DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`

Feature	Description
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`

Feature	Description
school_state	State where school is located (Two-letter U.S. postal code). **Example:** WY
project_subject_subcategories	One or more (comma-separated) subject subcategories for the project. **Examples:** • Literacy • Literature & Writing, Social Sciences
project_resource_summary	An explanation of the resources needed for the project. **Example:** • My students need hands on literacy materials to manage sensory needs!
project_essay_1	First application essay*
project_essay_2	Second application essay*
project_essay_3	Third application essay*
project_essay_4	Fourth application essay*
project_submitted_datetime	Datetime when project application was submitted. **Example:** 2016-04-28 12:43:56.245
teacher_id	A unique identifier for the teacher of the proposed project. **Example:** bdf8baa8fedef6bfeec7ae4ff1c15c56

Feature	Description
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same teacher. **Example:** 2

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

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

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

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

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

Label	Description

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 [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
```

```
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

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

from tqdm import tqdm
import os

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

1.1 Reading Data

In [2]:
```
project_data = pd.read_csv('C:/Users/pramod reddy chandi/Desktop/pram/applied ai course/DonorsChoose_2018/train_data.csv')
```

```
resource_data = pd.read_csv('C:/Users/pramod reddy chandi/Desktop/pram/
applied ai course/DonorsChoose_2018/resources.csv')
```

In [3]:
```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
```

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

In [4]:
```
# how to replace elements in list python: https://stackoverflow.com/a/2
582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(
project_data.columns)]


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


# how to reorder columns pandas python: https://stackoverflow.com/a/131
48611/4084039
project_data = project_data[cols]


project_data.head(2)
```

Out[4]:

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

In [5]:
```
print("Number of data points in train data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

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

Out[5]:

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

1.2 preprocessing of `project_subject_categories`

In [6]:
```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stacko
verflow.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
```

```
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [7]:
```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039

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

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

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

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

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

1.3 Text preprocessing

In [8]:
```
# 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 [9]:
```
project_data.head(2)
```

Out[9]:

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

In [10]:
```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

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)

In [11]:
```
# 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 [12]:
```
sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

```
\"A person is a person, no matter how small.\" (Dr.Seuss) I teach the s
mallest students with the biggest enthusiasm for learning. My students
learn in many different ways using all of our senses and multiple intel
ligences. I use a wide range of techniques to help all my students succ
eed. \r\nStudents in my class come from a variety of different backgrou
nds which makes for wonderful sharing of experiences and cultures, incl
```

```
uding Native Americans.\r\nOur school is a caring community of successf
ul learners which can be seen through collaborative student project bas
ed learning in and out of the classroom. Kindergarteners in my class lo
ve to work with hands-on materials and have many different opportunitie
s to practice a skill before it is mastered. Having the social skills t
o work cooperatively with friends is a crucial aspect of the kindergart
en curriculum.Montana is the perfect place to learn about agriculture a
nd nutrition. My students love to role play in our pretend kitchen in t
he early childhood classroom. I have had several kids ask me, \"Can we
try cooking with REAL food?\" I will take their idea and create \"Commo
n Core Cooking Lessons\" where we learn important math and writing conc
epts while cooking delicious healthy food for snack time. My students w
ill have a grounded appreciation for the work that went into making the
food and knowledge of where the ingredients came from as well as how it
is healthy for their bodies. This project would expand our learning of
nutrition and agricultural cooking recipes by having us peel our own ap
ples to make homemade applesauce, make our own bread, and mix up health
y plants from our classroom garden in the spring. We will also create o
ur own cookbooks to be printed and shared with families. \r\nStudents w
ill gain math and literature skills as well as a life long enjoyment fo
r healthy cooking.nannan
==================================================
```

In [13]:
```
# \r \n \t remove from string python: http://texthandler.com/info/remov
e-line-breaks-python/
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

```
 A person is a person, no matter how small.  (Dr.Seuss) I teach the sma
llest students with the biggest enthusiasm for learning. My students le
arn in many different ways using all of our senses and multiple intelli
gences. I use a wide range of techniques to help all my students succee
d.   Students in my class come from a variety of different backgrounds
which makes for wonderful sharing of experiences and cultures, includin
g Native Americans.  Our school is a caring community of successful lea
rners which can be seen through collaborative student project based lea
rning in and out of the classroom. Kindergarteners in my class love to
```

```
work with hands-on materials and have many different opportunities to p
ractice a skill before it is mastered. Having the social skills to work
cooperatively with friends is a crucial aspect of the kindergarten curr
iculum.Montana is the perfect place to learn about agriculture and nutr
ition. My students love to role play in our pretend kitchen in the earl
y childhood classroom. I have had several kids ask me,  Can we try cook
ing with REAL food?  I will take their idea and create  Common Core Coo
king Lessons  where we learn important math and writing concepts while
cooking delicious healthy food for snack time. My students will have a
grounded appreciation for the work that went into making the food and k
nowledge of where the ingredients came from as well as how it is health
y for their bodies. This project would expand our learning of nutrition
and agricultural cooking recipes by having us peel our own apples to ma
ke homemade applesauce, make our own bread, and mix up healthy plants f
rom our classroom garden in the spring. We will also create our own coo
kbooks to be printed and shared with families.   Students will gain mat
h and literature skills as well as a life long enjoyment for healthy co
oking.nannan
```

In [14]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

```
 A person is a person no matter how small Dr Seuss I teach the smallest
students with the biggest enthusiasm for learning My students learn in
many different ways using all of our senses and multiple intelligences
I use a wide range of techniques to help all my students succeed Studen
ts in my class come from a variety of different backgrounds which makes
for wonderful sharing of experiences and cultures including Native Amer
icans Our school is a caring community of successful learners which can
be seen through collaborative student project based learning in and out
of the classroom Kindergarteners in my class love to work with hands on
materials and have many different opportunities to practice a skill bef
ore it is mastered Having the social skills to work cooperatively with
friends is a crucial aspect of the kindergarten curriculum Montana is t
he perfect place to learn about agriculture and nutrition My students l
ove to role play in our pretend kitchen in the early childhood classroo
m I have had several kids ask me Can we try cooking with REAL food I wi
ll take their idea and create Common Core Cooking Lessons where we lear
```

```
n important math and writing concepts while cooking delicious healthy f
ood for snack time My students will have a grounded appreciation for th
e work that went into making the food and knowledge of where the ingred
ients came from as well as how it is healthy for their bodies This proj
ect would expand our learning of nutrition and agricultural cooking rec
ipes by having us peel our own apples to make homemade applesauce make
our own bread and mix up healthy plants from our classroom garden in th
e spring We will also create our own cookbooks to be printed and shared
with families Students will gain math and literature skills as well as
a life long enjoyment for healthy cooking nannan
```

In [15]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'no
t'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves'
, 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselve
s', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'it
s', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'th
is', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'h
ave', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between',
'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out',
'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'h
ow', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 's
o', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should',
"should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't",
'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "is
```

```
n't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',
 "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [16]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwor
ds)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████
██████| 109248/109248 [00:48<00:00, 2261.97it/s]
```

In [17]:
```
# after preprocesing
preprocessed_essays[20000]
```

Out[17]:
```
'person person no matter small dr seuss teach smallest students biggest
enthusiasm learning students learn many different ways using senses mul
tiple intelligences use wide range techniques help students succeed stu
dents class come variety different backgrounds makes wonderful sharing
experiences cultures including native americans school caring community
successful learners seen collaborative student project based learning c
lassroom kindergarteners class love work hands materials many different
opportunities practice skill mastered social skills work cooperatively
friends crucial aspect kindergarten curriculum montana perfect place le
arn agriculture nutrition students love role play pretend kitchen early
childhood classroom several kids ask try cooking real food take idea cr
eate common core cooking lessons learn important math writing concepts
cooking delicious healthy food snack time students grounded appreciatio
```

```
n work went making food knowledge ingredients came well healthy bodies
project would expand learning nutrition agricultural cooking recipes us
peel apples make homemade applesauce make bread mix healthy plants clas
sroom garden spring also create cookbooks printed shared families stude
nts gain math literature skills well life long enjoyment healthy cookin
g nannan'
```

1.4 Preprocessing of `project_title`

In [18]:
```
# similarly you can preprocess the titles also

project_data.columns
#sent1= decontracted(project_data['project_title'].values[20000])
preprocessed_title = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['project_title'].values):
    sent1 = decontracted(sentance)
    sent1 = sent1.replace('\\r', ' ')
    sent1 = sent1.replace('\\"', ' ')
    sent1 = sent1.replace('\\n', ' ')
    sent1 = re.sub('[^A-Za-z0-9]+', ' ', sent1)
    # https://gist.github.com/sebleier/554280
    sent1 = ' '.join(e for e in sent1.split() if e not in stopwords)
    preprocessed_title.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████
████| 109248/109248 [00:02<00:00, 47434.02it/s]
```

1.5 Preparing data for models

In [19]:
```
project_data.columns
```

Out[19]:
```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_stat
e',
       'Date', 'project_grade_category', 'project_title', 'project_essa
y_1',
```

```
       'project_essay_2', 'project_essay_3', 'project_essay_4',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_appr
oved',
       'clean_categories', 'clean_subcategories', 'essay'],
      dtype='object')
```

we are going to consider

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

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

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

In [20]:
```
Y=project_data['project_is_approved']
```

In [21]:
```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity'
:'sum'}).reset_index()
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

In [22]:
```
project_data['preprocessed_essays'] = preprocessed_essays
project_data['preprocessed_title'] = preprocessed_title

column_values=['clean_categories', 'clean_subcategories', 'school_stat
e', 'project_grade_category', 'teacher_prefix','preprocessed_essays','p
```

```
reprocessed_title' ,'price']

def select_columns(dataframe, column_names):
    new_frame = dataframe.loc[:, column_names]
    return new_frame

process_columns=select_columns(project_data,column_values)
```

In [23]:
```
process_columns.head()
```

Out[23]:

	clean_categories	clean_subcategories	school_state	project_grade_category	teacher
0	Math_Science	AppliedSciences Health_LifeScience	CA	Grades PreK-2	Mrs.
1	SpecialNeeds	SpecialNeeds	UT	Grades 3-5	Ms.
2	Literacy_Language	Literacy	CA	Grades PreK-2	Mrs.
3	AppliedLearning	EarlyDevelopment	GA	Grades PreK-2	Mrs.
4	Literacy_Language	Literacy	WA	Grades 3-5	Mrs.

In [24]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
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, shuffle=Flase)# this is for time series split
X_train, X_test, y_train, y_test = train_test_split(process_columns, Y,
 test_size=0.33) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_
size=0.33) # this is random splitting


print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)
```

```
(49041, 8) (49041,)
(24155, 8) (24155,)
(36052, 8) (36052,)
==========================================================================
============================
```

In [25]:

```
print("train columns",X_train.columns)

print("cV columns",X_cv.columns)

print("test columns",X_test.columns)
```

```
train columns Index(['clean_categories', 'clean_subcategories', 'school
_state',
       'project_grade_category', 'teacher_prefix', 'preprocessed_essay
s',
       'preprocessed_title', 'price'],
      dtype='object')
cV columns Index(['clean_categories', 'clean_subcategories', 'school_st
ate',
       'project_grade_category', 'teacher_prefix', 'preprocessed_essay
s',
       'preprocessed_title', 'price'],
      dtype='object')
test columns Index(['clean_categories', 'clean_subcategories', 'school_
state',
```

```
       'project_grade_category', 'teacher_prefix', 'preprocessed_essay
s',
       'preprocessed_title', 'price'],
      dtype='object')
```

1.5.1 Vectorizing Categorical data

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

In [26]:
```
# 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()), l
owercase=False, binary=True)
vectorizer.fit(X_train['clean_categories'].values)

categories_one_hot_train = vectorizer.transform(X_train['clean_categori
es'].values)
categories_one_hot_test = vectorizer.transform(X_test['clean_categorie
s'].values)
categories_one_hot_cv = vectorizer.transform(X_cv['clean_categories'].v
alues)
print(vectorizer.get_feature_names())

print("Shape of train matrix after one hot encodig ",categories_one_hot
_train.shape)
print("Shape of test matrix after one hot encodig ",categories_one_hot_
test.shape)
print("Shape of cv matrix after one hot encodig ",categories_one_hot_cv
.shape)
```

```
['Math_Science', 'History_Civics', 'AppliedLearning', 'Music_Arts', 'Wa
rmth', 'Health_Sports', 'SpecialNeeds', 'Care_Hunger', 'Literacy_Langua
ge']
Shape of train matrix after one hot encodig  (49041, 9)
Shape of test matrix after one hot encodig  (36052, 9)
Shape of cv matrix after one hot encodig  (24155, 9)
```

In [27]:
```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys
()), lowercase=False, binary=True)
vectorizer.fit(X_train['clean_subcategories'].values)
print(vectorizer.get_feature_names())

sub_categories_one_hot_train = vectorizer.transform(X_train['clean_subc
ategories'].values)
sub_categories_one_hot_test = vectorizer.transform(X_test['clean_subcat
egories'].values)
sub_categories_one_hot_cv = vectorizer.transform(X_cv['clean_subcategor
ies'].values)

print("Shape of train matrix after one hot encodig ",sub_categories_one
_hot_train.shape)
print("Shape of test matrix after one hot encodig ",sub_categories_one_
hot_test.shape)
print("Shape of cv matrix after one hot encodig ",sub_categories_one_ho
t_cv.shape)
```

```
['Civics_Government', 'NutritionEducation', 'FinancialLiteracy', 'Mathe
matics', 'Gym_Fitness', 'ESL', 'VisualArts', 'History_Geography', 'Othe
r', 'Health_LifeScience', 'Economics', 'ForeignLanguages', 'Literature_
Writing', 'TeamSports', 'College_CareerPrep', 'CharacterEducation', 'Ea
rlyDevelopment', 'AppliedSciences', 'Literacy', 'Music', 'SpecialNeed
s', 'Care_Hunger', 'EnvironmentalScience', 'PerformingArts', 'Health_We
llness', 'SocialSciences', 'Warmth', 'ParentInvolvement', 'CommunitySer
vice', 'Extracurricular']
Shape of train matrix after one hot encodig  (49041, 30)
Shape of test matrix after one hot encodig  (36052, 30)
Shape of cv matrix after one hot encodig  (24155, 30)
```

In [28]:
```
# we use count vectorizer to convert the values of categorical data :sc
hool_state
vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'])
print(vectorizer.get_feature_names())
```

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

print("Shape of train matrix after one hot encodig ",school_state_one_h
ot_train.shape)
print("Shape of test matrix after one hot encodig ",school_state_one_ho
t_test.shape)
print("Shape of cv matrix after one hot encodig ",school_state_one_hot_
cv.shape)
```

```
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'h
i', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi',
'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny',
'oh', 'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt',
'wa', 'wi', 'wv', 'wy']
Shape of train matrix after one hot encodig  (49041, 51)
Shape of test matrix after one hot encodig  (36052, 51)
Shape of cv matrix after one hot encodig  (24155, 51)
```

In [29]:
```
#we use count vectorizer to convert the values of categorical data :pro
ject_grade_category
vectorizer1 = CountVectorizer(stop_words=None)
k=X_train['project_grade_category']
l=X_test['project_grade_category']
m=X_test['project_grade_category']

k.replace(['Grades PreK-2', 'Grades 6-8', 'Grades 3-5','Grades 9-12'],
['A1', 'B2' ,'C3', 'D4'],inplace=True)
l.replace(['Grades PreK-2', 'Grades 6-8', 'Grades 3-5','Grades 9-12'],
['A1', 'B2' ,'C3', 'D4'],inplace=True)
m.replace(['Grades PreK-2', 'Grades 6-8', 'Grades 3-5','Grades 9-12'],
['A1', 'B2' ,'C3', 'D4'],inplace=True)

vectorizer1.fit(k)
```

```
project_grade_category_one_hot_train=vectorizer1.transform(X_train['pro
ject_grade_category'].values)
project_grade_category_one_hot_test=vectorizer1.transform(X_test['proje
ct_grade_category'].values)
project_grade_category_one_hot_cv=vectorizer1.transform(X_cv['project_g
rade_category'].values)

print("Shape of train matrix after one hot encodig ",project_grade_cate
gory_one_hot_train.shape)
print("Shape of test matrix after one hot encodig ",project_grade_categ
ory_one_hot_test.shape)
print("Shape of cv matrix after one hot encodig ",project_grade_categor
y_one_hot_cv.shape)
```

```
Shape of train matrix after one hot encodig  (49041, 4)
Shape of test matrix after one hot encodig  (36052, 4)
Shape of cv matrix after one hot encodig  (24155, 4)
```

In [30]:

```
#we use count vectorizer to convert the values of categorical data : te
acher_prefix
# getting error as we have null balues replacing them with 0

vectorizer1 = CountVectorizer()
project_data['teacher_prefix'].unique()

X_train['teacher_prefix'].fillna("", inplace = True)
X_test['teacher_prefix'].fillna("", inplace = True)
X_cv['teacher_prefix'].fillna("", inplace = True)

vectorizer1.fit(X_train['teacher_prefix'].values)
print(vectorizer1.get_feature_names())

teacher_prefix_one_hot_train = vectorizer1.transform(X_train['teacher_p
refix'].values)
teacher_prefix_one_hot_test = vectorizer1.transform(X_test['teacher_pre
fix'].values)
teacher_prefix_one_hot_cv = vectorizer1.transform(X_cv['teacher_prefix'
].values)
```

```
print("Shape of train matrix after one hot encodig ",teacher_prefix_one
_hot_train.shape)
print("Shape of test matrix after one hot encodig ",teacher_prefix_one_
hot_test.shape)
print("Shape of cv matrix after one hot encodig ",teacher_prefix_one_ho
t_cv.shape)
```

```
['dr', 'mr', 'mrs', 'ms', 'teacher']
Shape of train matrix after one hot encodig  (49041, 5)
Shape of test matrix after one hot encodig  (36052, 5)
Shape of cv matrix after one hot encodig  (24155, 5)
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [31]:
```
# We are considering only the words which appeared in at least 10 docum
ents(rows or projects).
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(X_train['preprocessed_essays'])

text_bow_train= vectorizer.transform(X_train['preprocessed_essays'])
text_bow_test= vectorizer.transform(X_test['preprocessed_essays'])
text_bow_cv= vectorizer.transform(X_cv['preprocessed_essays'])

print("Shape of train matrix after one hot encodig ",text_bow_train.sha
pe)
print("Shape of test matrix after one hot encodig ",text_bow_test.shape
)
print("Shape of cv matrix after one hot encodig ",text_bow_cv.shape)
```

```
Shape of train matrix after one hot encodig  (49041, 12022)
Shape of test matrix after one hot encodig  (36052, 12022)
Shape of cv matrix after one hot encodig  (24155, 12022)
```

In [32]:
```
# before you vectorize the title make sure you preprocess it
```

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

title_bow_train = vectorizer.transform(X_train['preprocessed_title'])
title_bow_test = vectorizer.transform(X_test['preprocessed_title'])
title_bow_cv= vectorizer.transform(X_cv['preprocessed_title'])

print("Shape of train matrix after one hot encodig title_bow",title_bow
_train.shape)
print("Shape of test matrix after one hot encodig title_bow",title_bow_
test.shape)
print("Shape of cv matrix after one hot encodig title_bow",title_bow_cv
.shape)
```

```
Shape of train matrix after one hot encodig title_bow (49041, 86)
Shape of test matrix after one hot encodig title_bow (36052, 86)
Shape of cv matrix after one hot encodig title_bow (24155, 86)
```

1.5.2.2 TFIDF vectorizer

In [33]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['preprocessed_essays'])

text_tfidf_train= vectorizer.transform(X_train['preprocessed_essays'])
text_tfidf_test= vectorizer.transform(X_test['preprocessed_essays'])
text_tfidf_cv = vectorizer.transform(X_cv['preprocessed_essays'])

print("Shape of train matrix after one hot encodig ",text_tfidf_train.s
hape)
print("Shape of test matrix after one hot encodig ",text_tfidf_test.sha
pe)
print("Shape of cv matrix after one hot encodig ",text_tfidf_cv.shape)
```

```
Shape of train matrix after one hot encodig  (49041, 12022)
Shape of test matrix after one hot encodig  (36052, 12022)
Shape of cv matrix after one hot encodig  (24155, 12022)
```

In [34]:
```
# Similarly you can vectorize for title also

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

title_tfidf_train = vectorizer.transform(X_train['preprocessed_title'])
title_tfidf_test = vectorizer.transform(X_test['preprocessed_title'])
title_tfidf_cv = vectorizer.transform(X_cv['preprocessed_title'])

print("Shape of train matrix after one hot encodig ",title_tfidf_train.
shape)
print("Shape of test matrix after one hot encodig ",title_tfidf_test.sh
ape)
print("Shape of cv matrix after one hot encodig ",title_tfidf_cv.shape)
```

```
Shape of train matrix after one hot encodig  (49041, 86)
Shape of test matrix after one hot encodig  (36052, 86)
Shape of cv matrix after one hot encodig  (24155, 86)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [35]:
```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
```

In [36]:
```
i=0
list_of_sentance_train=[]
for sentance in X_train['preprocessed_essays']:
    list_of_sentance_train.append(sentance.split())
```

In [37]:
```
# this line of code trains your w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_train,min_count=25,size=50, workers
=64)
```

In [38]:
```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 25 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 25 times  8545
sample words  ['studies', 'little', 'orally', 'cardboard', 'chicago',
'unlimited', 'behave', 'could', 'rescue', 'pbis', 'competitive', 'perce
ive', 'differentiated', 'animated', 'understood', '3doodlers', 'contrib
ute', 'relay', 'lockers', 'backbone', 'beg', 'essay', '49', 'tied', 'al
igns', 'entered', 'furthering', 'refer', 'tangle', 'audiobooks', 'diver
sified', 'pursue', 'research', 'clearer', 'adult', 'regardless', 'ozobo
ts', 'spacial', 'nurtured', 'type', 'complaining', 'advisory', 'anchor
s', 'puts', 'mill', 'bubbling', 'regarding', 'accommodations', 'define
d', 'toolbox']
```

In [39]:

```
# average Word2Vec of essays
# compute average word2vec for each review.
essay_vectors_train = []; # the avg-w2v for each sentence/review is sto
red in this list
for sent in tqdm(list_of_sentance_train): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, yo
u might need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    essay_vectors_train.append(sent_vec)
essay_vectors_train = np.array(essay_vectors_train)
print(essay_vectors_train.shape)
print(essay_vectors_train[0])
```

```
100%|
| 49041/49041 [07:48<00:00, 104.75it/s]
(49041, 50)
[ 0.63315258 -0.38574097 -0.49810698  0.38309076 -0.53476269 -0.4793623
6
  0.20469291 -0.47785118  0.34406394 -0.52757923 -1.05584799 -0.0610756
```

```
8
 -0.14338744  1.38287357  0.37065551  1.40284351 -0.89916537 -0.1887511
1
 -0.04067172 -0.68511452  0.7866739   0.06721374  1.74266589 -0.5997477
7
  0.94973975 -0.41035335  0.82651196 -0.30959466  0.96600433 -0.1325038
1
 -0.75312552 -0.36536954  0.26445248  0.44387232  0.58330333 -0.3860301
 -0.31205367  0.20435899 -0.24296447 -0.02822294 -0.65006374  0.8104277
9
  0.95757416 -1.00533667  0.55504456  0.63251838  0.62107267 -0.2028684
2
 -0.67301373 -0.78361219]
```

In [40]:
```
i=0
list_of_sentance_cv=[]
for sentance in X_cv['preprocessed_essays']:
    list_of_sentance_cv.append(sentance.split())
```

In [41]:
```
# average Word2Vec
# compute average word2vec for each review.
essay_vectors_cv = []; # the avg-w2v for each sentence/review is stored
 in this list
for sent in tqdm(list_of_sentance_cv): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, yo
u might need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    essay_vectors_cv.append(sent_vec)
essay_vectors_cv = np.array(essay_vectors_cv)
print(essay_vectors_cv.shape)
print(essay_vectors_cv[0])
```

```
100%|██████████| 24155/24155 [03:52<00:00, 103.86it/s]
```

```
(24155, 50)
[  9.53901520e-01  -3.38333754e-01  -1.90204265e-01   3.47366393e-01
  -5.70053321e-01  -7.81679301e-02   4.16064905e-01  -3.91037091e-01
   2.59542746e-01  -2.39204063e-01  -1.96139124e-01   3.96624206e-01
  -6.51593508e-01   1.91560539e-01   1.86224542e-01   5.03966172e-01
  -1.28824693e+00  -2.99887191e-01  -6.04845485e-01  -9.78743494e-02
   4.89699847e-02  -4.67218162e-02   1.24493763e+00   5.67011099e-02
   1.96432622e-01  -5.91053800e-01   1.39561967e-01  -6.09768880e-01
   5.05659403e-04   1.18879966e-01  -4.26634618e-02  -2.17173374e-01
   5.91026742e-01  -5.44897232e-02   1.55947240e-01  -1.56597419e-01
  -6.63146409e-01  -9.84003629e-02  -1.88159140e-01  -3.59666951e-03
  -4.85682459e-01   7.41200148e-01   1.21538559e+00  -1.00219605e+00
   3.55914476e-01   8.10548070e-01   1.42527608e-01   1.40021049e-01
  -3.65709583e-01  -7.09221705e-01]
```

In [42]:
```
i=0
list_of_sentance_test=[]
for sentance in X_test['preprocessed_essays']:
    list_of_sentance_test.append(sentance.split())
```

In [43]:
```
# average Word2Vec
# compute average word2vec for each review.
essay_vectors_test = []; # the avg-w2v for each sentence/review is stor
ed in this list
for sent in tqdm(list_of_sentance_test): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, yo
u might need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
```

```
        sent_vec /= cnt_words
    essay_vectors_test.append(sent_vec)
essay_vectors_test = np.array(essay_vectors_test)
print(essay_vectors_test.shape)
print(essay_vectors_test[0])
```

```
100%|██████████| 36052/36052 [05:56<00:00, 101.03it/s]
```

```
(36052, 50)
[ 0.48663328 -0.01331412 -0.4272521   0.33839254 -0.61721247  0.1874131
3
  0.25352137  0.10518354  0.63577932 -0.90403529 -0.0105616  -0.4575105
1
 -0.24054827  0.71376757  0.45419003  0.52572976 -1.42636291 -0.6324172
4
  0.02746982 -0.45699802 -0.06002245 -0.07732141  1.25869903 -0.2827734
  0.0117711  -0.51462311  0.31373233 -0.34042369 -0.05477299  0.0752009
2
 -0.6096528  -0.33518618  0.44138674 -0.44514305 -0.31779995  0.1965309
 -0.69268889  0.20678781 -0.18949649 -0.14782676 -0.41137686  1.1135659
7
  0.75567536 -0.85514091  0.35814783  0.74868519  0.43415771 -0.1074735
1
 -1.12504558 -1.05718799]
```

In [44]:
```
#similarly doing it for preprocessed title
i=0
list_of_sentance_train=[]
for sentance in X_train['preprocessed_title']:
    list_of_sentance_train.append(sentance.split())
```

In [45]:
```
# this line of code trains your w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=
16)
```

In [46]:
```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  86
sample words  ['resources', 'best', 'provide', 'subscription', 'corresp
ond', 'always', 'genuinely', 'also', 'rich', 'parents', 'not', 'games',
'world', 'importance', 'environment', 'magazines', 'text', 'afford', 'r
eal', 'graders', 'high', 'nonfiction', 'want', 'engaging', 'homes', 'wo
rksheets', 'curiosity', 'lifelong', 'interesting', 'around', 'topics',
'videos', 'economic', 'find', 'inspire', 'lead', 'used', 'children', 'e
nthusiasm', 'know', 'important', 'backgrounds', 'simply', 'interest',
'kids', 'allow', 'issues', 'absolutely', 'printable', 'past']
```

In [47]:

```
# compute average word2vec for each review.
title_vectors_train = []; # the avg-w2v for each sentence/review is sto
red in this list
for sent in tqdm(list_of_sentance_train): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, yo
u might need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    title_vectors_train.append(sent_vec)
title_vectors_train = np.array(title_vectors_train)
print(title_vectors_train.shape)
print(title_vectors_train[0])
```

```
100%|
| 49041/49041 [00:14<00:00, 3288.79it/s]
(49041, 50)
[-0.02142887  0.01730876  0.00072661  0.0555929  -0.12814029 -0.0025030
4
 -0.0376166  -0.02882473  0.19510403  0.16193078 -0.21292551 -0.1383688
8
  0.06834265  0.01037928 -0.11450379  0.0834282  -0.25598777 -0.1281918
```

```
8
  0.11296396  0.06143274  0.09660182  0.01210864  0.05169685  0.1332716
4
  0.18137633  0.17992615 -0.43159751  0.27692373  0.33809955 -0.1277390
8
 -0.01261954 -0.00577792 -0.24338869  0.08353683 -0.06980331 -0.1761835
6
  0.17251488 -0.19569046  0.29281613 -0.17539076  0.10599237  0.0991772
8
 -0.04549673 -0.09507812 -0.03024092  0.04500106 -0.23121983 -0.3244706
8
 -0.13421676 -0.14621522]
```

In [48]:
```
i=0
list_of_sentance_cv=[]
for sentance in X_cv['preprocessed_title']:
    list_of_sentance_cv.append(sentance.split())
```

In [49]:
```
# compute average word2vec for each review.
title_vectors_cv = []; # the avg-w2v for each sentence/review is stored
 in this list
for sent in tqdm(list_of_sentance_cv): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, yo
u might need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    title_vectors_cv.append(sent_vec)
title_vectors_cv = np.array(title_vectors_cv)
print(title_vectors_cv.shape)
print(title_vectors_cv[0])
```

```
100%|
```

```
████████| 24155/24155 [00:06<00:00, 3654.36it/s]
(24155, 50)
[-0.02142887  0.01730876  0.00072661  0.0555929  -0.12814029 -0.0025030
4
 -0.0376166  -0.02882473  0.19510403  0.16193078 -0.21292551 -0.1383688
8
  0.06834265  0.01037928 -0.11450379  0.0834282  -0.25598777 -0.1281918
8
  0.11296396  0.06143274  0.09660182  0.01210864  0.05169685  0.1332716
4
  0.18137633  0.17992615 -0.43159751  0.27692373  0.33809955 -0.1277390
8
 -0.01261954 -0.00577792 -0.24338869  0.08353683 -0.06980331 -0.1761835
6
  0.17251488 -0.19569046  0.29281613 -0.17539076  0.10599237  0.0991772
8
 -0.04549673 -0.09507812 -0.03024092  0.04500106 -0.23121983 -0.3244706
8
 -0.13421676 -0.14621522]
```

In [50]:
```
i=0
list_of_sentance_test=[]
for sentance in X_test['preprocessed_title']:
    list_of_sentance_test.append(sentance.split())
```

In [51]:
```
# compute average word2vec for each review.
title_vectors_test = []; # the avg-w2v for each sentence/review is stor
ed in this list
for sent in tqdm(list_of_sentance_test): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, yo
u might need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
```

```
    if cnt_words != 0:
        sent_vec /= cnt_words
    title_vectors_test.append(sent_vec)
title_vectors_test = np.array(title_vectors_test)
print(title_vectors_test.shape)
print(title_vectors_test[0])
```

```
100%|██████████| 36052/36052 [00:10<00:00, 3574.23it/s]
```

```
(36052, 50)
[-0.02142887  0.01730876  0.00072661  0.0555929  -0.12814029 -0.0025030
4
 -0.0376166  -0.02882473  0.19510403  0.16193078 -0.21292551 -0.1383688
8
  0.06834265  0.01037928 -0.11450379  0.0834282  -0.25598777 -0.1281918
8
  0.11296396  0.06143274  0.09660182  0.01210864  0.05169685  0.1332716
4
  0.18137633  0.17992615 -0.43159751  0.27692373  0.33809955 -0.1277390
8
 -0.01261954 -0.00577792 -0.24338869  0.08353683 -0.06980331 -0.1761835
6
  0.17251488 -0.19569046  0.29281613 -0.17539076  0.10599237  0.0991772
8
 -0.04549673 -0.09507812 -0.03024092  0.04500106 -0.23121983 -0.3244706
8
 -0.13421676 -0.14621522]
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [52]:

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

```
.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [53]:
```
# stronging variables into pickle files python: http://www.jessicayung.
com/how-to-use-pickle-to-save-and-load-variables-in-python/
# make sure you have the glove_vectors file
with open('C:/Users/pramod reddy chandi/Desktop/pram/applied ai course/
DonorsChoose_2018/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [54]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_train = []; # the avg-w2v for each sentence/review is
 stored in this list
for sentence in tqdm(X_train['preprocessed_essays']): # for each revie
w/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentenc
e/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and t
he tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentenc
e.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_train.append(vector)

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

```
100%|██████████| 49041/49041 [01:17<00:00, 636.58it/s]
```

```
49041
300
```

In [55]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_test = []; # 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 sentenc
e/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and t
he tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentenc
e.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_test.append(vector)

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

```
100%|██████████| 36052/36052 [00:55<00:00, 646.32it/s]
```

```
36052
300
```

In [56]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is st
ored in this list
for sentence in tqdm(X_cv['preprocessed_essays']): # for each review/se
```

```
ntence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentenc
e/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and t
he tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentenc
e.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_cv.append(vector)

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

```
100%|████████████████████████████████████████████████████████████
████████| 24155/24155 [00:36<00:00, 655.98it/s]
```

```
24155
300
```

In [57]:

```
# Similarly you can vectorize for title also
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['preprocessed_title'])
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model
.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [58]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_title_train = []; # the avg-w2v for each sentence/review is s
tored in this list
```

```
for sentence in tqdm(X_train['preprocessed_title']): # for each review/
sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentenc
e/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and t
he tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentenc
e.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_title_train.append(vector)

print(len(tfidf_w2v_title_train))
print(len(tfidf_w2v_title_train[0]))
```

```
100%|████████████████████████████████████████████████████████████████
████████| 49041/49041 [00:48<00:00, 1019.30it/s]
```

```
49041
300
```

In [59]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_title_test = []; # the avg-w2v for each sentence/review is st
ored in this list
for sentence in tqdm(X_test['preprocessed_title']): # for each review/s
entence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentenc
e/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and t
```

```
he tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentenc
e.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_title_test.append(vector)

print(len(tfidf_w2v_title_test))
print(len(tfidf_w2v_title_test[0]))
```

```
100%|██████████████████████████████████████████████████████████████
████████| 36052/36052 [00:34<00:00, 1057.47it/s]
```

```
36052
300
```

In [60]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_title_cv = []; # the avg-w2v for each sentence/review is stor
ed in this list
for sentence in tqdm(X_cv['preprocessed_title']): # for each review/sen
tence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentenc
e/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and t
he tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentenc
e.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_title_cv.append(vector)
```

```
print(len(tfidf_w2v_title_cv))
print(len(tfidf_w2v_title_cv[0]))
```

```
100%|██████████| 24155/24155 [00:22<00:00, 1091.12it/s]
```

```
24155
300
```

1.5.3 Vectorizing Numerical features

In [61]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import 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(X_train['price'].values.reshape(-1,1)) # finding the mean and standard deviation of this data
print("Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
price_standardized_train= price_scalar.transform(X_train['price'].values.reshape(-1, 1))
price_standardized_test= price_scalar.transform(X_test['price'].values.reshape(-1, 1))
price_standardized_cv= price_scalar.transform(X_cv['price'].values.reshape(-1, 1))
```

```
Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}
```

In [129]:
```
price_standardized_train
```

Out[129]:

	price_standard_train
0	0.680611
1	-0.803214
2	-0.317573
3	-0.712875
4	-0.757906
5	0.540758
6	-0.658794
7	0.345210
8	-0.551440
9	0.470498
10	-0.186075
11	0.653292
12	-0.796559
13	-0.800791
14	-0.810510
15	-0.627159
16	-0.481208
17	-0.520613
18	-0.231439
19	0.566796

	price_standard_train
20	0.136127
21	-0.806194
22	-0.573997
23	-0.297801
24	-0.796308
25	-0.674139
26	-0.390284
27	-0.391203
28	-0.738161
29	-0.142298
...	...
49011	-0.267530
49012	-0.016898
49013	0.404637
49014	-0.038452
49015	-0.448793
49016	-0.321889
49017	-0.754870
49018	-0.434646
49019	-0.534175
49020	0.604029
49021	-0.158923

	price_standard_train
49022	1.535851
49023	0.537277
49024	0.599072
49025	-0.490314
49026	-0.477170
49027	-0.771579
49028	-0.603711
49029	-0.657402
49030	-0.683217
49031	0.670920
49032	0.759616
49033	-0.214591
49034	0.116745
49035	-0.063878
49036	-0.782746
49037	-0.768906
49038	-0.535734
49039	2.649966
49040	0.294192

49041 rows × 1 columns

1.5.4 Merging all the above features

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

In [63]:
```
# 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_1 = hstack((school_state_one_hot_train,categories_one_hot_train,sub_categories_one_hot_train))
X_cat_train=hstack((X_1,teacher_prefix_one_hot_train,project_grade_category_one_hot_train))


X_2 = hstack((school_state_one_hot_test,categories_one_hot_test,sub_categories_one_hot_test))
X_cat_test=hstack((X_2,teacher_prefix_one_hot_test,project_grade_category_one_hot_test))

X_3 = hstack((school_state_one_hot_cv,categories_one_hot_cv,sub_categories_one_hot_cv))
X_cat_cv=hstack((X_3,teacher_prefix_one_hot_cv,project_grade_category_one_hot_cv))

#dealing with numerical values
#considering the value of price standardized values

price_standardized_train = pd.DataFrame({'price_standard_train':price_standardized_train[:,0]})
price_standardized_test = pd.DataFrame({'price_standard_test':price_standardized_test[:,0]})
price_standardized_cv = pd.DataFrame({'price_standard_cv':price_standardized_cv[:,0]})


#combining numerical ,project_title(BOW)  and preprocessed_essay (BOW)
num_text_train=hstack((price_standardized_train,text_bow_train,title_bow_train))
```

```
num_text_test=hstack((price_standardized_test,text_bow_test,title_bow_t
est))
num_text_cv=hstack((price_standardized_cv,text_bow_cv,title_bow_cv))

#froming features for set1

set1_train=hstack((X_cat_train,num_text_train))
set1_test=hstack((X_cat_test,num_text_test))
set1_cv=hstack((X_cat_cv,num_text_cv))

#numerical + project_title(TFIDF)+ preprocessed_essay (TFIDF)
num_tfidf_train=hstack((price_standardized_train,text_tfidf_train,title
_tfidf_train))
num_tfidf_test=hstack((price_standardized_test,text_tfidf_test,title_tf
idf_test))
num_tfidf_cv=hstack((price_standardized_cv,text_tfidf_cv,title_tfidf_cv
))

#froming features for set2

set2_train=hstack((X_cat_train,num_tfidf_train))
set2_test=hstack((X_cat_test,num_tfidf_test))
set2_cv=hstack((X_cat_cv,num_tfidf_cv))

#numerical + project_title(AVG W2V)+ preprocessed_essay (AVG W2V)
num_w2v_train=hstack((price_standardized_train,essay_vectors_train,titl
e_vectors_train))
num_w2v_test=hstack((price_standardized_test,essay_vectors_test,title_v
ectors_test))
num_w2v_cv=hstack((price_standardized_cv,essay_vectors_cv,title_vectors
_cv))

#forming features for set3

set3_train=hstack((X_cat_train,num_w2v_train))
set3_test=hstack((X_cat_test,num_w2v_test))
set3_cv=hstack((X_cat_cv,num_w2v_cv))

#numerical+project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)
```

```
num_tfidfw2v_train=hstack((price_standardized_train,tfidf_w2v_vectors_t
rain,tfidf_w2v_title_train))
num_tfidfw2v_test=hstack((price_standardized_test,tfidf_w2v_vectors_tes
t,tfidf_w2v_title_test))
num_tfidfw2v_cv=hstack((price_standardized_cv,tfidf_w2v_vectors_cv,tfid
f_w2v_title_cv))

#forming features for set4


set4_train=hstack((X_cat_train,num_tfidfw2v_train))
set4_test=hstack((X_cat_test,num_tfidfw2v_test))
set4_cv=hstack((X_cat_cv,num_tfidfw2v_cv))

#y values are
#y_train
#y_test
#y_cv
```

Assignment 3: Apply KNN

1. **[Task-1] Apply KNN(brute force version) on these feature sets**
    - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)
    - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF)
    - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_essay (AVG W2V)
    - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)

2. **Hyper paramter tuning to find best K**
    - Find the best hyper parameter which results in the maximum AUC value

- Find the best hyper paramter using k-fold cross validation (or) simple cross validation data
- Use gridsearch-cv or randomsearch-cv or write your own for loops to do this task

3. **Representation of results**

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, as shown in the figure
  - Once you find the best hyper parameter, you need to train your model-M using the best hyper-param. Now, find the AUC on test data and plot the ROC curve on both train and test using model-M.
  - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points

4. **[Task-2]**

- Select top 2000 features from feature Set 2 using `SelectKBest` and then apply KNN on top of these features
-

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

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

5. **Conclusion**

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

Note: Data Leakage

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

2. K Nearest Neighbor

Task 1.1 Applying KNN brute force on BOW, SET 1

In [64]:
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probab
ility estimates of the positive class
    # not the predicted outputs

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

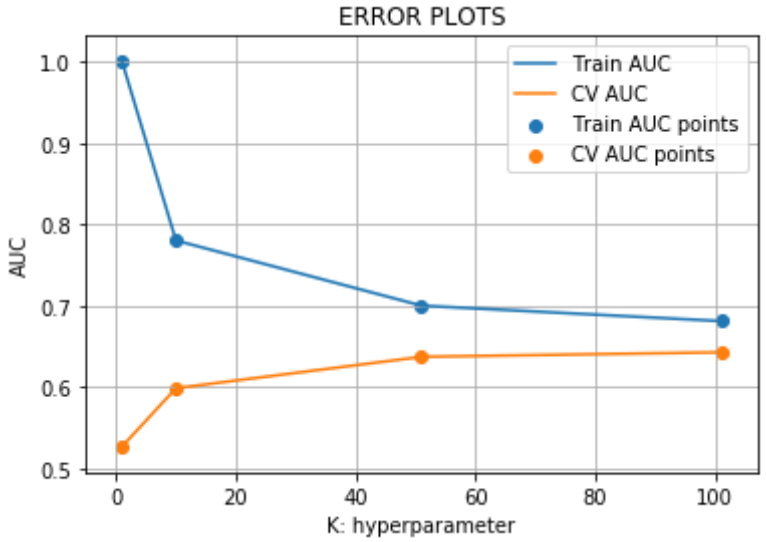
```
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [72]:
```
#performing the KNN on set1 data
set1_train
```

Out[72]:
```
<49041x12208 sparse matrix of type '<class 'numpy.float64'>'
        with 9472756 stored elements in COOrdinate format>
```

In [77]:
```
%%time
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 clas
s, confidence values, 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 wit
h greater label.

"""
X_tr=set1_train.tocsr()
X_cr=set1_cv.tocsr()

train_auc = []
cv_auc = []
K = [1, 10,51, 101]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i,n_jobs=-1)
    neigh.fit(X_tr, y_train)

    y_train_pred = batch_predict(neigh, X_tr)
```

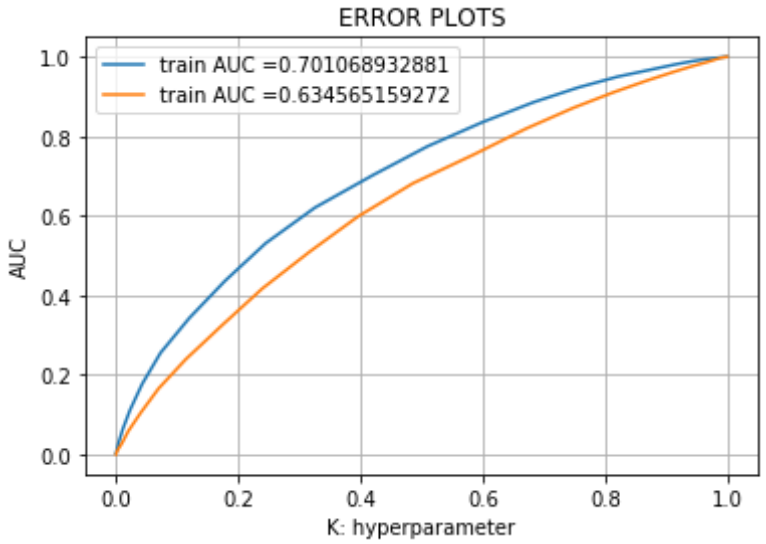
```
    y_cv_pred = batch_predict(neigh, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter +should be proba
bility estimates of the positive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

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

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




```
Wall time: 46min 46s
```

In [78]:
```
#here we are choosing the best_k based on forloop results
best_k = 50

#from the graph we can take the best value of 50 as maximum AUC on cv d
ata and gap between the train and cv is less
```

In [80]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc
_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

X_te=set1_test.tocsr()

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

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

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

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



In [81]:
```
# 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(fpr*(1-tpr))]

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

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

In [84]:
```
%%time
print("="*100)
```

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, tr
ain_fpr, train_tpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test
_fpr, test_tpr)))
```

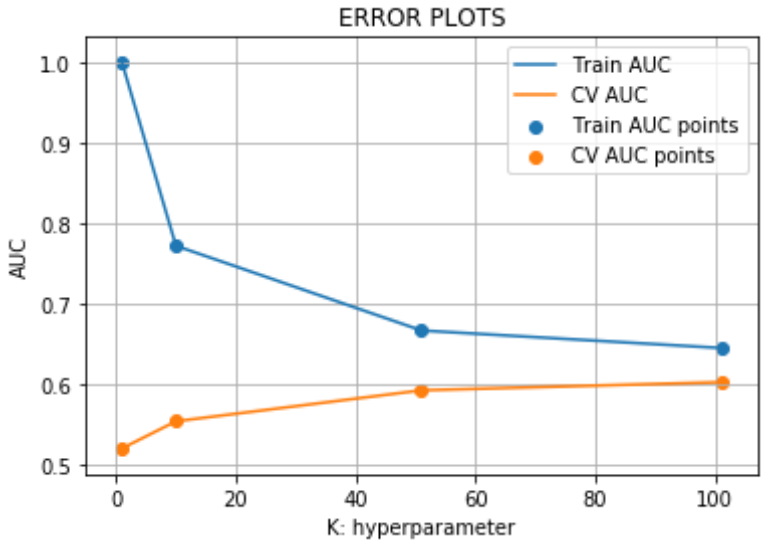
```
====================================================================
============================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.418672375154 for threshold 0.78
[[ 4321  3130]
 [12388 29202]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.361407051794 for threshold 0.78
[[ 2797  2641]
 [ 9746 20868]]
Wall time: 89.8 ms
```

2.4.2 Applying KNN brute force on TFIDF, SET 2

In [92]:
```
#Preparing data to perform TFIDf on data#
X_tr=set2_train.tocsr()
X_cr=set2_cv.tocsr()
X_te=set2_test.tocsr()
```

In [87]:
```
%%time
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 clas
```

```
s, confidence values, 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 wit
h greater label.

"""

train_auc = []
cv_auc = []
K = [1, 10,51, 101]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i,n_jobs=-1)
    neigh.fit(X_tr, y_train)

    y_train_pred = batch_predict(neigh, X_tr)
    y_cv_pred = batch_predict(neigh, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter +should be proba
bility estimates of the positive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

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

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

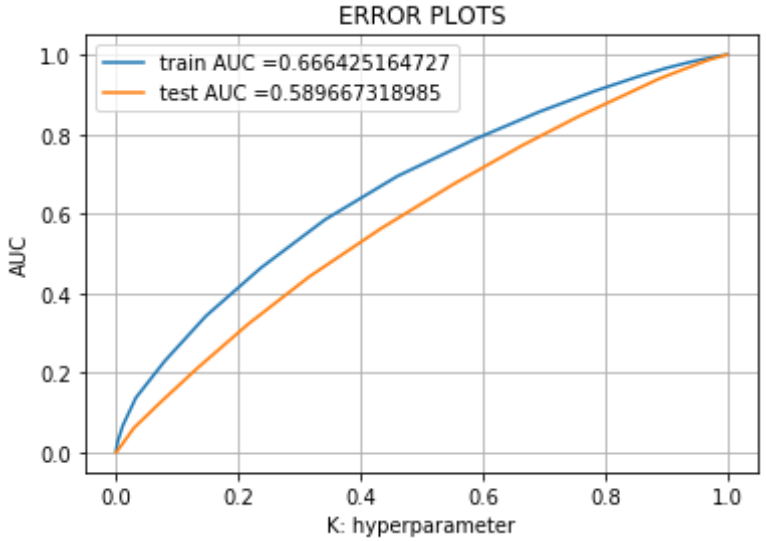



```
Wall time: 39min 20s
```

In [88]:
```
#here we are choosing the best_k based on forloop results
best_k = 50

#from the graph we can take the best value of 50 as maximum AUC on cv d
ata and gap between the train and cv is less
```

In [94]:
```
%%time
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc
_curve.html#sklearn.metrics.roc_curve


from sklearn.metrics import roc_curve, auc

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

y_train_pred = batch_predict(neigh, X_tr)
```

```
y_test_pred = batch_predict(neigh, X_te)

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

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




```
Wall time: 41min 34s
```

In [95]:
```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, tr
```

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

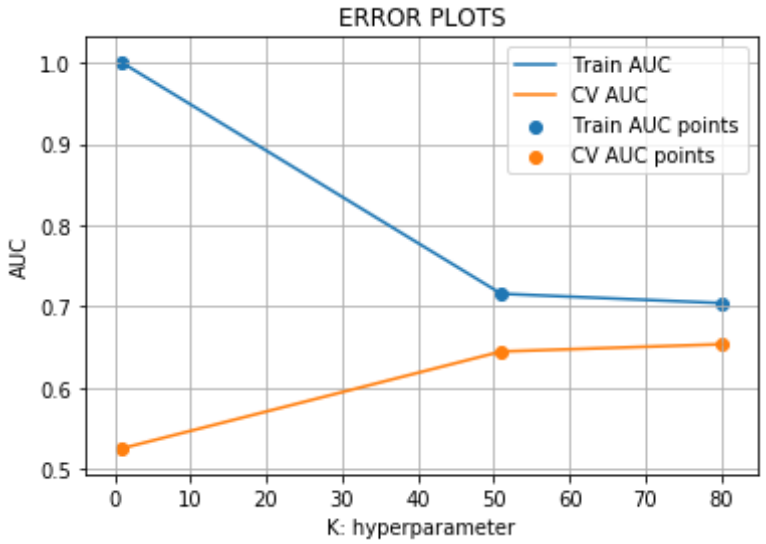
```
====================================================================
============================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.385183390505 for threshold 0.86
[[ 4904  2547]
 [17250 24340]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.318946988439 for threshold 0.84
[[ 2439  2999]
 [ 9980 20634]]
```

2.4.3 Applying KNN brute force on AVG W2V, SET 3

In [96]:
```
# Ple#Preparing data to perform TFIDf on data#
X_tr=set3_train.tocsr()
X_cr=set3_cv.tocsr()
X_te=set3_test.tocsr()
```

In [97]:
```
%%time
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 clas
s, confidence values, 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 wit
h greater label.
```

```
"""

train_auc = []
cv_auc = []
K = [1, 51, 80]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i,n_jobs=-1)
    neigh.fit(X_tr, y_train)

    y_train_pred = batch_predict(neigh, X_tr)
    y_cv_pred = batch_predict(neigh, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter +should be proba
bility estimates of the positive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

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

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

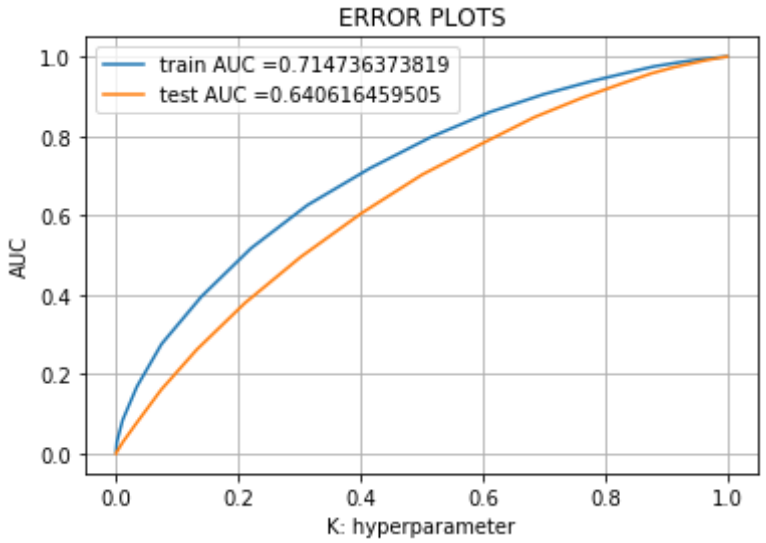



```
Wall time: 43min 7s
```

In [98]:
```
#here we are choosing the best_k based on forloop results
best_k = 54

#from the graph we can take the best value of 50 as maximum AUC on cv d
ata and gap between the train and cv is less
```

In [99]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc
_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

neigh = KNeighborsClassifier(n_neighbors=best_k,n_jobs=-1)
neigh.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs

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

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
```

```
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

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



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

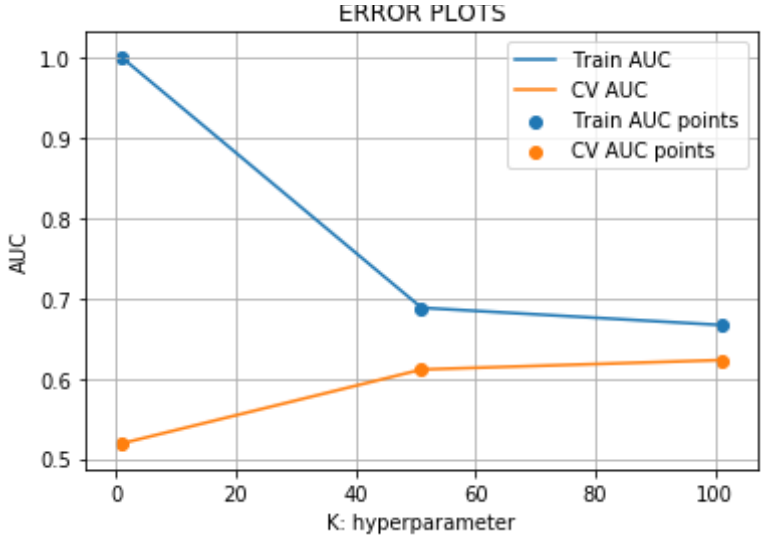
```
================================================================================
==============================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.429888369987 for threshold 0.87
[[ 5119  2332]
 [15566 26024]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.362312511712 for threshold 0.852
[[ 2714  2724]
 [ 9100 21514]]
Wall time: 88.8 ms
```

2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

In [101]:
```
# Ple#Preparing data to perform TFIDf on data#
X_tr=set4_train.tocsr()
X_cr=set4_cv.tocsr()
X_te=set4_test.tocsr()
```

In [102]:
```
%%time
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 clas
s, confidence values, 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 wit
h greater label.

"""

train_auc = []
```

```
cv_auc = []
K = [1,51, 101]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i,n_jobs=-1)
    neigh.fit(X_tr, y_train)

    y_train_pred = batch_predict(neigh, X_tr)
    y_cv_pred = batch_predict(neigh, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter +should be proba
bility estimates of the positive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

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

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

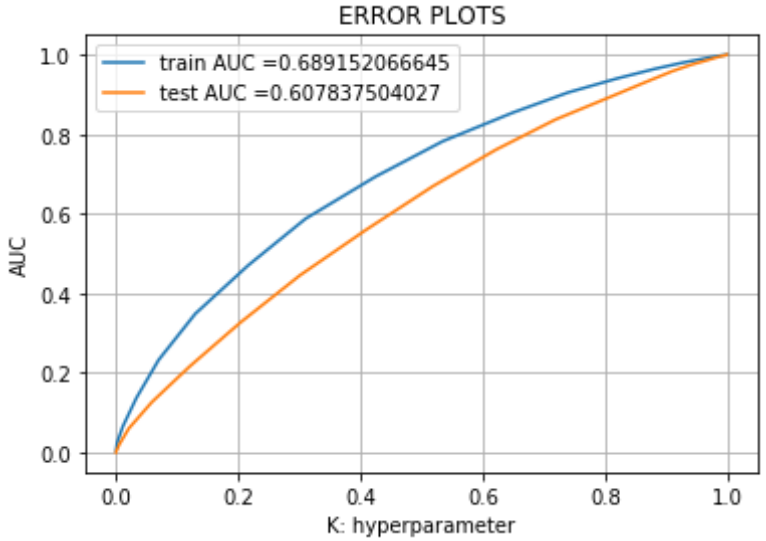



```
Wall time: 1h 31min 59s
```

In [103]:
```
#here we are choosing the best_k based on forloop results
best_k = 50

#from the graph we can take the best value of 50 as maximum AUC on cv d
ata and gap between the train and cv is less
```

In [104]:
```
%%time

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

neigh = KNeighborsClassifier(n_neighbors=best_k,n_jobs=-1)
neigh.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs

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

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

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




```
Wall time: 44min 33s
```

In [105]:
```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, tr
ain_fpr, train_tpr)))
```

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

```
====================================================================
============================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.405461550528 for threshold 0.84
[[ 4288  3163]
 [12749 28841]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.330803035185 for threshold 0.84
[[ 2618  2820]
 [10119 20495]]
```

2.5 Feature selection with `SelectKBest`

In [157]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_selection import SelectKBest, chi2
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['preprocessed_essays'].values) # fit has to happ
en only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer.transform(X_train['preprocessed_essay
s'].values)
X_cv_essay_tfidf = vectorizer.transform(X_cv['preprocessed_essays'].val
ues)
X_test_essay_tfidf = vectorizer.transform(X_test['preprocessed_essays']
.values)


#Selecting top 2000 best features from the generated tfidf features
selector = SelectKBest(chi2, k = 2000 )
selector.fit(X_train_essay_tfidf,y_train)
X_train_essay_2000 = selector.transform(X_train_essay_tfidf)
X_cv_essay_2000 = selector.transform(X_cv_essay_tfidf)
```

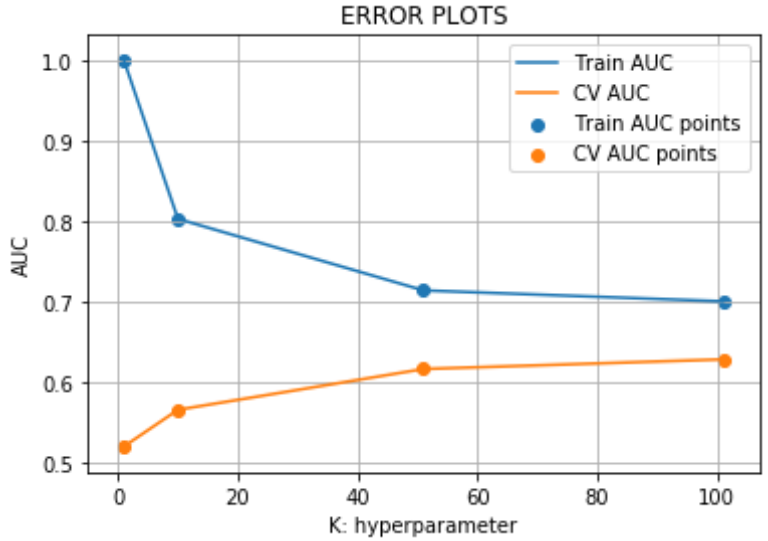
```
X_test_essay_2000 = selector.transform(X_test_essay_tfidf)
print(X_train_essay_2000.shape)
print(X_cv_essay_2000.shape)
print(X_test_essay_2000.shape)
```

```
(49041, 2000)
(24155, 2000)
(36052, 2000)
```

In [159]:

```
%%time
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 clas
s, confidence values, 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 wit
h greater label.

"""

train_auc = []
cv_auc = []
K = [1, 10,51, 101]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i,n_jobs=-1)
    neigh.fit(X_train_essay_2000, y_train)

    y_train_pred = batch_predict(neigh, X_train_essay_2000)
    y_cv_pred = batch_predict(neigh, X_cv_essay_2000)

    # roc_auc_score(y_true, y_score) the 2nd parameter +should be proba
bility estimates of the positive class
    # not the predicted outputs
```

```
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

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

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



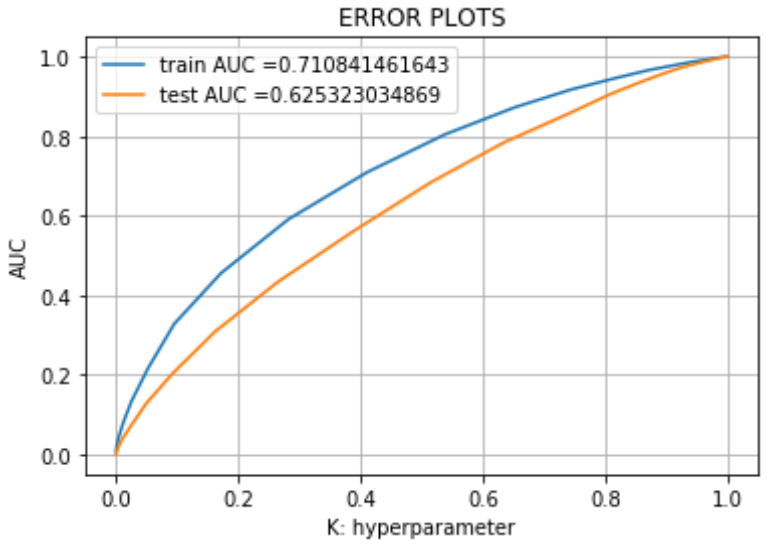

```
Wall time: 30min 48s
```

In [160]:
```
#here we are choosing the best_k based on forloop results
best_k = 60
```

```
#from the graph we can take the best value of 50 as maximum AUC on cv d
ata and gap between the train and cv is less
```

In [161]:

```
%%time

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

neigh = KNeighborsClassifier(n_neighbors=best_k,n_jobs=-1)
neigh.fit(X_train_essay_2000, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_train_essay_2000)
y_test_pred = batch_predict(neigh, X_test_essay_2000)

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

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



Wall time: 6min 50s

In [162]:
```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, tr
ain_fpr, train_tpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test
_fpr, test_tpr)))
```

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.424596081108 for threshold 0.85
[[ 4391  3060]
 [12083 29507]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.343343769168 for threshold 0.85
[[ 2636  2802]
 [ 9660 20954]]
```

3. Conclusions

In [165]:
```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable

x12=PrettyTable()

x12.field_names = ["Vectorizer", "model", "hyperparameter", "Train AUC"
,"Test AUC"]

x12.add_row(["BOW", "brute", 50, 0.7010, 0.6345])

x12.add_row(["TFIDF", "brute", 50, 0.6664,0.5896])

x12.add_row(["W2V", "brute", 54, 0.7147, 0.6406])

x12.add_row(["TFIDFW2V", "brute", 50, 0.6891, 0.6078])

print(x12)
```

```
+------------+-------+----------------+-----------+----------+
| Vectorizer | model | hyperparameter | Train AUC | Test AUC |
+------------+-------+----------------+-----------+----------+
|    BOW     | brute |       50       |   0.701   |  0.6345  |
|   TFIDF    | brute |       50       |   0.6664  |  0.5896  |
|    W2V     | brute |       54       |   0.7147  |  0.6406  |
|  TFIDFW2V  | brute |       50       |   0.6891  |  0.6078  |
+------------+-------+----------------+-----------+----------+
```

Conclusion :Thus we have compared the different models and computed the hyperparameters using brute force method.We find best hyperparameter with value equals 50.