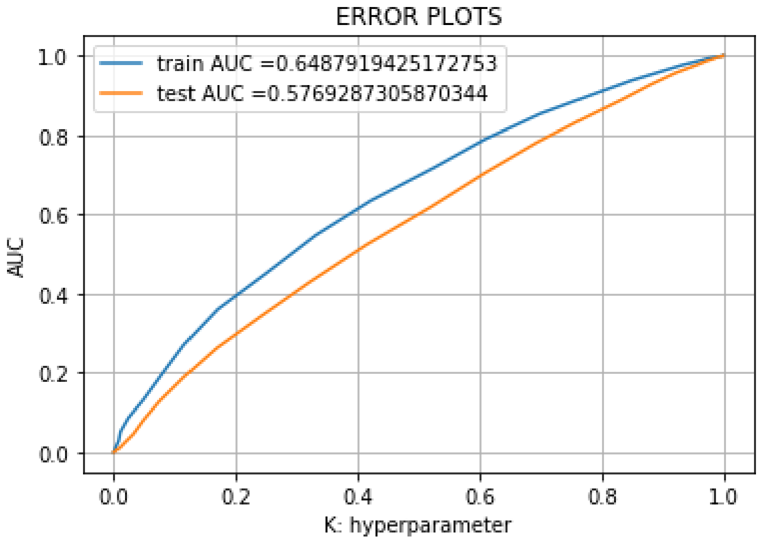


In [100]:

```
#here we are choosing the best_k based on forloop results
best_k = 101
```

In [101]:

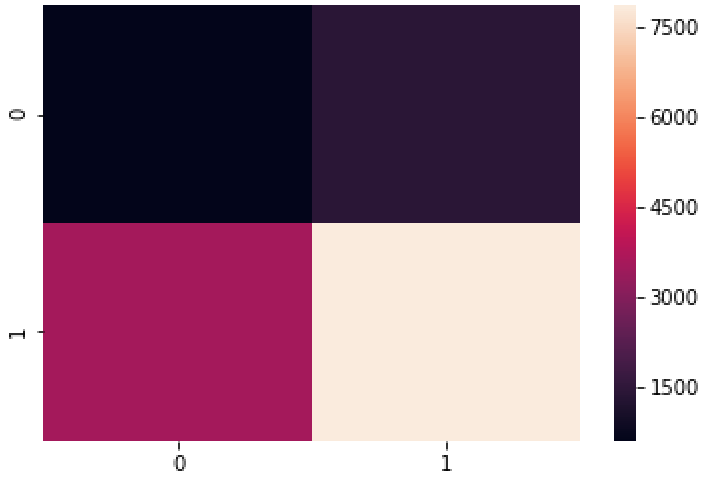
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#skle
arn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

y_train_pred = []
y_test_pred = []
y_te = []
y_tr = []
neigh1 = KNeighborsClassifier(n_neighbors=best_k)
neigh1.fit(X_tr1_m1, y_tr1)
y_train_pred1 = batch_predict(neigh1, X_tr1_m1)
y_test_pred1 = batch_predict(neigh1, X_te1_m1)

neigh2 = KNeighborsClassifier(n_neighbors=best_k)
neigh2.fit(X_tr2_m1, y_tr2)
y_train_pred2 = batch_predict(neigh2, X_tr2_m1)
y_test_pred2 = batch_predict(neigh2, X_te2_m1)

neigh3 = KNeighborsClassifier(n_neighbors=best_k)
neigh3.fit(X_tr3_m1, y_tr3)
y_train_pred3 = batch_predict(neigh3, X_tr3_m1)
y_test_pred3 = batch_predict(neigh3, X_te3_m1)

y_train_pred.extend(y_train_pred1)
y_train_pred.extend(y_train_pred2)
y_train_pred.extend(y_train_pred3)

y_test_pred.extend(y_test_pred1)
y_test_pred.extend(y_test_pred2)
y_test_pred.extend(y_test_pred3)

y_tr.extend(y_tr1)
y_tr.extend(y_tr2)
y_tr.extend(y_tr3)

y_te.extend(y_te1)
y_te.extend(y_te2)
y_te.extend(y_te3)


train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_te, 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 [102]:

```
import seaborn as sns
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
con_mat_tr = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr,
train_fpr))
ax = sns.heatmap(con_mat_tr)
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_f
pr)))
```

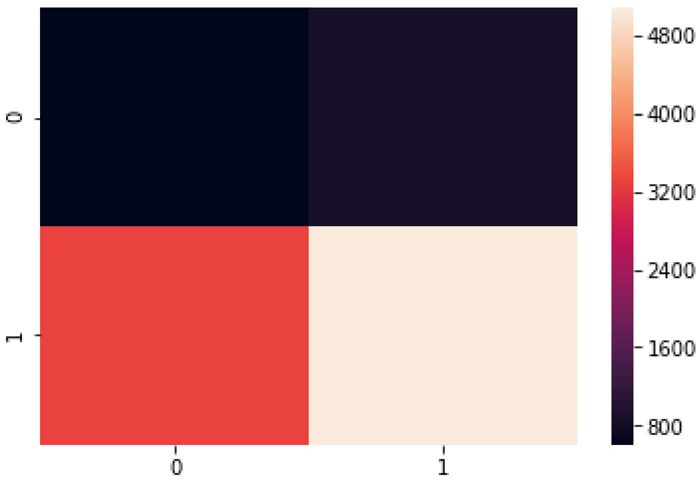
```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24935448407524566 for threshold 0.842
the maximum value of tpr*(1-fpr) 0.24935448407524566 for threshold 0.842
[[ 611 1416]
 [3581 7859]]
```



In [103]:

```
print("Test confusion matrix")
con_mat_te = confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, tes
t_fpr))
ax = sns.heatmap(con_mat_te)
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr
)))
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2496085387801884 for threshold 0.851
the maximum value of tpr*(1-fpr) 0.2496085387801884 for threshold 0.851
[[ 602  889]
 [3325 5084]]
```



2.5 Feature selection with `SelectKBest`

In [210]:

```
from sklearn.feature_selection import SelectKBest, chi2

print(X_tr_m2.shape)
print(X_cv_m2.shape)
print(X_te_m2.shape)
print('='*50)
sel = SelectKBest(chi2, k=2000).fit(X_tr_m2, y_train)
X_tr_m2_new = sel.transform(X_tr_m2)
X_cv_m2_new = sel.transform(X_cv_m2)
X_te_m2_new = sel.transform(X_te_m2)
print(X_tr_m2_new.shape)
print(X_cv_m2_new.shape)
print(X_te_m2_new.shape)
```

```
(49039, 14112)
(24155, 14112)
(36051, 14112)
==================================================
(49039, 2000)
(24155, 2000)
(36051, 2000)
```

In [211]:

```
X_tr1_m2, X_tr2_m2, y_tr1, y_tr2 = train_test_split(X_tr_m2_new, y_train, test_size=0.3
3, stratify=y_train)
X_tr1_m2, X_tr3_m2, y_tr1, y_tr3 = train_test_split(X_tr1_m2, y_tr1, test_size=0.50, st
ratify=y_tr1)

X_cv1_m2, X_cv2_m2, y_cv1, y_cv2 = train_test_split(X_cv_m2_new, y_cv, test_size=0.33,
stratify=y_cv)
X_cv1_m2, X_cv3_m2, y_cv1, y_cv3 = train_test_split(X_cv1_m2, y_cv1, test_size=0.50, st
ratify=y_cv1)

X_te1_m2, X_te2_m2, y_te1, y_te2 = train_test_split(X_te_m2_new, y_test, test_size=0.33
, stratify=y_test)
X_te1_m2, X_te3_m2, y_te1, y_te3 = train_test_split(X_te1_m2, y_te1, test_size=0.50, st
ratify=y_te1)

print("Final Data1 matrix")
print(X_tr1_m2.shape, y_tr1.shape)
print(X_cv1_m2.shape, y_cv1.shape)
print(X_te1_m2.shape, y_te1.shape)
print("="*100)

print("Final Data2 matrix")
print(X_tr2_m2.shape, y_tr2.shape)
print(X_cv2_m2.shape, y_cv2.shape)
print(X_te2_m2.shape, y_te2.shape)
print("="*100)

print("Final Data3 matrix")
print(X_tr3_m2.shape, y_tr3.shape)
print(X_cv3_m2.shape, y_cv3.shape)
print(X_te3_m2.shape, y_te3.shape)
print("="*100)
```

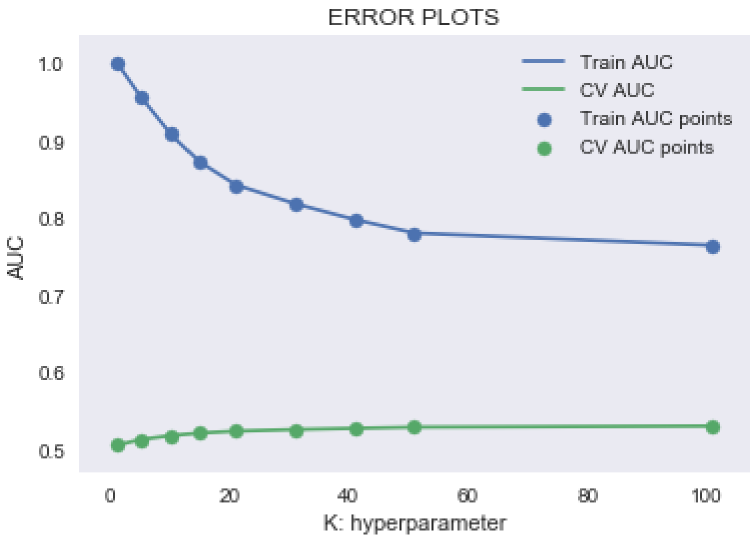
```
Final Data1 matrix
(16428, 2000) (16428,)
(8091, 2000) (8091,)
(12077, 2000) (12077,)
===========================================================================
==========================
Final Data2 matrix
(16183, 2000) (16183,)
(7972, 2000) (7972,)
(11897, 2000) (11897,)
===========================================================================
==========================
Final Data3 matrix
(16428, 2000) (16428,)
(8092, 2000) (8092,)
(12077, 2000) (12077,)
===========================================================================
==========================
```

In [212]:

```
y_train_pred = []
y_tr = []
y_cv_pred = []
y_cr = []

train_auc = []
cv_auc = []

K = [1, 5, 10, 15, 21, 31, 41, 51, 101]
for i in K:
    neigh1 = KNeighborsClassifier(n_neighbors=i)
    neigh1.fit(X_tr1_m2, y_tr1)
    y_train_pred1 = batch_predict(neigh1, X_tr1_m2)
    y_cv_pred1 = batch_predict(neigh1, X_cv1_m2)

    neigh2 = KNeighborsClassifier(n_neighbors=i)
    neigh2.fit(X_tr2_m2, y_tr2)
    y_train_pred2 = batch_predict(neigh2, X_tr2_m2)
    y_cv_pred2 = batch_predict(neigh2, X_cv2_m2)

    neigh3 = KNeighborsClassifier(n_neighbors=i)
    neigh3.fit(X_tr3_m2, y_tr3)
    y_train_pred3 = batch_predict(neigh3, X_tr3_m2)
    y_cv_pred3 = batch_predict(neigh3, X_cv3_m2)

    y_train_pred.extend(y_train_pred1)
    y_train_pred.extend(y_train_pred2)
    y_train_pred.extend(y_train_pred3)
    y_tr.extend(y_tr1)
    y_tr.extend(y_tr2)
    y_tr.extend(y_tr3)

    y_cv_pred.extend(y_cv_pred1)
    y_cv_pred.extend(y_cv_pred2)
    y_cv_pred.extend(y_cv_pred3)
    y_cr.extend(y_cv1)
    y_cr.extend(y_cv2)
    y_cr.extend(y_cv3)

    # 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_tr,y_train_pred))
    cv_auc.append(roc_auc_score(y_cr, y_cv_pred))
```

In [213]:

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

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

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

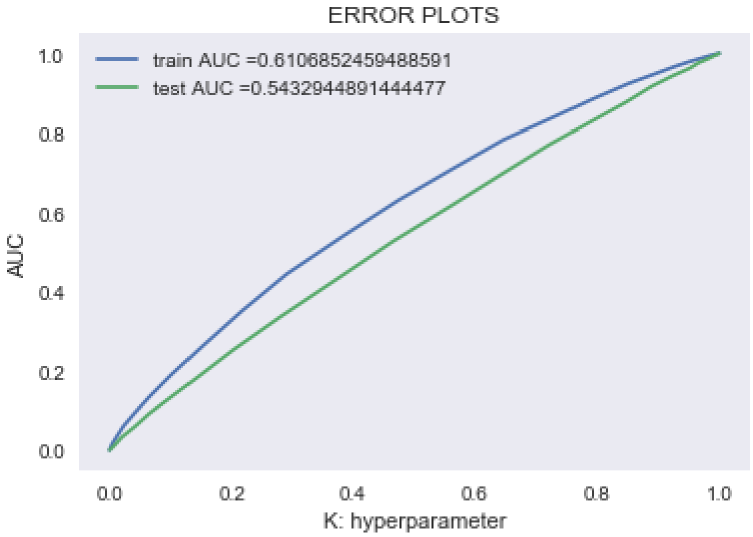


In [214]:

```
#here we are choosing the best_k based on forloop results
best_k = 101
```

In [215]:

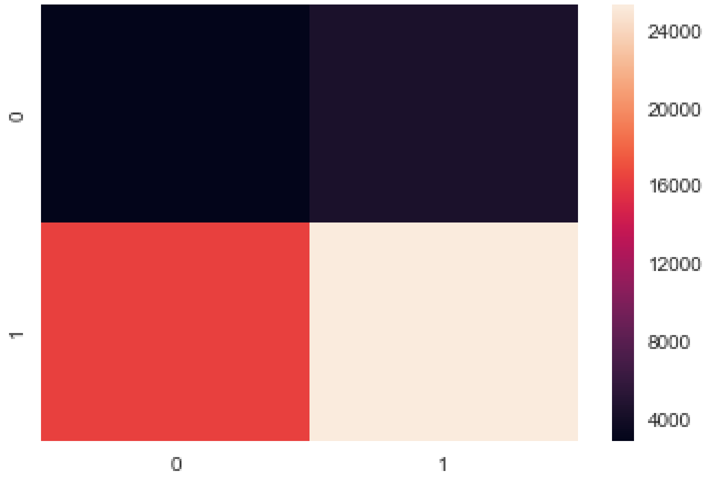
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#skle
arn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

y_train_pred = []
y_test_pred = []
y_te = []
y_tr = []
neigh1 = KNeighborsClassifier(n_neighbors=best_k)
neigh1.fit(X_tr1_m2, y_tr1)
y_train_pred1 = batch_predict(neigh1, X_tr1_m2)
y_test_pred1 = batch_predict(neigh1, X_te1_m2)

neigh2 = KNeighborsClassifier(n_neighbors=best_k)
neigh2.fit(X_tr2_m2, y_tr2)
y_train_pred2 = batch_predict(neigh2, X_tr2_m2)
y_test_pred2 = batch_predict(neigh2, X_te2_m2)

neigh3 = KNeighborsClassifier(n_neighbors=best_k)
neigh3.fit(X_tr3_m2, y_tr3)
y_train_pred3 = batch_predict(neigh3, X_tr3_m2)
y_test_pred3 = batch_predict(neigh3, X_te3_m2)

y_train_pred.extend(y_train_pred1)
y_train_pred.extend(y_train_pred2)
y_train_pred.extend(y_train_pred3)

y_test_pred.extend(y_test_pred1)
y_test_pred.extend(y_test_pred2)
y_test_pred.extend(y_test_pred3)

y_tr.extend(y_tr1)
y_tr.extend(y_tr2)
y_tr.extend(y_tr3)

y_te.extend(y_te1)
y_te.extend(y_te2)
y_te.extend(y_te3)


train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_te, 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 [216]:

```
import seaborn as sns
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
con_mat_tr = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr,
train_fpr))
ax = sns.heatmap(con_mat_tr)
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_f
pr)))
```

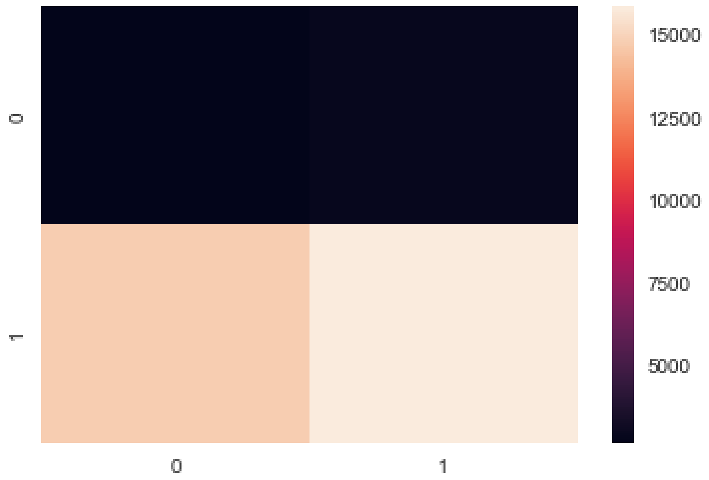
```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2492996224875013 for threshold 0.842
the maximum value of tpr*(1-fpr) 0.2492996224875013 for threshold 0.842
[[ 2894  4531]
 [16395 25219]]
```



In [217]:

```
print("Test confusion matrix")
con_mat_te = confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, tes
t_fpr))
ax = sns.heatmap(con_mat_te)
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr
)))
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2487822342046479 for threshold 0.851
the maximum value of tpr*(1-fpr) 0.2487822342046479 for threshold 0.851
[[ 2626  2833]
 [14758 15834]]
```



3. Conclusions

In [218]:

```
from prettytable import PrettyTable

x = PrettyTable()

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

x.add_row(["BOW", "Brute", 101, 0.61])
x.add_row(["TFIDF", "Brute", 101, 0.55])
x.add_row(["W2V", "Brute", 101, 0.58])
x.add_row(["TFIDFW2V", "Brute", 101, 0.57])
print(x)
```

```
+------------+-------+-----------------+------+
| Vectorizer | Model | Hyper Parameter | AUC  |
+------------+-------+-----------------+------+
|    BOW     | Brute |       101       | 0.61 |
|   TFIDF    | Brute |       101       | 0.55 |
|    W2V     | Brute |       101       | 0.58 |
|  TFIDFW2V  | Brute |       101       | 0.57 |
+------------+-------+-----------------+------+
```