DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located ([Two-letter U.S. postal code (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Postal_codes)](https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Postal_codes)). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy` • `Literature & Writing, Social Sciences`

Feature	Description
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • `My students need hands on literacy materials to manage sensory needs!`
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*
`project_essay_4`	Fourth application essay*
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
`teacher_id`	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same teacher. **Example:** `2`

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

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

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

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

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

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

Notes on the Essay Data

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

- __project_essay_1:__ "Introduce us to your classroom"
- __project_essay_2:__ "Tell us more about your students"
- __project_essay_3:__ "Describe how your students will use the materials you're requesting"
- __project_essay_4:__ "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
```

```
C:\Users\Rahul\Anaconda3\lib\site-packages\gensim\utils.py:1209: UserWarning: detected Windows; aliasing chunkize to chunkiz
e_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [2]:

```
from tqdm import tqdm
for i in tqdm(range(int(10e6))):
    pass
```

```
100%|████████████████████████████████████████████████████████████████| 10000000/10000000 [00:02<00:00, 3702750.54it/s]
```

1.1 Reading Data

Considered 5000 datapoints due to memory issues

In [3]:

```
project_data = pd.read_csv('train_data.csv',nrows=5000)
resource_data = pd.read_csv('resources.csv')
```

In [4]:

```
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 (5000, 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 [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 Data Preprocessing

1.2.1. Text preprocessing: project_subject_categories

a) Removing special characters from project_subject_categories

In [6]:

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

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & 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())
```

b) Replacing the original project_subject_categories with the cleaned project_subject_categories in the dataframe

In [7]:
```
project_data['clean_categories'] = cat_list
project_data.drop(['project_subject_categories'], axis=1, inplace=True)
project_data.head(2)
```

Out[7]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	project_subject
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades PreK-2	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grades 6-8	Civics & Gc

1.2.2 Text preprocessing: project_subject_subcategories

a) Removing special characters from project_subject_categories

In [8]:

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

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

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

b) Replacing the original project_subject_subcategories with the cleaned project_subject_subcategories in the dataframe

In [9]:

```
project_data['clean_subcategories'] = sub_cat_list
project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)
project_data.head(2)
```

Out[9]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	project_title	p
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades PreK-2	Educational Support for English Learners at Home	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grades 6-8	Wanted: Projector for Hungry Learners	

1.2.3 Merging all the Project Essay's into one

In [10]:

```
# 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 [11]:

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

Out[11]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	project_title	p
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades PreK-2	Educational Support for English Learners at Home	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grades 6-8	Wanted: Projector for Hungry Learners	

Dropping project_essay1,2,3 and 4

In [12]:

```
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
project_data.drop(['project_essay_4'], axis=1, inplace=True)
```

In [13]:

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

Out[13]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	project_title	p
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades PreK-2	Educational Support for English Learners at Home	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grades 6-8	Wanted: Projector for Hungry Learners	N

1.2.4 Adding the price column from resource_data into the project_data dataframe

In [14]:

```
# we get the cost of the project using resource.csv file
resource_data.head(2)
```

Out[14]:

	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

a) Resetting the indices for all the groups

In [15]:
```
# https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexes-for-all-groups-in-one-step
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
price_data.head(2)
```

Out[15]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21

b) Joining the price_data dataframe with the project_data dataframe

In [16]:
```
# join two dataframes in python:
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

In [17]:
```
project_data.head(2)
```

Out[17]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	project_title	p
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades PreK-2	Educational Support for English Learners at Home	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grades 6-8	Wanted: Projector for Hungry Learners	N

1.3 Text preprocessing

1.3.1 Essay Text

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

Out[18]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	project_title	p
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades PreK-2	Educational Support for English Learners at Home	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grades 6-8	Wanted: Projector for Hungry Learners	M

In [19]:

```
# printing some random essays.
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[2000])
print("="*50)
print(project_data['essay'].values[999])
print("="*50)
```

My students are English learners that are working on English as their second or third languages. We are a melting pot of refugees, immigrants, and native-born Americans bringing the gift of language to our school. \r\n\r\n We have over 24 languages represented in our English Learner program with students at every level of mastery. We also have over 40 countries represented with the families within our school. Each student brings a wealth of knowledge and experiences to us that open our eyes to new cultures, beliefs, and respect.\"The limits of your language are the limits of your world.\"-Ludwig Wittgenstein Our English learner's have a strong support system at home that begs for more resources. Many times our parents are learning to read and speak English along side of their children. Sometimes this creates barriers for parents to be able to help their child learn phonetics, letter recognition, and other reading skills.\r\n\r\nBy providing these dvd's and players, students are able to continue their mastery of the English language even if no one at home is able to assist. All families with students within the Level 1 proficiency status, will be a offered to be a part of this program. These educational videos will be specially chosen by the English Learner Teacher and will be sent home regularly to watch. The videos are to help the child develop early reading skills.\r\n\r\nParents that do not have access to a dvd player will have the opportunity to check out a dvd player to use for the year. The plan is to use these videos and educational dvd's for the years to come for other EL students.\r\nnannan

==

The 51 fifth grade students that will cycle through my classroom this year all love learning, at least most of the time. At our school, 97.3% of the students receive free or reduced price lunch. Of the 560 students, 97.3% are minority students. \r\nThe school has a vibrant community that loves to get together and celebrate. Around Halloween there is a whole school parade to show off the beautiful costumes that students wear. On Cinco de Mayo we put on a big festival with crafts made by the students, dances, and games. At the end of the year the school hosts a carnival to celebrate the hard work put in during the school year, with a dunk tank being the most popular activity.My students will use these five brightly colored Hokki stools in place of regular, stationary, 4-legged chairs. As I will only have a total of ten in the classroom and not enough for each student to have an individual one, they will be used in a variety of ways. During independent reading time they will be used as special chairs students will each use on occasion. I will utilize them in place of chairs at my small group tables during math and reading times. The rest of the day they will be used by the students who need the highest amount of movement in their life in order to stay focused on school.\r\n\r\nWhenever asked what the classroom is missing, my students always say more Hokki Stools. They can't get their fill of the 5 stools we already have. When the students are sitting in group with me on the Hokki Stools, they are always moving, but at the same time doing their work. Anytime the students get to pick where they can sit, the Hokki Stools are the first to be taken. There are always students who head over to the kidney table to get one of the stools who are disappointed as there are not enough of them. \r\n\r\nWe ask a lot of students to sit for 7 hours a day. The Hokki stools will be a compromise that allow my students to do desk work and move at the same time. These stools will help students to meet their 60 minutes a day of movement by allowing them to activate their core muscles for balance while they sit. For many of my students, these chairs will take away the barrier that exists in schools for a child who can't sit still.nannan

==

How do you remember your days of school? Was it in a sterile environment with plain walls, rows of desks, and a teacher in front of the room? A typical day in our room is nothing like that. I work hard to create a warm inviting themed room for my students look forward to coming to each day.\r\n\r\nMy class is made up of 28 wonderfully unique boys and girls of mixed races in Arkansas.\r\nThey attend a Title I school, which means there is a high enough percentage of free and reduced-price lunch to qualify. Our school is an \"open classroom\" concept, which is very unique as there are no walls separating the classrooms. These 9 and 10 year-old students are very eager learners; they are like sponges, absorbing all the information and experiences and keep on wanting more.With these resources such as the comfy red throw pillows and the whimsical nautical hanging decor and the blue fish nets, I will be able to help create the mood in our classroom setting to be one of a themed nautical environment. Creating a classroom environment is very important in the success in each and every child's education. The naut ical photo props will be used with each child as they step foot into our classroom for the first time on Meet the Teacher evening. I'll take pictures of each child with them, have them developed, and then hung in our classroom ready for their first day of 4th grade. This kind gesture will set the tone before even the first day of school! The nautical thank you cards will be used throughout the year by the students as they create thank you cards to their team groups.\r\n\r\nYour generous donations will help me to help make our classroom a fun, inviting, learning environment from day one.\r\n\r\nIt costs lost of money out of my own pocket on resources to get our classroom ready. Please consider helping with this project to make our new school year a very successful one. Thank you!nannan

==

Describing my students isn't an easy task. Many would say that they are inspirational, creative, and hard-working. They are all unique - unique in their interests, their learning, their abilities, and so much more. What they all have in common is their desire to learn each day, despite difficulties that they encounter. \r\nOur classroom is amazing - because we understand that everyone learns at their own pace. As the teacher, I pride myself in making sure my students are always engaged, motivated, and inspired to create their own learning! \r\nThis project is to help my students choose seating that is more appropriate for them, developmentally. Many students tire of sitting in chairs during lessons, and having different seats available helps to keep them engaged and learning.\r\nFlexible seating is important in our classroom, as many of our students struggle with attention, focus, and engagement. We currently have stability balls for seating, as well as regular chairs, but these stools will help students who have trouble with balance, or find it difficult to sit on a stability ball for a long period of time. We are excited to try these stools as a part of our engaging classroom community!nannan

==

Welcome to our spectacular 1st and 2nd grade ELL classroom. I have the most amazing class of motivated second language learners. These youngsters come from homes with hardworking families that support education. The students along with their parents want to succeed and place value on doing well school. However, life challenges seem to make this difficult for many of the families at my school.\r\nEach day, my students come to class eager to start the day and learn. My classroom brings much stability and ongoing support which they don't always get at home. Our typical day includes hands-on experiences, cooperative learning, and plenty of opportunities for success. I want each student to feel like the classroom is a safe, happy place. It is my hope that each student develops a lifelong love for learning. Our Title 1 school community works hard toward our goals of student success and growth.Student engagement is the key to success in learning. My first and second graders often struggle with the ability to focus and pay attention. They need an opportunity for extra movement which will allow their brains to be more alert and attentive. I would like to provide them with a few \"tools\" to help relieve stress, reduce anxiety, and relax.\r\n\r\n   Having \"tools\" like balance balls, wobble cushions, and squishy-fidget balls will provide much needed sensory input for my students.\r\n\r\nThese items will enable them to channel their physical energy in a positive way, allowing them to focus on their work and reach their full potential as learners.nannan

==

In [20]:

```
# 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 [21]:

```
sent = decontracted(project_data['essay'].values[2000])
print(sent)
print("="*50)
```

```
Describing my students is not an easy task.  Many would say that they are inspirational, creative, and hard-working.  They a
re all unique - unique in their interests, their learning, their abilities, and so much more.  What they all have in common
is their desire to learn each day, despite difficulties that they encounter.  \r\nOur classroom is amazing - because we unde
rstand that everyone learns at their own pace.  As the teacher, I pride myself in making sure my students are always engage
d, motivated, and inspired to create their own learning! \r\nThis project is to help my students choose seating that is more
appropriate for them, developmentally.  Many students tire of sitting in chairs during lessons, and having different seats a
vailable helps to keep them engaged and learning.\r\nFlexible seating is important in our classroom, as many of our students
struggle with attention, focus, and engagement.  We currently have stability balls for seating, as well as regular chairs, b
ut these stools will help students who have trouble with balance, or find it difficult to sit on a stability ball for a long
period of time.  We are excited to try these stools as a part of our engaging classroom community!nannan
==================================================
```

In [22]:

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

```
Describing my students is not an easy task.  Many would say that they are inspirational, creative, and hard-working.  They a
re all unique - unique in their interests, their learning, their abilities, and so much more.  What they all have in common
is their desire to learn each day, despite difficulties that they encounter.    Our classroom is amazing - because we unders
tand that everyone learns at their own pace.  As the teacher, I pride myself in making sure my students are always engaged,
motivated, and inspired to create their own learning!   This project is to help my students choose seating that is more appr
opriate for them, developmentally.  Many students tire of sitting in chairs during lessons, and having different seats avail
able helps to keep them engaged and learning.  Flexible seating is important in our classroom, as many of our students strug
gle with attention, focus, and engagement.  We currently have stability balls for seating, as well as regular chairs, but th
ese stools will help students who have trouble with balance, or find it difficult to sit on a stability ball for a long peri
od of time.  We are excited to try these stools as a part of our engaging classroom community!nannan
```

In [23]:

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

```
Describing my students is not an easy task Many would say that they are inspirational creative and hard working They are all
unique unique in their interests their learning their abilities and so much more What they all have in common is their desir
e to learn each day despite difficulties that they encounter Our classroom is amazing because we understand that everyone le
arns at their own pace As the teacher I pride myself in making sure my students are always engaged motivated and inspired to
create their own learning This project is to help my students choose seating that is more appropriate for them developmental
ly Many students tire of sitting in chairs during lessons and having different seats available helps to keep them engaged an
d learning Flexible seating is important in our classroom as many of our students struggle with attention focus and engageme
nt We currently have stability balls for seating as well as regular chairs but these stools will help students who have trou
ble with balance or find it difficult to sit on a stability ball for a long period of time We are excited to try these stool
s as a part of our engaging classroom community nannan
```

In [24]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', '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", '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', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [25]:

```
# Combining all the above statemennts
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 not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████████████| 5000/5000 [00:03<00:00, 1654.61it/s]
```

```
In [26]: # after preprocesing
         preprocessed_essays[2000]
```

```
Out[26]: 'describing students not easy task many would say inspirational creative hard working they unique unique interests learning
         abilities much what common desire learn day despite difficulties encounter our classroom amazing understand everyone learns
         pace as teacher i pride making sure students always engaged motivated inspired create learning this project help students ch
         oose seating appropriate developmentally many students tire sitting chairs lessons different seats available helps keep enga
         ged learning flexible seating important classroom many students struggle attention focus engagement we currently stability b
         alls seating well regular chairs stools help students trouble balance find difficult sit stability ball long period time we
         excited try stools part engaging classroom community nannan'
```

1.3.2 Project title Text

```
In [27]: project_data.head(2)
```

```
Out[27]:
```

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	project_title	p
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades PreK-2	Educational Support for English Learners at Home	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grades 6-8	Wanted: Projector for Hungry Learners	

In [28]:

```
# Printing some random titles

print(project_data['project_title'].values[0])
print('='*50)
print(project_data['project_title'].values[1500])
print('='*50)
print(project_data['project_title'].values[2500])
print('='*50)
print(project_data['project_title'].values[4500])
print('='*50)
```

```
Educational Support for English Learners at Home
==================================================
Listening Center
==================================================
Food for the Brain!
==================================================
Ohana Means Family...We Support Each Other!
==================================================
```

In [29]:

```
# Testing out the decontracting

sent = decontracted(project_data['project_title'].values[4500])
sent
```

Out[29]:

```
'Ohana Means Family...We Support Each Other!'
```

In [30]:

```
# Testing out the removal of line breaks

sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

```
Ohana Means Family...We Support Each Other!
```

In [31]:

```
# Combining all the above statemennts
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 not in stopwords)
    preprocessed_titles.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████| 5000/5000 [00:00<00:00, 25252.07it/s]
```

In [32]:

```
print('Before preprocessing  :-------------------------',project_data['project_title'].values[4500])
# After Preprocessing
print('After preprocessing -------------------------',preprocessed_titles[4500])
```

```
Before preprocessing  :------------------------- Ohana Means Family...We Support Each Other!
After preprocessing ------------------------- ohana means family we support each other
```

1. 4 Preparing data for models

In [33]:

```
project_data.columns
```

Out[33]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', 'project_title',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'price',
       'quantity'],
      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
- price : numerical
```

In [34]:
```
project_data.head(2)
```

Out[34]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	project_title	p
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades PreK-2	Educational Support for English Learners at Home	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grades 6-8	Wanted: Projector for Hungry Learners	N

In [35]:
```
project_data.shape
```

Out[35]:
```
(5000, 16)
```

Selecting the features

1.4.1 Merging the preprocessed essays and preprocessed titles into the dataframe

In [36]:

```
project_data['preprocessed_essays'] = preprocessed_essays
```

In [37]:

```
project_data['preprocessed_titles'] = preprocessed_titles
```

In [38]:

```
project_data.head(1)
```

Out[38]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	project_title	prc
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades PreK-2	Educational Support for English Learners at Home	

1.4.2 Slicing the dataframe to contain only the necessary features

In [39]:

```
df_pre_slicing = project_data[['teacher_prefix', 'school_state','project_grade_category','teacher_number_of_previously_posted_projects','clean_categories', 'clean_subcategories', 'preprocessed_essays','price','preprocessed_titles','project_is_approved']]
```

In [40]:
```
df_pre_slicing.head(2)
```

Out[40]:

	teacher_prefix	school_state	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcategories	preprocessed
0	Mrs.	IN	Grades PreK-2	0	Literacy_Language	ESL Literacy	my studen learner e
1	Mr.	FL	Grades 6-8	7	History_Civics Health_Sports	Civics_Government TeamSports	our stude school eager l

In [41]:
```
df_pre_slicing.shape
```

Out[41]:
```
(5000, 10)
```

1.4.3 Selecting the X and y features

In [42]:
```
X = df_pre_slicing.iloc[:, :-1]
```

In [43]:
```
X.head(2)
```

Out[43]:

	teacher_prefix	school_state	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcategories	preprocessed
0	Mrs.	IN	Grades PreK-2	0	Literacy_Language	ESL Literacy	my studen learner e
1	Mr.	FL	Grades 6-8	7	History_Civics Health_Sports	Civics_Government TeamSports	our stude school eager l

In [44]:
```
y = df_pre_slicing['project_is_approved']
```

In [45]:

```
print(X.shape)
print(y.shape)
```

```
(5000, 9)
(5000,)
```

1.4.4 Splitting the data

In [46]:

```
# 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, stratify=y_train)
```

1.5 Vectorizing the categorical data

1.5.1 Converting text to vectors

a) BOW on preprocessed_essays

In [47]:

```python
# BOW on essay


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

print("="*100)



from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['preprocessed_essays'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['preprocessed_essays'].values)
X_cv_essay_bow = vectorizer.transform(X_cv['preprocessed_essays'].values)
X_test_essay_bow = vectorizer.transform(X_test['preprocessed_essays'].values)

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
print("="*100)
```

```
(2244, 9) (2244,)
(1106, 9) (1106,)
(1650, 9) (1650,)
====================================================================================================
After vectorizations
(2244, 5000) (2244,)
(1106, 5000) (1106,)
(1650, 5000) (1650,)
====================================================================================================
```

b) BOW on preprocessed_titles

```
In [48]: # BOW on preprocessed titles

         from sklearn.feature_extraction.text import CountVectorizer
         vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
         vectorizer.fit(X_train['preprocessed_titles'].values) # fit has to happen only on train data

         # we use the fitted CountVectorizer to convert the text to vector
         X_train_title_bow = vectorizer.transform(X_train['preprocessed_titles'].values)
         X_cv_title_bow = vectorizer.transform(X_cv['preprocessed_titles'].values)
         X_test_title_bow = vectorizer.transform(X_test['preprocessed_titles'].values)

         print("After vectorizations")
         print(X_train_title_bow.shape, y_train.shape)
         print(X_cv_title_bow.shape, y_cv.shape)
         print(X_test_title_bow.shape, y_test.shape)
         print("="*100)
```

```
After vectorizations
(2244, 195) (2244,)
(1106, 195) (1106,)
(1650, 195) (1650,)
====================================================================================================
```

1.5.2 One hot encoding the categorical features

a) one hot encoding the catogorical features: state

In [49]:
```
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
(2244, 51) (2244,)
(1106, 51) (1106,)
(1650, 51) (1650,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'm
d', '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']
====================================================================================================
```

b) one hot encoding the catogorical features: teacher_prefix

In [50]:

```
vectorizer = CountVectorizer()
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
(2244, 4) (2244,)
(1106, 4) (1106,)
(1650, 4) (1650,)
['mr', 'mrs', 'ms', 'teacher']
====================================================================================================
```

c) one hot encoding the catogorical features: project_grade_category

In [51]:

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].values)
X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].values)

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
(2244, 3) (2244,)
(1106, 3) (1106,)
(1650, 3) (1650,)
['12', 'grades', 'prek']
====================================================================================================
```

d) one hot encoding the catogorical features: clean_categories

In [52]:

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

# we use the fitted CountVectorizer to convert the text to vector
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
(2244, 9) (2244,)
(1106, 9) (1106,)
(1650, 9) (1650,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language', 'math_science', 'music_arts', 'sp
ecialneeds', 'warmth']
====================================================================================================
```

e) one hot encoding the catogorical features: clean_subcategories

In [53]:

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

# we use the fitted CountVectorizer to convert the text to vector
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
(2244, 30) (2244,)
(1106, 30) (1106,)
(1650, 30) (1650,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep', 'communityservice', 'ear
lydevelopment', 'economics', 'environmentalscience', 'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym
_fitness', 'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'mathematics', 'm
usic', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socialsciences', 'specialneeds', 'teamsports',
'visualarts', 'warmth']
====================================================================================================
```

1.5.3 Normalizing the numerical features: Price

In [54]:

```
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
(2244, 1) (2244,)
(1106, 1) (1106,)
(1650, 1) (1650,)
====================================================================================================
```

1.5.4 Standardizing the numerical features: Teacher previously posted projects

In [55]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

X_train_teacher_no_of_previously_submitted_projects_standardized = scaler.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_cv_teacher_no_of_previously_submitted_projects_standardized = scaler.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_test_teacher_no_of_previously_submitted_projects_standardized = scaler.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
```

```
C:\Users\Rahul\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.

C:\Users\Rahul\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.

C:\Users\Rahul\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.

C:\Users\Rahul\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

1.5.4 Concatinating all the features(BOW encoded)- Set 1

In [56]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_bow = hstack((X_train_essay_bow, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_price_norm,X_train_teacher_no_of_previously_submitted_projects_standardized)).tocsr()
X_cr_bow = hstack((X_cv_essay_bow, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_price_norm,X_cv_teacher_no_of_previously_submitted_projects_standardized)).tocsr()
X_te_bow = hstack((X_test_essay_bow, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_price_norm,X_test_teacher_no_of_previously_submitted_projects_standardized)).tocsr()

print("Final Data matrix")
print(X_tr_bow.shape, y_train.shape)
print(X_cr_bow.shape, y_cv.shape)
print(X_te_bow.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(2244, 5060) (2244,)
(1106, 5060) (1106,)
(1650, 5060) (1650,)
====================================================================================================
```

1.6 Applying KNN for BOW

In [58]:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_tr_bow,y_train)
pred_y = knn.predict(X_te_bow) # predicting y
```

1.6.1 Evaluting the perfomance of the model for Bow

a) Accuracy score

In [59]:

```
from sklearn.metrics import accuracy_score
```

In [60]:
```
accuracy_score(y_test,pred_y)
```

Out[60]:
```
0.8224242424242424
```

b) Confusion matrix and classification report for bow without hypertuning

In [61]:
```
from sklearn.metrics import confusion_matrix,classification_report
```

In [62]:
```
print(confusion_matrix(y_test,pred_y))
print(classification_report(y_test,pred_y))
```

```
[[  13  239]
 [  54 1344]]
              precision    recall  f1-score   support

           0       0.19      0.05      0.08       252
           1       0.85      0.96      0.90      1398

   micro avg       0.82      0.82      0.82      1650
   macro avg       0.52      0.51      0.49      1650
weighted avg       0.75      0.82      0.78      1650
```

1.6.2 Applying k_fold Cross_validation for bow

In [63]:
```
from sklearn.model_selection import cross_val_score
```

In [65]:

```
#create a new KNN model

knn_cv = KNeighborsClassifier(n_neighbors=5)

#train model with cv of 5

cv_scores = cross_val_score(knn_cv, X_tr_bow, y_train, cv=5)

#print each cv score (accuracy) and average them

print(cv_scores)
print('cv_scores mean:{}'.format(np.mean(cv_scores)))
```

```
[0.80222222 0.82888889 0.81696429 0.79017857 0.828125  ]
cv_scores mean:0.8132757936507936
```

The accuracy score after K-fold CV has not increased much

1.6.3 Hypertuning model parameters for bow

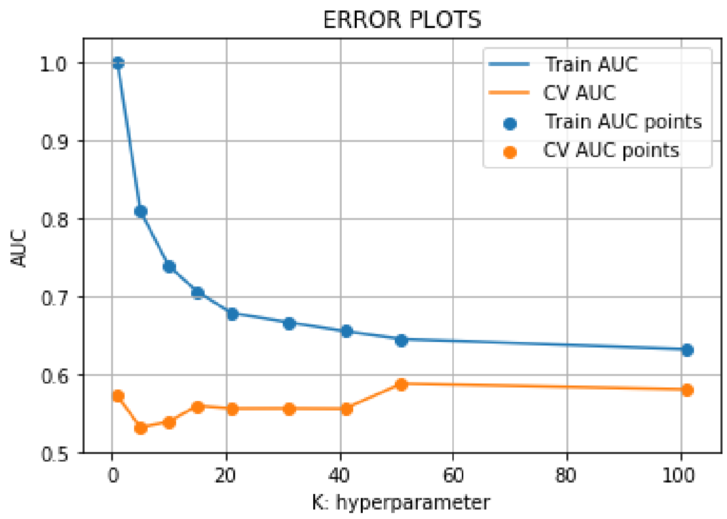
a) Simple loop

In [66]:

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

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

    return y_data_pred
```

In [68]:

```
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 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 with greater label.

"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51, 101]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_tr_bow, y_train)

    y_train_pred = batch_predict(neigh, X_tr_bow)
    y_cv_pred = batch_predict(neigh, X_cr_bow)

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

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

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

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



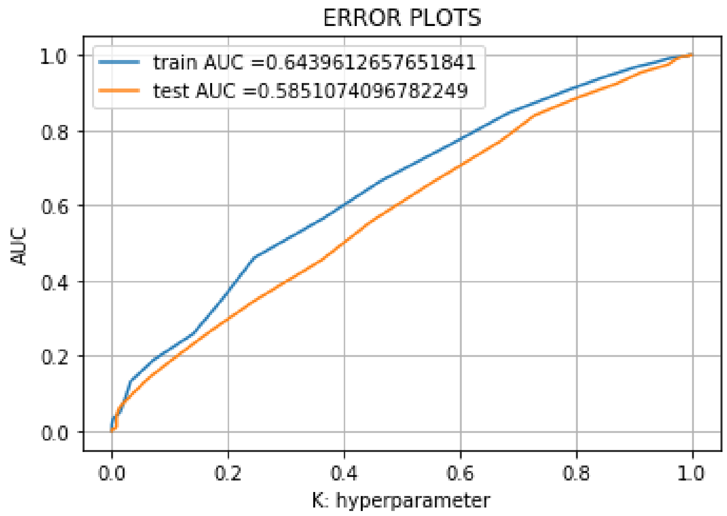
b) Fitting the model to the new hyper parameter

Lets us consider k = 99

```
In [69]: best_k = 99
```

In [143]:

```
# 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_bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr_bow)
y_test_pred = batch_predict(neigh, X_te_bow)

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

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



c) Confusion matrix for train data

In [72]:

```
# 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 threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [73]:

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

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24978625901986937 for threshold 0.788
[[ 166  176]
 [ 577 1325]]
```

d) Confusion matrix for test data

In [75]:

```
#This is Future Unseen Data
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24809460821365587 for threshold 0.778
[[  81  171]
 [ 304 1094]]
```

In [ ]:

```
```

Converting text to vectors (tfidf encoding)

1.7 TFIDF on text data

a) tfidf on preprocessed_essay

In [76]:

```
# TFIDF on essay


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

print("="*100)

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['preprocessed_essays'].values)

X_train_essay_tfidf = vectorizer.transform(X_train['preprocessed_essays'].values)
X_cv_essay_tfidf = vectorizer.transform(X_cv['preprocessed_essays'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['preprocessed_essays'].values)

print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
```

```
(2244, 9) (2244,)
(1106, 9) (1106,)
(1650, 9) (1650,)
====================================================================================================
After vectorizations
(2244, 5000) (2244,)
(1106, 5000) (1106,)
(1650, 5000) (1650,)
====================================================================================================
```

b) tfidf on preprocessed_titles

In [77]:

```
# TFIDF on title

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['preprocessed_titles'].values)

X_train_title_tfidf = vectorizer.transform(X_train['preprocessed_titles'].values)
X_cv_title_tfidf = vectorizer.transform(X_cv['preprocessed_titles'].values)
X_test_title_tfidf = vectorizer.transform(X_test['preprocessed_titles'].values)

print("After vectorizations")
print(X_train_title_tfidf.shape, y_train.shape)
print(X_cv_title_tfidf.shape, y_cv.shape)
print(X_test_title_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(2244, 195) (2244,)
(1106, 195) (1106,)
(1650, 195) (1650,)
====================================================================================================
```

1.7.1 Concatenating all the features

In [78]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_tfidf = hstack((X_train_essay_tfidf, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_price_norm,X_train_teacher_no_of_previously_submitted_projects_standardized)).tocsr()
X_cr_tfidf = hstack((X_cv_essay_tfidf, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_price_norm,X_cv_teacher_no_of_previously_submitted_projects_standardized)).tocsr()
X_te_tfidf = hstack((X_test_essay_tfidf, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_price_norm,X_test_teacher_no_of_previously_submitted_projects_standardized)).tocsr()

print("Final Data matrix")
print(X_tr_tfidf.shape, y_train.shape)
print(X_cr_tfidf.shape, y_cv.shape)
print(X_te_tfidf.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(2244, 5060) (2244,)
(1106, 5060) (1106,)
(1650, 5060) (1650,)
====================================================================================================
```

1.7.2 Applying KNN for tfidf

In [79]:

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_tr_tfidf,y_train)
pred_y = knn.predict(X_te_tfidf)
```

1.7.3 Confusion matrix for non hypertuned KNN on tfidf encoded data

In [80]:

```
print(confusion_matrix(y_test,pred_y))
print(classification_report(y_test,pred_y))
```

```
[[   2  250]
 [  30 1368]]
              precision    recall  f1-score   support

           0       0.06      0.01      0.01       252
           1       0.85      0.98      0.91      1398

   micro avg       0.83      0.83      0.83      1650
   macro avg       0.45      0.49      0.46      1650
weighted avg       0.73      0.83      0.77      1650
```

Applying k_fold cv for tfidf encoded data

create a new KNN model

knn_cv = KNeighborsClassifier(n_neighbors=5)

train model with cv of 5

cv_scores = cross_val_score(knn_cv, X_tr_tfidf, y_train, cv=5)

print each cv score (accuracy) and average them

print(cv_scores) print('cv_scores mean:{}'.format(np.mean(cv_scores)))

In [ ]:

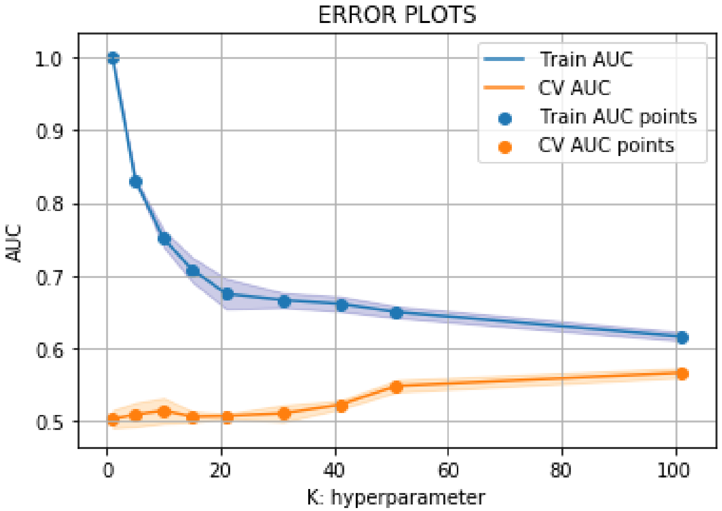
```

```

1.7.4 Hyper parameter tuning for tfidf

a) Grid search cv for Tfidf

In [81]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV

neigh = KNeighborsClassifier()
parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51,101]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(X_tr_tfidf, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

plt.plot(parameters['n_neighbors'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')


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

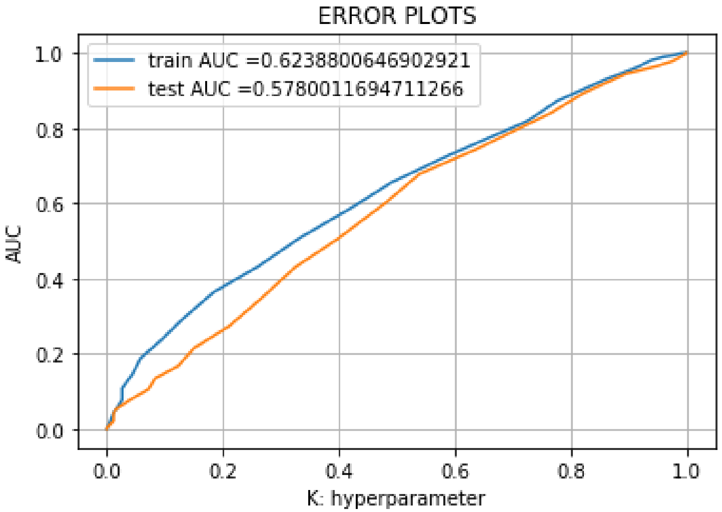


In [82]:

```
best_k_2 = 102
```

In [142]:

```
# 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_tfidf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr_tfidf)
y_test_pred = batch_predict(neigh, X_te_tfidf)

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

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



1.7.5 Confusion matrix for gridsearch for tfidf

a) Train data

In [84]:

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

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24622960911049555 for threshold 0.828
[[ 150  192]
 [ 555 1347]]
```

b) Test data

In [85]:

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

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24872448979591838 for threshold 0.838
[[117 135]
 [536 862]]
```

In [ ]:

```

```

1.8 Converting text to vectors (Avg W2V encoding)

a) Using Pretrained Models: Avg W2V

In [86]:

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

```
Loading Glove Model

1917495it [04:39, 6870.64it/s]

Done. 1917495  words loaded!
```

In [87]:

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

for i in preprocessed_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))
```

```
all the words in the coupus 782085
the unique words in the coupus 17889
The number of words that are present in both glove vectors and our coupus 17406 ( 97.3 %)
word 2 vec length 17406
```

In [88]:

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

In [89]:

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

1.8.1 AVG_w2v For train (essay and title)

a) For train essay

```
In [90]: # average Word2Vec
         # compute average word2vec for each review.
         avg_w2v_vectors_tr_essay = []; # 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
             avg_w2v_vectors_tr_essay.append(vector)

         print(len(avg_w2v_vectors_tr_essay))
         print(len(avg_w2v_vectors_tr_essay[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 2244/2244 [00:01<00:00, 2197.09it/s]

2244
300
```

b) For train title

In [91]:

```
avg_w2v_vectors_tr_title = []; # 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
    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_tr_title.append(vector)

print(len(avg_w2v_vectors_tr_title))
print(len(avg_w2v_vectors_tr_title[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 2244/2244 [00:00<00:00, 48635.89it/s]

2244
300
```

1.8.2 AVG_w2v For test essay and title

a) For test essay

```
In [92]: avg_w2v_vectors_te_essay = []; # 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
             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_te_essay.append(vector)

         print(len(avg_w2v_vectors_te_essay))
         print(len(avg_w2v_vectors_te_essay[0]))
```

```
100%|███████████████████████████████████████████████████████████████████████████| 1650/1650 [00:00<00:00, 2894.04it/s]

1650
300
```

b) For test title

```
In [93]: avg_w2v_vectors_te_title = []; # 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
             avg_w2v_vectors_te_title.append(vector)

         print(len(avg_w2v_vectors_te_title))
         print(len(avg_w2v_vectors_te_title[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 1650/1650 [00:00<00:00, 58756.22it/s]

1650
300
```

1.8.3 AVG_w2v For CV essay and title

a) For train cv essay

In [94]:

```
avg_w2v_vectors_cv_tr_essay = []; # the avg-w2v for each sentence/review is stored in this list
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
    avg_w2v_vectors_cv_tr_essay.append(vector)

print(len(avg_w2v_vectors_cv_tr_essay))
print(len(avg_w2v_vectors_cv_tr_essay[0]))
```

```
100%|██████████████████████████████████████████████████| 1106/1106 [00:00<00:00, 3008.98it/s]

1106
300
```

b) For test cv title

In [95]:

```
avg_w2v_vectors_cv_te_title = []; # 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
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_cv_te_title.append(vector)

print(len(avg_w2v_vectors_cv_te_title))
print(len(avg_w2v_vectors_cv_te_title[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 1106/1106 [00:00<00:00, 61258.21it/s]

1106
300
```

1.8.4 Concatenating all the features of AVG_w2v(set-3)

In [96]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_avg_w2v = hstack((avg_w2v_vectors_tr_essay,avg_w2v_vectors_tr_title, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_price_norm, X_train_teacher_no_of_previously_submitted_projects_standardized)).tocsr()
X_cr_avg_w2v = hstack((avg_w2v_vectors_cv_tr_essay,avg_w2v_vectors_cv_te_title, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_price_norm, X_cv_teacher_no_of_previously_submitted_projects_standardized)).tocsr()
X_te_avg_w2v = hstack((avg_w2v_vectors_te_essay,avg_w2v_vectors_te_title, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_price_norm, X_test_teacher_no_of_previously_submitted_projects_standardized)).tocsr()

print("Final Data matrix")
print(X_tr_avg_w2v.shape, y_train.shape)
print(X_cr_avg_w2v.shape, y_cv.shape)
print(X_te_avg_w2v.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(2244, 660) (2244,)
(1106, 660) (1106,)
(1650, 660) (1650,)
====================================================================================================
```

1.8.5 Applying KNN on avg_w2v

In [98]:

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_tr_avg_w2v,y_train)
pred_y = knn.predict(X_te_avg_w2v)
```

1.8.6 Confusion matrix for simple KNN

```
In [99]: print(confusion_matrix(y_test,pred_y))
         print(classification_report(y_test,pred_y))
```

```
[[  12  240]
 [  37 1361]]
              precision    recall  f1-score   support

           0       0.24      0.05      0.08       252
           1       0.85      0.97      0.91      1398

   micro avg       0.83      0.83      0.83      1650
   macro avg       0.55      0.51      0.49      1650
weighted avg       0.76      0.83      0.78      1650
```

Applying k_fold cv for avg_w2v encoded data

create a new KNN model

knn_cv = KNeighborsClassifier(n_neighbors=5)

train model with cv of 5

cv_scores = cross_val_score(knn_cv, X_tr, y_train, cv=5)

print each cv score (accuracy) and average them

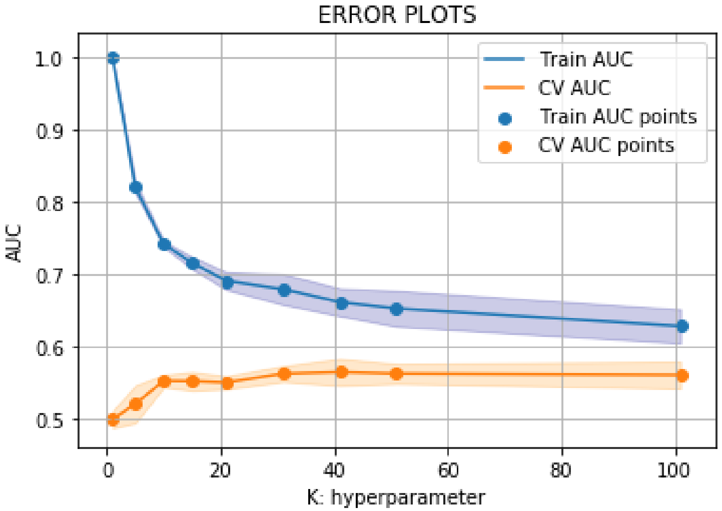
print(cv_scores) print('cv_scores mean:{}'.format(np.mean(cv_scores)))

1.8.7 Hypertuning parameter K

Gridsearch on avg_w2v data

In [100]:

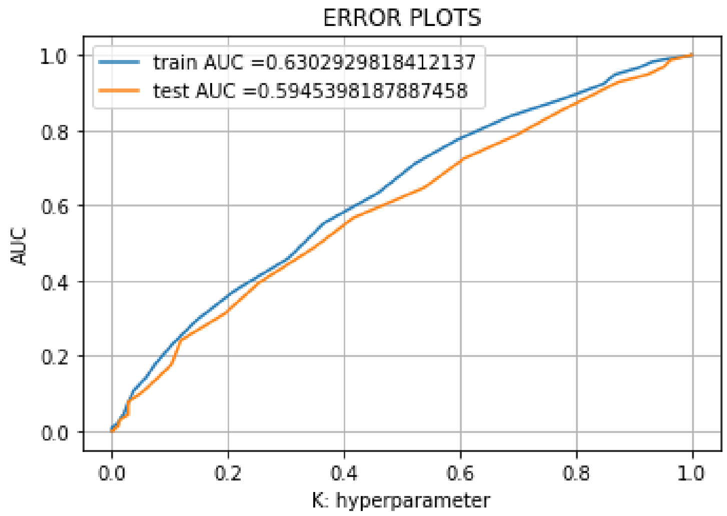
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV

neigh = KNeighborsClassifier()
parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51,101]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(X_tr_avg_w2v, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

plt.plot(parameters['n_neighbors'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')


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



In [140]:
```
best_k = 120
```

In [141]:

```python
# 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_avg_w2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr_avg_w2v)
y_test_pred = batch_predict(neigh, X_te_avg_w2v)

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

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



1.8.8 Confusion matrix for avg_w2v

a) Train data

In [103]:
```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24752915426968986 for threshold 0.832
[[ 188  154]
 [ 682 1220]]
```

b) Test data

In [104]:
```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24842529604434369 for threshold 0.851
[[162  90]
 [719 679]]
```

1.9 Converting text to vectors (tfidf W2V encoding)

1.9.1 Using Pretrained Models: TFIDF weighted W2V

In [105]:

```
# 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 [106]:

```
# tfidf 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((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
    tfidf_w2v_vectors.append(vector)

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

```
100%|████████████████████████████████████████████████████████████| 5000/5000 [00:11<00:00, 434.74it/s]

5000
300
```

1.9.2 tfidf_Word2vec on train,test and cv datasets on titles and essays

a) For train essay

```
In [107]: tfidf_w2v_train_essay = []; # the avg-w2v for each preporcessed_titles is stored in this list
          for sentence in tqdm(X_train['preprocessed_essays']): # for each title
              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
              tfidf_w2v_train_essay.append(vector)

          print(len(tfidf_w2v_train_essay))
          print(len(tfidf_w2v_train_essay[0]))
```

```
100%|██████████████████████████████████████████████████| 2244/2244 [00:05<00:00, 437.39it/s]

2244
300
```

b) For train titles

```
In [108]: tfidf_w2v_train_titles = []; # the avg-w2v for each preporcessed_titles is stored in this list
          for sentence in tqdm(X_train['preprocessed_titles']): # for each title
              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
              tfidf_w2v_train_titles.append(vector)

          print(len(tfidf_w2v_train_titles))
          print(len(tfidf_w2v_train_titles[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 2244/2244 [00:00<00:00, 26802.34it/s]

2244
300
```

1.9.3 tfidf_w2v For test (essay and title)

a) For test essay

```
In [109]: tfidf_w2v_vectors_test_essay = []; # the avg-w2v for each preporcessed_titles is stored in this list
          for sentence in tqdm(X_test['preprocessed_titles']): # for each title
              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
              tfidf_w2v_vectors_test_essay.append(vector)

          print(len(tfidf_w2v_vectors_test_essay))
          print(len(tfidf_w2v_vectors_test_essay[0]))
```

```
100%|██████████████████████████████████████████████████| 1650/1650 [00:00<00:00, 27891.48it/s]

1650
300
```

b) For test title

In [110]:

```
tfidf_w2v_vectors_test_title = []; # the avg-w2v for each preporcessed_titles is stored in this list
for sentence in tqdm(X_test['preprocessed_titles']): # for each title
    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
    tfidf_w2v_vectors_test_title.append(vector)

print(len(tfidf_w2v_vectors_test_title))
print(len(tfidf_w2v_vectors_test_title[0]))
```

```
100%|██████████████████████████████████████████████████| 1650/1650 [00:00<00:00, 27438.53it/s]

1650
300
```

1.9.4 tfidf_w2v For cv (essay and title)

a) For CV train essay

```
In [111]: tfidf_w2v_vectors_tr_cv_essay = []; # the avg-w2v for each preporcessed_titles is stored in this list
          for sentence in tqdm(X_cv['preprocessed_essays']): # for each title
              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
              tfidf_w2v_vectors_tr_cv_essay.append(vector)

          print(len(tfidf_w2v_vectors_tr_cv_essay))
          print(len(tfidf_w2v_vectors_tr_cv_essay[0]))
```

```
100%|██████████████████████████████████████████████████| 1106/1106 [00:02<00:00, 432.67it/s]

1106
300
```

b) For CV test title

In [112]:
```python
tfidf_w2v_vectors_tr_cv_title = []; # the avg-w2v for each preporcessed_titles is stored in this list
for sentence in tqdm(X_cv['preprocessed_titles']): # for each title
    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
    tfidf_w2v_vectors_tr_cv_title.append(vector)

print(len(tfidf_w2v_vectors_tr_cv_title))
print(len(tfidf_w2v_vectors_tr_cv_title[0]))
```

```
100%|██████████████████████████████████████████████████| 1106/1106 [00:00<00:00, 27575.73it/s]

1106
300
```

1.9.5 Concatenating all the features of tfidf_w2v

In [113]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_tfidf_w2v = hstack((tfidf_w2v_train_essay,tfidf_w2v_train_titles, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_price_norm, X_train_teacher_no_of_previously_submitted_projects_standardized)).tocsr()
X_cr_tfidf_w2v = hstack((tfidf_w2v_vectors_tr_cv_essay,tfidf_w2v_vectors_tr_cv_title, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_price_norm, X_cv_teacher_no_of_previously_submitted_projects_standardized)).tocsr()
X_te_tfidf_w2v = hstack((tfidf_w2v_vectors_test_essay,tfidf_w2v_vectors_test_title, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_price_norm, X_test_teacher_no_of_previously_submitted_projects_standardized)).tocsr()

print("Final Data matrix")
print(X_tr_tfidf_w2v.shape, y_train.shape)
print(X_cr_tfidf_w2v.shape, y_cv.shape)
print(X_te_tfidf_w2v.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(2244, 660) (2244,)
(1106, 660) (1106,)
(1650, 660) (1650,)
====================================================================================================
```

Applying KNN on tfidf_w2v

knn = KNeighborsClassifier(n_neighbors=5) knn.fit(X_tr,y_train) pred_y = knn.predict(X_te_tfidf_w2v)

Confusion matrix for simple KNN

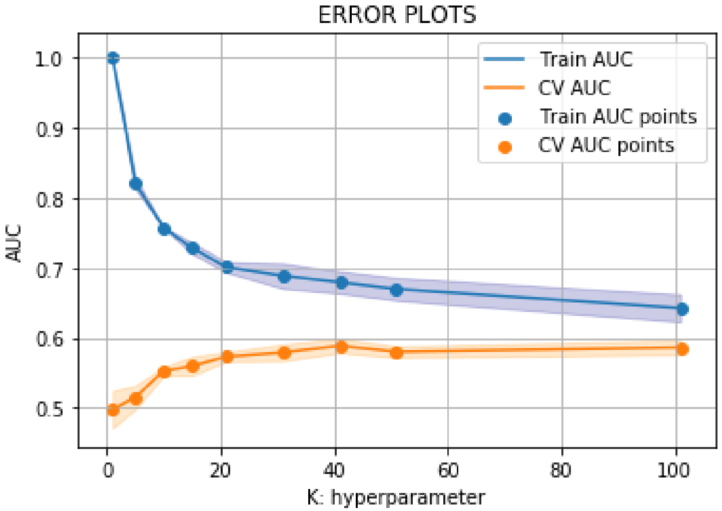
print(confusion_matrix(y_test,pred_y)) print(classification_report(y_test,pred_y))

1.9.6 Hypertuning parameter K

Gridsearch on tfidf_w2v data

In [114]:

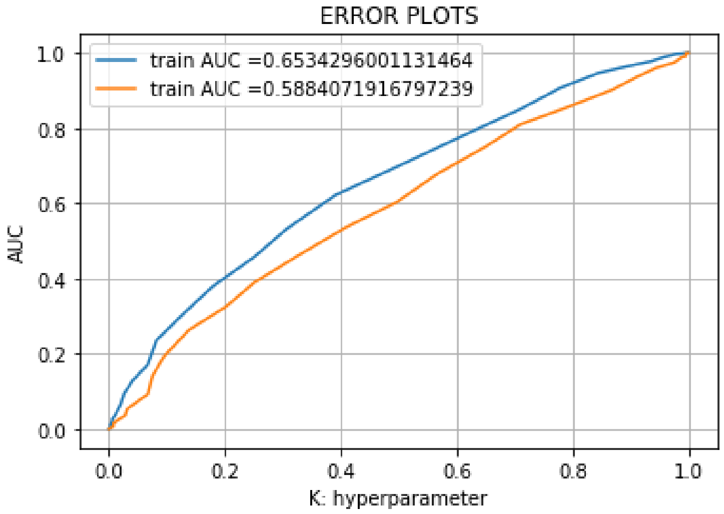
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV

neigh = KNeighborsClassifier()
parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51,101]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(X_tr_tfidf_w2v, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

plt.plot(parameters['n_neighbors'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')


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



In [138]:
```
best_k = 99
```

In [139]:

```
# 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_tfidf_w2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr_tfidf_w2v)
y_test_pred = batch_predict(neigh, X_te_tfidf_w2v)

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

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



1.9.7 Confusion matrix for tfidf_w2v

a) Train data

In [117]:

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

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2491450360794774 for threshold 0.822
[[ 181  161]
 [ 607 1295]]
```

b) Test data

In [118]:

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

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2496063240110859 for threshold 0.842
[[131 121]
 [581 817]]
```

From the analysis above we have found that tfidf works the best for KNN

Selecting the top 2000 features for tfidf

In [119]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_tfidf = hstack((X_train_essay_tfidf, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_price_norm,X_train_teacher_no_of_previously_submitted_projects_standardized)).tocsr()
X_cr_tfidf = hstack((X_cv_essay_tfidf, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_price_norm,X_cv_teacher_no_of_previously_submitted_projects_standardized)).tocsr()
X_te_tfidf = hstack((X_test_essay_tfidf, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_price_norm,X_test_teacher_no_of_previously_submitted_projects_standardized)).tocsr()

print("Final Data matrix")
print(X_tr_tfidf.shape, y_train.shape)
print(X_cr_tfidf.shape, y_cv.shape)
print(X_te_tfidf.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(2244, 5060) (2244,)
(1106, 5060) (1106,)
(1650, 5060) (1650,)
====================================================================================================
```

In [ ]:

```
#from sklearn.datasets import load_digits
#X_train2, y_test = load_digits(return_X_y=True)
#y_test1 = y_test.iloc[0]
#print(type(y_train1))
#print(y_train1)
#print(X_train2.shape, y_train.shape)
```

2.0 Selecting k_best features

In [123]:

```
from sklearn.datasets import load_digits
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.feature_selection import f_classif
```

In [126]:

```
# selecting the best 2000 features using SelectKBest from TFIDF model
best_feature = SelectKBest(f_classif, k = 2000)
best_feature.fit(X_tr_tfidf, y_train)
# selecting the best 2000 features for train, test and cross validation
X_tfidf_train_new = best_feature.transform(X_tr_tfidf)
X_tfidf_cv_new = best_feature.transform(X_cr_tfidf)
X_tfidf_test_new = best_feature.transform(X_te_tfidf)
```

```
C:\Users\Rahul\Anaconda3\lib\site-packages\sklearn\feature_selection\univariate_selection.py:114: UserWarning:

Features [0 0] are constant.
```

In [130]:

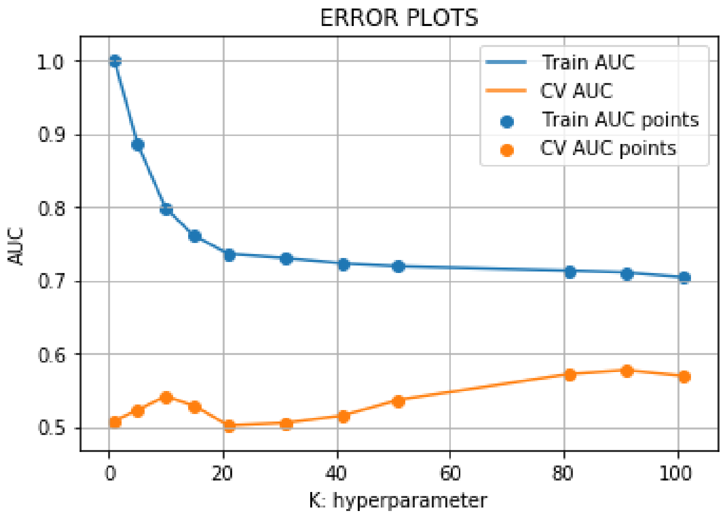
```
print(X_tfidf_train_new.shape)
print(X_tfidf_cv_new.shape)
print(X_tfidf_test_new.shape)
```

```
(2244, 2000)
(1106, 2000)
(1650, 2000)
```

In [131]:

```
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51, 81, 91, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_tfidf_train_new, y_train)

    y_train_pred = batch_predict(neigh, X_tfidf_train_new)
    y_cv_pred = batch_predict(neigh, X_tfidf_cv_new)

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

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

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

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

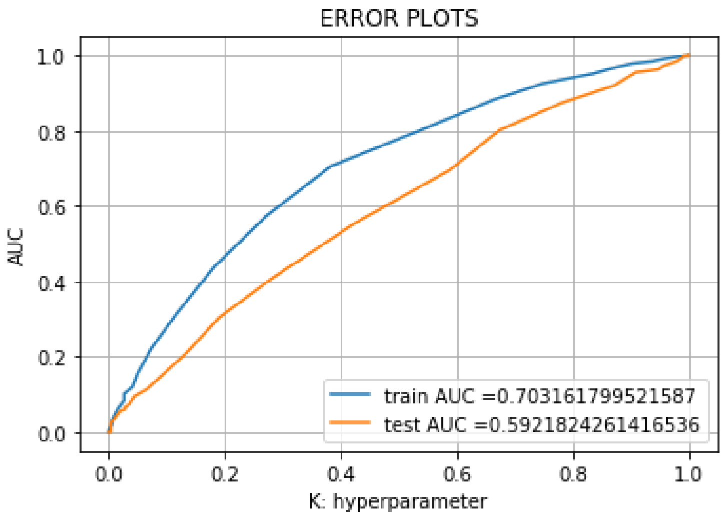
```
100%|████████████████████████████████████████| 11/11 [00:04<00:00,  2.27it/s]
```



```
In [132]: best_k = 103
```

X_tfidf_train_new = best_feature.transform(X_tr_tfidf) X_tfidf_cv_new = best_feature.transform(X_cr_tfidf) X_tfidf_test_new = best_feature.transform(X_te_tfidf)

```
In [137]: # 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_tfidf_train_new, y_train)
          # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
          # not the predicted outputs

          y_train_pred = batch_predict(neigh, X_tfidf_train_new)
          y_test_pred = batch_predict(neigh, X_tfidf_test_new)

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

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



Confusion Matrix

a) Train data

```
In [134]: print("="*100)
          print("Train confusion matrix")
          print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24855511097431687 for threshold 0.825
[[ 158  184]
 [ 378 1524]]
```

b) Test data

```
In [135]: print("Test confusion matrix for Test Data")
          print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
Test confusion matrix for Test Data
the maximum value of tpr*(1-fpr) 0.2443153187200806 for threshold 0.845
[[145 107]
 [623 775]]
```

Conclusion

In [144]:

```
from prettytable import PrettyTable

tb = PrettyTable()
tb.field_names= ("Vectorizer", "Model", "HyperParameter", "AUC")
tb.add_row(["BOW", "Auto", 99, 59])
tb.add_row(["Tf-Idf", "Auto", 102, 56])
tb.add_row(["AVG-W2v", "Auto", 120, 59])
tb.add_row(["Tf-Idf W2v", "Auto", 99, 57])
tb.add_row(["Tf-Idf KBest", "Auto", 103, 59])
print(tb.get_string(titles = "KNN - Observations"))
#print(tb)
```

```
+--------------+-------+----------------+-----+
|  Vectorizer  | Model | HyperParameter | AUC |
+--------------+-------+----------------+-----+
|     BOW      |  Auto |       99       |  59 |
|    Tf-Idf    |  Auto |      102       |  56 |
|   AVG-W2v    |  Auto |      120       |  59 |
|  Tf-Idf W2v  |  Auto |       99       |  57 |
| Tf-Idf KBest |  Auto |      103       |  59 |
+--------------+-------+----------------+-----+
```

Consider 5000 datapoints due to memory issues

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: