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	Desc
`project_id`	A unique identifier for the proposed project. **Example:** `p0`
`project_title`	Title of the project. **Exa** • `Art Will Make You H` • `First Grad`
`project_grade_category`	Grade level of students for which the project is targeted. One of the fo enumerated v • `Grades P` • `Grade` • `Grade` • `Grades`

Feature	Desc
project_subject_categories	One or more (comma-separated) subject categories for the project fr following enumerated list of v • Applied Lea • Care & H • Health & S • History & C • Literacy & Lan • Math & Sc • Music & The • Special • W **Exan** • Music & The • Literacy & Language, Math & Sc
school_state	State where school is located (Two-letter U.S. posta (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Postal_c **Exampl**
project_subject_subcategories	One or more (comma-separated) subject subcategories for the p **Exan** • Lit • Literature & Writing, Social Sci
project_resource_summary	An explanation of the resources needed for the project. **Exa** • My students need hands on literacy materials to ma sensory needs!<
project_essay_1	First application
project_essay_2	Second application
project_essay_3	Third application
project_essay_4	Fourth application
project_submitted_datetime	Datetime when project application was submitted. **Example:** 2016-0 12:43:5
teacher_id	A unique identifier for the teacher of the proposed project. **Exa** bdf8baa8fedef6bfeec7ae4ff1c
teacher_prefix	Teacher's title. One of the following enumerated v • • • • • • Tea
teacher_number_of_previously_posted_projects	Number of project applications previously submitted by the same te **Exam**

* 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`

Feature	Description
`quantity`	Quantity of the resource required. **Example:** `3`
`price`	Price of the resource required. **Example:** `9.95`

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

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

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

Notes on the Essay Data

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

- **project_essay_1:** "Introduce us to your classroom"
- **project_essay_2:** "Tell us more about your students"
- **project_essay_3:** "Describe how your students will use the materials you're requesting"
- **project_essay_3:** "Close by sharing why your project will make a difference"

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

- **project_essay_1:** "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."
- **project_essay_2:** "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

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

In [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
from nltk.corpus import stopwords
```

1.1 Reading Data

In [2]:

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

In [3]:

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

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

In [4]:

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


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


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


project_data.head(2)
```

Out[4]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	00:
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	00:

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

In [6]:

```
# https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexes-for-all-gr
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index

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

In [7]:

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

Out[7]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date
0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36
1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25

1.2 preprocessing of `project_subject_categories`

In [8]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/473019

# 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", "
        if 'The' in j.split(): # this will split each of the catogory based on space "Math
            j=j.replace('The','') # if we have the words "The" we are going to replace it w
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

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

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

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

1.3 preprocessing of project_subject_subcategories

In [9]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/473019

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

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

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

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

1.3 Text preprocessing

In [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	Date
0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36
1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25

In [12]:

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

```
I have been fortunate enough to use the Fairy Tale STEM kits in my classroom
as well as the STEM journals, which my students really enjoyed.  I would lov
e to implement more of the Lakeshore STEM kits in my classroom for the next
school year as they provide excellent and engaging STEM lessons.My students
come from a variety of backgrounds, including language and socioeconomic sta
tus.  Many of them don't have a lot of experience in science and engineering
and these kits give me the materials to provide these exciting opportunities
for my students.Each month I try to do several science or STEM/STEAM project
s.  I would use the kits and robot to help guide my science instruction in e
ngaging and meaningful ways.  I can adapt the kits to my current language ar
ts pacing guide where we already teach some of the material in the kits like
tall tales (Paul Bunyan) or Johnny Appleseed.  The following units will be t
aught in the next school year where I will implement these kits: magnets, mo
tion, sink vs. float, robots.  I often get to these units and don't know If
I am teaching the right way or using the right materials.    The kits will g
ive me additional ideas, strategies, and lessons to prepare my students in s
cience.It is challenging to develop high quality science activities.  These
kits give me the materials I need to provide my students with science activi
ties that will go along with the curriculum in my classroom.  Although I hav
e some things (like magnets) in my classroom, I don't know how to use them e
ffectively.  The kits will provide me with the right amount of materials and
show me how to use them in an appropriate way.
==================================================
I teach high school English to students with learning and behavioral disabil
ities. My students all vary in their ability level. However, the ultimate go
al is to increase all students literacy levels. This includes their reading,
writing, and communication levels.I teach a really dynamic group of student
s. However, my students face a lot of challenges. My students all live in po
verty and in a dangerous neighborhood. Despite these challenges, I have stud
ents who have the the desire to defeat these challenges. My students all hav
e learning disabilities and currently all are performing below grade level.
My students are visual learners and will benefit from a classroom that fulfi
lls their preferred learning style.The materials I am requesting will allow
my students to be prepared for the classroom with the necessary supplies.  T
oo often I am challenged with students who come to school unprepared for cla
ss due to economic challenges.  I want my students to be able to focus on le
arning and not how they will be able to get school supplies.  The supplies w
ill last all year.  Students will be able to complete written assignments an
d maintain a classroom journal.  The chart paper will be used to make learni
ng more visual in class and to create posters to aid students in their learn
ing.  The students have access to a classroom printer.  The toner will be us
ed to print student work that is completed on the classroom Chromebooks.I wa
nt to try and remove all barriers for the students learning and create oppor
tunities for learning. One of the biggest barriers is the students not havin
g the resources to get pens, paper, and folders. My students will be able to
increase their literacy skills because of this project.
==================================================
```

\"Life moves pretty fast. If you don't stop and look around once in awhile, you could miss it.\" from the movie, Ferris Bueller's Day Off. Think back...what do you remember about your grandparents? How amazing would it be to be able to flip through a book to see a day in their lives?My second graders are voracious readers! They love to read both fiction and nonfiction books. Their favorite characters include Pete the Cat, Fly Guy, Piggie and Elephant, and Mercy Watson. They also love to read about insects, space and plants. My students are hungry bookworms! My students are eager to learn and read about the world around them. My kids love to be at school and are like little sponges absorbing everything around them. Their parents work long hours and usually do not see their children. My students are usually cared for by their grandparents or a family friend. Most of my students do not have someone who speaks English at home. Thus it is difficult for my students to acquire language.Now think forward... wouldn't it mean a lot to your kids, nieces or nephews or grandchildren, to be able to see a day in your life today 30 years from now? Memories are so precious to us and being able to share these memories with future generations will be a rewarding experience. As part of our social studies curriculum, students will be learning about changes over time. Students will be studying photos to learn about how their community has changed over time. In particular, we will look at photos to study how the land, buildings, clothing, and schools have changed over time. As a culminating activity, my students will capture a slice of their history and preserve it through scrap booking. Key important events in their young lives will be documented with the date, location, and names. Students will be using photos from home and from school to create their second grade memories. Their scrap books will preserve their unique stories for future generations to enjoy.Your donation to this project will provide my second graders with an opportunity to learn about social studies in a fun and creative manner. Through their scrapbooks, children will share their story with others and have a historical document for the rest of their lives.
==
\"A person's a person, no matter how small.\" (Dr.Seuss) I teach the smallest students with the biggest enthusiasm for learning. My students learn in many different ways using all of our senses and multiple intelligences. I use a wide range of techniques to help all my students succeed. \r\nStudents in my class come from a variety of different backgrounds which makes for wonderful sharing of experiences and cultures, including Native Americans.\r\nOur school is a caring community of successful learners which can be seen through collaborative student project based learning in and out of the classroom. Kindergarteners in my class love to work with hands-on materials and have many different opportunities to practice a skill before it is mastered. Having the social skills to work cooperatively with friends is a crucial aspect of the kindergarten curriculum.Montana is the perfect place to learn about agriculture and nutrition. My students love to role play in our pretend kitchen in the early childhood classroom. I have had several kids ask me, \"Can we try cooking with REAL food?\" I will take their idea and create \"Common Core Cooking Lessons\" where we learn important math and writing concepts while cooking delicious healthy food for snack time. My students will have a grounded appreciation for the work that went into making the food and knowledge of where the ingredients came from as well as how it's healthy for their bodies. This project would expand our learning of nutrition and agricultural cooking recipes by having us peel our own apples to make homemade applesauce, make our own bread, and mix up healthy plants from our classroom garden in the spring. We will also create our own cookbooks to be printed and shared with families. \r\nStudents will gain math and literature skills as well as a life long enjoyment for healthy cooking.nannan
==

My classroom consists of twenty-two amazing sixth graders from different cultures and backgrounds. They are a social bunch who enjoy working in partners and working with groups. They are hard-working and eager to head to m

```
iddle school next year. My job is to get them ready to make this transitio
n and make it as smooth as possible. In order to do this, my students need
to come to school every day and feel safe and ready to learn. Because they
are getting ready to head to middle school, I give them lots of choice- ch
oice on where to sit and work, the order to complete assignments, choice o
f projects, etc. Part of the students feeling safe is the ability for them
to come into a welcoming, encouraging environment. My room is colorful and
the atmosphere is casual. I want them to take ownership of the classroom b
ecause we ALL share it together. Because my time with them is limited, I w
ant to ensure they get the most of this time and enjoy it to the best of t
heir abilities.Currently, we have twenty-two desks of differing sizes, yet
the desks are similar to the ones the students will use in middle school.
We also have a kidney table with crates for seating. I allow my students t
o choose their own spots while they are working independently or in group
s. More often than not, most of them move out of their desks and onto the
crates. Believe it or not, this has proven to be more successful than maki
ng them stay at their desks! It is because of this that I am looking towar
d the “Flexible Seating” option for my classroom.\r\n The students look fo
rward to their work time so they can move around the room. I would like to
get rid of the constricting desks and move toward more “fun” seating optio
ns. I am requesting various seating so my students have more options to si
t. Currently, I have a stool and a papasan chair I inherited from the prev
ious sixth-grade teacher as well as five milk crate seats I made, but I wo
uld like to give them more options and reduce the competition for the “goo
d seats”. I am also requesting two rugs as not only more seating options b
ut to make the classroom more welcoming and appealing. In order for my stu
dents to be able to write and complete work without desks, I am requesting
a class set of clipboards. Finally, due to curriculum that requires groups
to work together, I am requesting tables that we can fold up when we are n
ot using them to leave more room for our flexible seating options.\r\nI kn
ow that with more seating options, they will be that much more excited abo
ut coming to school! Thank you for your support in making my classroom one
students will remember forever!nannan
==================================================
```

In [13]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [14]:

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

```
\"A person is a person, no matter how small.\" (Dr.Seuss) I teach the smalle
st students with the biggest enthusiasm for learning. My students learn in m
any different ways using all of our senses and multiple intelligences. I use
a wide range of techniques to help all my students succeed. \r\nStudents in
my class come from a variety of different backgrounds which makes for wonder
ful sharing of experiences and cultures, including Native Americans.\r\nOur
school is a caring community of successful learners which can be seen throug
h collaborative student project based learning in and out of the classroom.
Kindergarteners in my class love to work with hands-on materials and have ma
ny different opportunities to practice a skill before it is mastered. Having
the social skills to work cooperatively with friends is a crucial aspect of
the kindergarten curriculum.Montana is the perfect place to learn about agri
culture and nutrition. My students love to role play in our pretend kitchen
in the early childhood classroom. I have had several kids ask me, \"Can we t
ry cooking with REAL food?\" I will take their idea and create \"Common Core
Cooking Lessons\" where we learn important math and writing concepts while c
ooking delicious healthy food for snack time. My students will have a ground
ed appreciation for the work that went into making the food and knowledge of
where the ingredients came from as well as how it is healthy for their bodie
s. This project would expand our learning of nutrition and agricultural cook
ing recipes by having us peel our own apples to make homemade applesauce, ma
ke our own bread, and mix up healthy plants from our classroom garden in the
spring. We will also create our own cookbooks to be printed and shared with
families. \r\nStudents will gain math and literature skills as well as a lif
e long enjoyment for healthy cooking.nannan
==================================================
```

In [15]:

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

```
 A person is a person, no matter how small.  (Dr.Seuss) I teach the smallest
students with the biggest enthusiasm for learning. My students learn in many
different ways using all of our senses and multiple intelligences. I use a w
ide range of techniques to help all my students succeed.   Students in my cl
ass come from a variety of different backgrounds which makes for wonderful s
haring of experiences and cultures, including Native Americans.  Our school
is a caring community of successful learners which can be seen through colla
borative student project based learning in and out of the classroom. Kinderg
arteners in my class love to work with hands-on materials and have many diff
erent opportunities to practice a skill before it is mastered. Having the so
cial skills to work cooperatively with friends is a crucial aspect of the ki
ndergarten curriculum.Montana is the perfect place to learn about agricultur
e and nutrition. My students love to role play in our pretend kitchen in the
early childhood classroom. I have had several kids ask me,  Can we try cooki
ng with REAL food?  I will take their idea and create  Common Core Cooking L
essons  where we learn important math and writing concepts while cooking del
icious healthy food for snack time. My students will have a grounded appreci
ation for the work that went into making the food and knowledge of where the
ingredients came from as well as how it is healthy for their bodies. This pr
oject would expand our learning of nutrition and agricultural cooking recipe
s by having us peel our own apples to make homemade applesauce, make our own
bread, and mix up healthy plants from our classroom garden in the spring. We
will also create our own cookbooks to be printed and shared with families.
Students will gain math and literature skills as well as a life long enjoyme
nt for healthy cooking.nannan
```

In [16]:

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

```
 A person is a person no matter how small Dr Seuss I teach the smallest stud
ents with the biggest enthusiasm for learning My students learn in many diff
erent ways using all of our senses and multiple intelligences I use a wide r
ange of techniques to help all my students succeed Students in my class come
from a variety of different backgrounds which makes for wonderful sharing of
experiences and cultures including Native Americans Our school is a caring c
ommunity of successful learners which can be seen through collaborative stud
ent project based learning in and out of the classroom Kindergarteners in my
class love to work with hands on materials and have many different opportuni
ties to practice a skill before it is mastered Having the social skills to w
ork cooperatively with friends is a crucial aspect of the kindergarten curri
culum Montana is the perfect place to learn about agriculture and nutrition
My students love to role play in our pretend kitchen in the early childhood
classroom I have had several kids ask me Can we try cooking with REAL food I
will take their idea and create Common Core Cooking Lessons where we learn i
mportant math and writing concepts while cooking delicious healthy food for
snack time My students will have a grounded appreciation for the work that w
ent into making the food and knowledge of where the ingredients came from as
well as how it is healthy for their bodies This project would expand our lea
rning of nutrition and agricultural cooking recipes by having us peel our ow
n apples to make homemade applesauce make our own bread and mix up healthy p
lants from our classroom garden in the spring We will also create our own co
okbooks to be printed and shared with families Students will gain math and l
iterature skills as well as a life long enjoyment for healthy cooking nannan
```

In [17]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they'
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'l
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'u
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'd
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', '
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'v
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'do
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn'
            'won', "won't", 'wouldn', "wouldn't"]
```

In [18]:

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

```
100%|████████████████████████████████| 109248/109248 [01:58<00:00, 918.96i
t/s]
```

In [19]:

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

Out[19]:

```
'a person person no matter small dr seuss i teach smallest students biggest
enthusiasm learning my students learn many different ways using senses multi
ple intelligences i use wide range techniques help students succeed students
class come variety different backgrounds makes wonderful sharing experiences
cultures including native americans our school caring community successful l
earners seen collaborative student project based learning classroom kinderga
rteners class love work hands materials many different opportunities practic
e skill mastered having social skills work cooperatively friends crucial asp
ect kindergarten curriculum montana perfect place learn agriculture nutritio
n my students love role play pretend kitchen early childhood classroom i sev
eral kids ask can try cooking real food i take idea create common core cooki
ng lessons learn important math writing concepts cooking delicious healthy f
ood snack time my students grounded appreciation work went making food knowl
edge ingredients came well healthy bodies this project would expand learning
nutrition agricultural cooking recipes us peel apples make homemade applesau
ce make bread mix healthy plants classroom garden spring we also create cook
books printed shared families students gain math literature skills well life
long enjoyment healthy cooking nannan'
```

1.4 Preprocessing of `project_title`

In [20]:

```
# similarly you can preprocess the titles also
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_titles = []
# tqdm is for printing the status bar
for sentence in tqdm(project_data['project_title'].values):
    sent = decontracted(sentence)
    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%|████████████████████████████████| 109248/109248 [00:05<00:00, 20646.71i
t/s]
```

In [21]:

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

1.5 Preprocessing of `teacher_prefix`

In [22]:

```
# we cannot remove rows where teacher prefix is not available therefore we are replacing 'n
# 'null'(string)
#https://stackoverflow.com/questions/42224700/attributeerror-float-object-has-no-attribute-
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna('null')

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

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

In [23]:

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

Out[23]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date
0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36
1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25

2 rows × 21 columns

In [24]:

```
# we cannot remove rows where school_state is not available therefore we are replacing 'nan
# 'null'(string)
#https://stackoverflow.com/questions/42224700/attributeerror-float-object-has-no-attribute-
project_data['school_state'] = project_data['school_state'].fillna('null')

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

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

In [25]:

```
# we cannot remove rows where project_grade_category is not available therefore we are repl
# 'null'(string)
#https://stackoverflow.com/questions/42224700/attributeerror-float-object-has-no-attribute-
project_data['project_grade_category'] = project_data['project_grade_category'].fillna('nul

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

# Delete element from Counter
del my_counter['Grades']
project_grade_category_dict = dict(my_counter)
sorted_project_grade_category_dict = dict(sorted(project_grade_category_dict.items(), key=l
```

In [26]:

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

Out[26]:

```
(109248, 21)
```

In [27]:

```
#to make dataset balanced
filtered_negative = project_data.loc[project_data['project_is_approved'] == 0]
#print(filtered_negative.count())

filtered_positive = project_data.loc[project_data['project_is_approved'] == 1]
sample_positive = filtered_positive.take(np.random.permutation(len(filtered_positive))[:165
```

In [28]:

```
print(filtered_negative.count())
```

```
Unnamed: 0                                        16542
id                                                16542
teacher_id                                        16542
teacher_prefix                                    16542
school_state                                      16542
Date                                              16542
project_grade_category                            16542
project_title                                     16542
project_essay_1                                   16542
project_essay_2                                   16542
project_essay_3                                     504
project_essay_4                                     504
project_resource_summary                          16542
teacher_number_of_previously_posted_projects      16542
project_is_approved                               16542
price                                             16542
quantity                                          16542
clean_categories                                  16542
clean_subcategories                               16542
essay                                             16542
clean_titles                                      16542
dtype: int64
```

In [29]:

```
print(sample_positive.count())
```

```
Unnamed: 0                                        16542
id                                                16542
teacher_id                                        16542
teacher_prefix                                    16542
school_state                                      16542
Date                                              16542
project_grade_category                            16542
project_title                                     16542
project_essay_1                                   16542
project_essay_2                                   16542
project_essay_3                                     577
project_essay_4                                     577
project_resource_summary                          16542
teacher_number_of_previously_posted_projects      16542
project_is_approved                               16542
price                                             16542
quantity                                          16542
clean_categories                                  16542
clean_subcategories                               16542
essay                                             16542
clean_titles                                      16542
dtype: int64
```

In [30]:

```
project_data = pd.concat([filtered_negative, sample_positive]).sort_index(kind='merge')
```

In [31]:

```
project_data = project_data.sample(frac=0.3)
```

In [32]:

```
project_data.count()
```

Out[32]:

```
Unnamed: 0                                      9925
id                                              9925
teacher_id                                      9925
teacher_prefix                                  9925
school_state                                    9925
Date                                            9925
project_grade_category                          9925
project_title                                   9925
project_essay_1                                 9925
project_essay_2                                 9925
project_essay_3                                  331
project_essay_4                                  331
project_resource_summary                        9925
teacher_number_of_previously_posted_projects    9925
project_is_approved                             9925
price                                           9925
quantity                                        9925
clean_categories                                9925
clean_subcategories                             9925
essay                                           9925
clean_titles                                    9925
dtype: int64
```

Splitting data

In [33]:

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

In [34]:

```
# train test split
#https://www.kaggle.com/shashank49/donors-choose-knn?scriptVersionId=11898125
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=
```

In [35]:

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

print("="*100)
```

```
(4454, 20) (4454,)
(2195, 20) (2195,)
(3276, 20) (3276,)
====================================================================================================
```

1.5 Preparing data for models

In [36]:

```
project_data.columns
```

Out[36]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', 'project_title', 'project_essay_1',
       'project_essay_2', 'project_essay_3', 'project_essay_4',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'price', 'quantity',
       'clean_categories', 'clean_subcategories', 'essay', 'clean_titles'],
      dtype='object')
```

In [37]:

```
X_train.columns
```

Out[37]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', 'project_title', 'project_essay_1',
       'project_essay_2', 'project_essay_3', 'project_essay_4',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'price', 'quantity',
       'clean_categories', 'clean_subcategories', 'essay', 'clean_titles'],
      dtype='object')
```

we are going to consider

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

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

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

1.5.1 Vectorizing Categorical data

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

In [38]:

```
# we use count vectorizer to convert the values into one
#one hot encoding the catogorical features: clean_categories

vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
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_clean_cat_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_cv_clean_cat_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_clean_cat_ohe = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_clean_cat_ohe.shape, y_train.shape)
print(X_cv_clean_cat_ohe.shape, y_cv.shape)
print(X_test_clean_cat_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(4454, 32) (4454,)
(2195, 32) (2195,)
(3276, 32) (3276,)
['appliedlearning', 'appliedlearning health_sports', 'appliedlearning litera
cy_language', 'appliedlearning math_science', 'appliedlearning music_arts',
'appliedlearning specialneeds', 'care_hunger', 'health_sports', 'health_spor
ts literacy_language', 'health_sports math_science', 'health_sports specialn
eeds', 'history_civics', 'history_civics literacy_language', 'history_civics
music_arts', 'history_civics specialneeds', 'literacy_language', 'literacy_l
anguage appliedlearning', 'literacy_language history_civics', 'literacy_lang
uage math_science', 'literacy_language music_arts', 'literacy_language speci
alneeds', 'math_science', 'math_science appliedlearning', 'math_science heal
th_sports', 'math_science history_civics', 'math_science literacy_language',
'math_science music_arts', 'math_science specialneeds', 'music_arts', 'speci
alneeds', 'warmth', 'warmth care_hunger']
====================================================================================================
```

In [39]:

```
# we use count vectorizer to convert the values into one
#one hot encoding the catogorical features: clean_subcategories

vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen only on train dat


# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_subcat_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
X_cv_clean_subcat_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_clean_subcat_ohe = vectorizer.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_clean_subcat_ohe.shape, y_train.shape)
print(X_cv_clean_subcat_ohe.shape, y_cv.shape)
print(X_test_clean_subcat_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(4454, 83) (4454,)
(2195, 83) (2195,)
(3276, 83) (3276,)
['appliedsciences', 'appliedsciences college_careerprep', 'appliedsciences e
nvironmentalscience', 'appliedsciences health_lifescience', 'appliedsciences
literacy', 'appliedsciences literature_writing', 'appliedsciences mathematic
s', 'appliedsciences specialneeds', 'appliedsciences visualarts', 'care_hung
er', 'charactereducation', 'charactereducation literacy', 'charactereducatio
n specialneeds', 'civics_government', 'civics_government history_geography',
'college_careerprep', 'college_careerprep literature_writing', 'college_care
erprep mathematics', 'communityservice', 'earlydevelopment', 'earlydevelopme
nt health_wellness', 'earlydevelopment literacy', 'earlydevelopment literatu
re_writing', 'earlydevelopment mathematics', 'earlydevelopment specialneed
s', 'economics', 'environmentalscience', 'environmentalscience health_lifesc
ience', 'environmentalscience history_geography', 'environmentalscience lite
racy', 'environmentalscience mathematics', 'esl', 'esl literacy', 'esl liter
ature_writing', 'esl mathematics', 'esl specialneeds', 'extracurricular', 'f
inancialliteracy', 'foreignlanguages', 'gym_fitness', 'gym_fitness health_we
llness', 'gym_fitness teamsports', 'health_lifescience', 'health_lifescience
mathematics', 'health_lifescience specialneeds', 'health_wellness', 'health_
wellness literacy', 'health_wellness mathematics', 'health_wellness nutritio
neducation', 'health_wellness specialneeds', 'health_wellness teamsports',
'history_geography', 'history_geography literacy', 'history_geography litera
ture_writing', 'history_geography socialsciences', 'history_geography visual
arts', 'literacy', 'literacy literature_writing', 'literacy mathematics', 'l
iteracy socialsciences', 'literacy specialneeds', 'literacy visualarts', 'li
terature_writing', 'literature_writing mathematics', 'literature_writing soc
ialsciences', 'literature_writing specialneeds', 'literature_writing visuala
rts', 'mathematics', 'mathematics specialneeds', 'mathematics visualarts',
'music', 'music performingarts', 'nutritioneducation', 'other', 'other speci
alneeds', 'parentinvolvement', 'performingarts', 'socialsciences', 'specialn
eeds', 'teamsports', 'visualarts', 'warmth', 'warmth care_hunger']
====================================================================================
========================
```

In [40]:

```
# you can do the similar thing with state, teacher_prefix and project_grade_category also
#one hot encoding the catogorical features: teacher_prefix

vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
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
(4454, 4) (4454,)
(2195, 4) (2195,)
(3276, 4) (3276,)
['mr', 'mrs', 'ms', 'teacher']
====================================================================================================
```

In [41]:

```
#state one hot encoding
#one hot encoding the catogorical features: school_state
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
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
(4454, 47) (4454,)
(2195, 47) (2195,)
(3276, 47) (3276,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'i
a', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo',
'ms', 'mt', 'nc', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'r
i', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'wa', 'wi', 'wv']
====================================================================================================
```

In [42]:

```
#project gradecategory one hot encoding
#one hot encoding the catogorical features: project_grade_category

vectorizer = CountVectorizer(vocabulary=list(sorted_project_grade_category_dict.keys()), lo
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only on train

# 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
(4454, 4) (4454,)
(2195, 4) (2195,)
(3276, 4) (3276,)
['9-12', '6-8', '3-5', 'PreK-2']
====================================================================================
========================
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [43]:

```
# We are considering only the words which appeared in at least 10 documents(rows or project
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=1000)
vectorizer.fit(X_train['essay'].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['essay'].values)
X_cv_essay_bow = vectorizer.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer.transform(X_test['essay'].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)
```

```
After vectorizations
(4454, 1000) (4454,)
(2195, 1000) (2195,)
(3276, 1000) (3276,)
====================================================================================
========================
```

In [44]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=1000)
vectorizer.fit(X_train['clean_titles'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_titles_bow = vectorizer.transform(X_train['clean_titles'].values)
X_cv_titles_bow = vectorizer.transform(X_cv['clean_titles'].values)
X_test_titles_bow = vectorizer.transform(X_test['clean_titles'].values)

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

```
After vectorizations
(4454, 408) (4454,)
(2195, 408) (2195,)
(3276, 408) (3276,)
====================================================================================
========================
```

In [45]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=1000)
vectorizer.fit(X_train['project_resource_summary'].values) # fit has to happen only on trai

# we use the fitted CountVectorizer to convert the text to vector
X_train_summary_bow = vectorizer.transform(X_train['project_resource_summary'].values)
X_cv_summary_bow = vectorizer.transform(X_cv['project_resource_summary'].values)
X_test_summary_bow = vectorizer.transform(X_test['project_resource_summary'].values)

print("After vectorizations")
print(X_train_summary_bow.shape, y_train.shape)
print(X_cv_summary_bow.shape, y_cv.shape)
print(X_test_summary_bow.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(4454, 1000) (4454,)
(2195, 1000) (2195,)
(3276, 1000) (3276,)
====================================================================================
========================
```

1.5.3 Vectorizing Numerical features

In [46]:

```
#price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_inde
#X_train = pd.merge(X_train, price_data, on='id', how='left')
```

In [47]:

```
X_train.head(2)
```

Out[47]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	
103641	44907	p176411	b8d7b9d975ccf1a540c2c64aff29f4bf	Ms.	OR	0
23637	4785	p196080	848931389ce9cadd2537f4b22c59dd54	Mr.	MI	0

In [48]:

```
from sklearn.preprocessing import StandardScaler
standard_vec = StandardScaler(with_mean = False)
# 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.
standard_vec.fit(X_train['price'].values.reshape(-1,1))

X_train_price_std = standard_vec.transform(X_train['price'].values.reshape(-1,1))
X_cv_price_std = standard_vec.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_std = standard_vec.transform(X_test['price'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_price_std.shape, y_train.shape)
print(X_cv_price_std.shape, y_cv.shape)
print(X_test_price_std.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(4454, 1) (4454,)
(2195, 1) (2195,)
(3276, 1) (3276,)
====================================================================================================
```

In [49]:

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

X_train_projects_std = standard_vec.transform(X_train['teacher_number_of_previously_posted_
X_cv_projects_std = standard_vec.transform(X_cv['teacher_number_of_previously_posted_projec
X_test_projects_std = standard_vec.transform(X_test['teacher_number_of_previously_posted_pr

print("After vectorizations")
print(X_train_projects_std.shape, y_train.shape)
print(X_cv_projects_std.shape, y_cv.shape)
print(X_test_projects_std.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(4454, 1) (4454,)
(2195, 1) (2195,)
(3276, 1) (3276,)
====================================================================================================
```

In [50]:

```
#function to get heatmap confusion matrix
def get_confusion_matrix(clf,X_te,y_test):
    y_pred = clf.predict(X_te)
    df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(2),range(2))
    df_cm.columns = ['Predicted NO','Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})
    sns.set(font_scale=1.4)#for label size
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

1.5.4 Merging all the above features

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

In [51]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_bow,X_train_titles_bow,X_train_clean_cat_ohe,X_train_clean_sub
X_cr = hstack((X_cv_essay_bow,X_cv_titles_bow,X_cv_clean_cat_ohe,X_cv_clean_subcat_ohe, X_c
X_te = hstack((X_test_essay_bow,X_test_titles_bow,X_test_clean_cat_ohe,X_test_clean_subcat_

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

```
Final Data matrix
(4454, 1580) (4454,)
(2195, 1580) (2195,)
(3276, 1580) (3276,)
====================================================================================
========================
```

Applying KNN

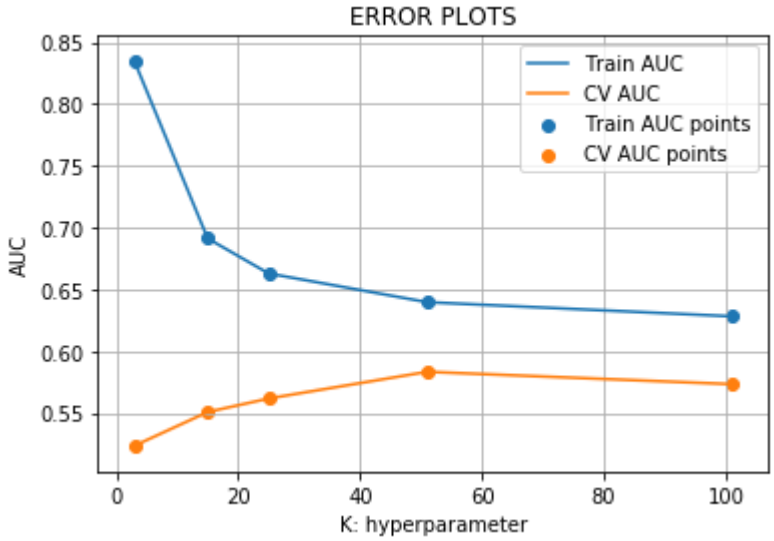
In [52]:

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

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 = 4900
    # 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
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [53]:

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

train_auc = []
cv_auc = []
K = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X_tr, y_train)

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

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # 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%|██████████████████████████████████████████████████| 5/5 [00:28<00:00,  5.67s/
it]
```

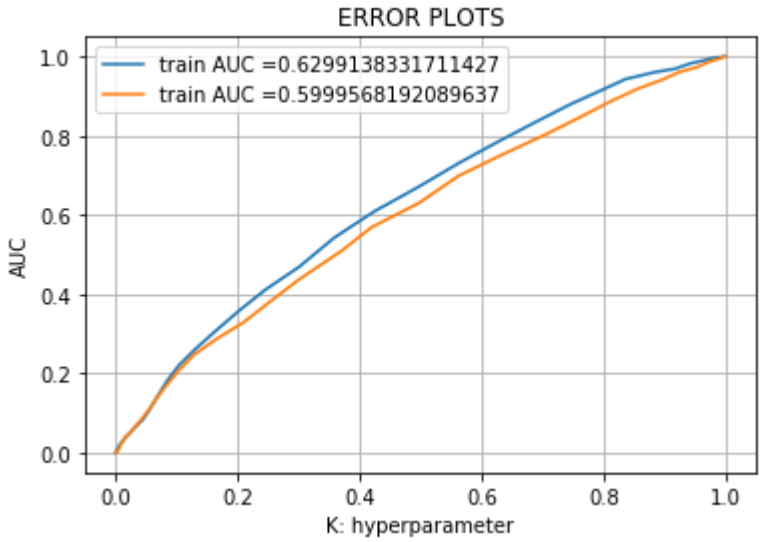


In [54]:

```
# from the error plot we choose K such that, we will have maximum AUC on cv data and gap be
# Note: based on the method you use you might get different hyperparameter values as best o
# so, you choose according to the method you choose, you use gridsearch if you are having m
# if you increase the cv values in the GridSearchCV you will get more rebust results.

#here we are choosing the best_k based on forloop results
best_k = 95
```

In [55]:

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


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

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

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="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()
```



In [56]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

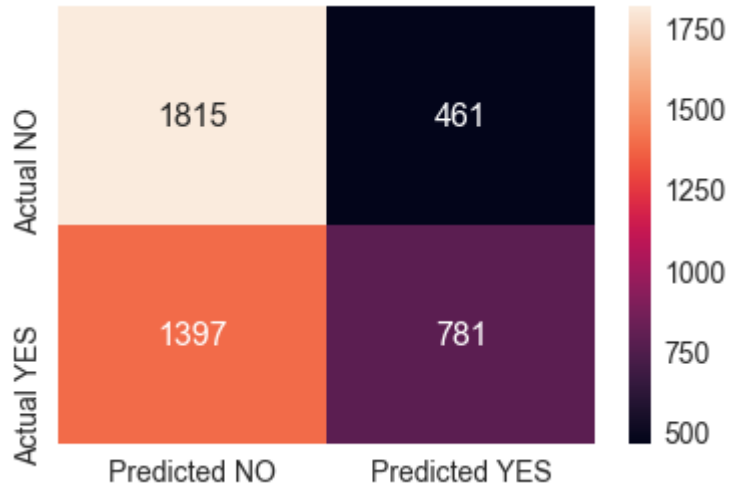
In [57]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
====================================================================================================
========================
the maximum value of tpr*(1-fpr) 0.3506808781213638 for threshold 0.463
Train confusion matrix
[[1310  966]
 [ 851 1327]]
Test confusion matrix
[[971 703]
 [690 912]]
```

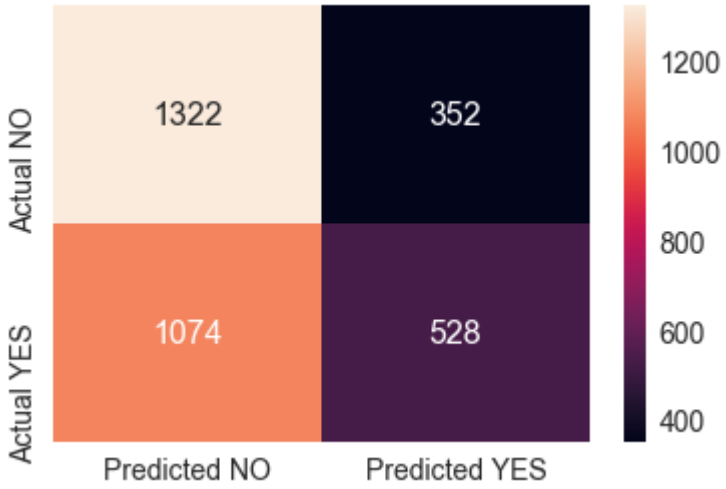
In [58]:

```
get_confusion_matrix(neigh,X_tr,y_train)
```



In [59]:

```
get_confusion_matrix(neigh,X_te,y_test)
```



Applying KNN brute force on TFIDF

In [60]:

```
%%time
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=2000)
vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data

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


#Selecting top 2000 best features from the generated tfidf features
selector = SelectKBest(chi2, k = 2000 )
selector.fit(X_train_essay_tfidf,y_train)
X_train_essay_2000 = selector.transform(X_train_essay_tfidf)
X_cv_essay_2000 = selector.transform(X_cv_essay_tfidf)
X_test_essay_2000 = selector.transform(X_test_essay_tfidf)
print(X_train_essay_2000.shape)
print(X_cv_essay_2000.shape)
print(X_test_essay_2000.shape)
```

```
(4454, 2000)
(2195, 2000)
(3276, 2000)
Wall time: 33.8 s
```

In [61]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2

vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=1000)
vectorizer.fit(X_train['clean_titles'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_titles_tfidf = vectorizer.transform(X_train['clean_titles'].values)
X_cv_titles_tfidf = vectorizer.transform(X_cv['clean_titles'].values)
X_test_titles_tfidf = vectorizer.transform(X_test['clean_titles'].values)
print("Train shape:",X_train_titles_tfidf.shape)
print("CV shape:",X_cv_titles_tfidf.shape)
print("Test shape:",X_test_titles_tfidf.shape)
```

```
Train shape: (4454, 408)
CV shape: (2195, 408)
Test shape: (3276, 408)
```

In [62]:

```
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=1000)
vectorizer.fit(X_train['project_resource_summary'].values) # fit has to happen only on trai

# we use the fitted CountVectorizer to convert the text to vector
X_train_summary_tfidf = vectorizer.transform(X_train['project_resource_summary'].values)
X_cv_summary_tfidf = vectorizer.transform(X_cv['project_resource_summary'].values)
X_test_summary_tfidf = vectorizer.transform(X_test['project_resource_summary'].values)

print("After vectorizations")
print(X_train_summary_tfidf.shape, y_train.shape)
print(X_cv_summary_tfidf.shape, y_cv.shape)
print(X_test_summary_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(4454, 1000) (4454,)
(2195, 1000) (2195,)
(3276, 1000) (3276,)
====================================================================================
========================
```

In [63]:

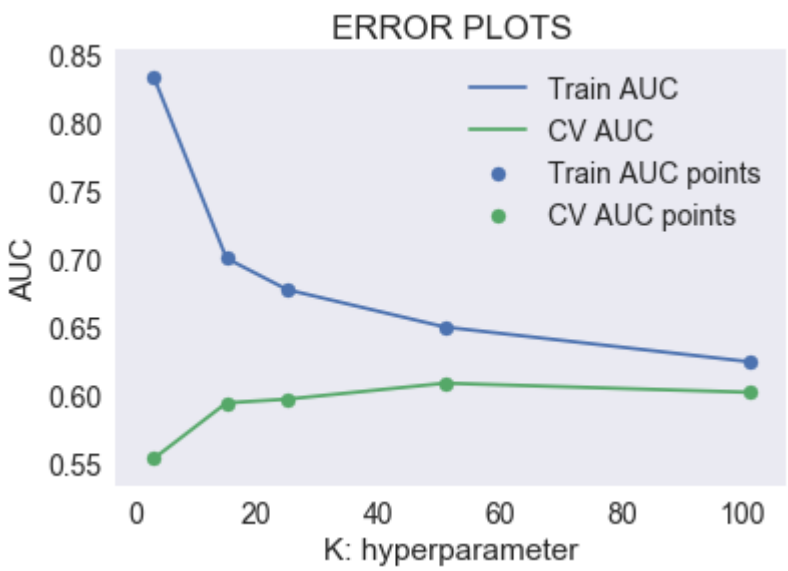
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_2000,X_train_titles_tfidf,X_train_summary_tfidf,X_train_clean_
X_cr = hstack((X_cv_essay_2000,X_cv_titles_tfidf,X_cv_summary_tfidf,X_cv_clean_cat_ohe,X_cv
X_te = hstack((X_test_essay_2000,X_test_titles_tfidf,X_test_summary_tfidf,X_test_clean_cat_

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

```
Final Data matrix
(4454, 3580) (4454,)
(2195, 3580) (2195,)
(3276, 3580) (3276,)
====================================================================================
========================
```

In [64]:

```
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
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 = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X_tr, y_train)

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

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # 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%|██████████████████████████████████████████| 5/5 [00:34<00:00,  6.83s/
it]
```

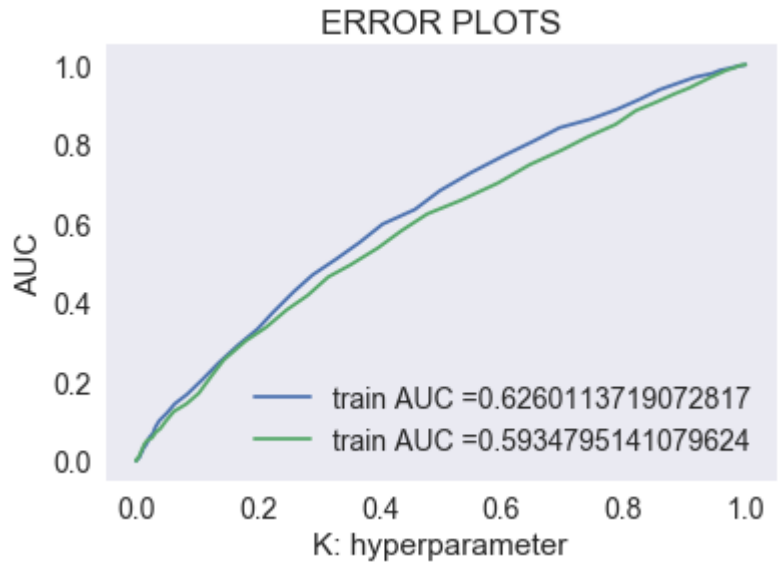


In [65]:

```
# from the error plot we choose K such that, we will have maximum AUC on cv data and gap be
# Note: based on the method you use you might get different hyperparameter values as best o
# so, you choose according to the method you choose, you use gridsearch if you are having m
# if you increase the cv values in the GridSearchCV you will get more rebust results.

#here we are choosing the best_k based on forloop results
best_k = 95
```

In [66]:

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


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

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

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="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()
```



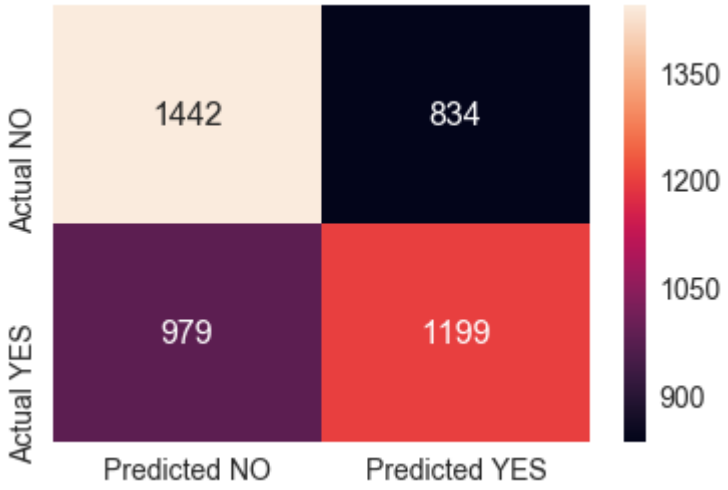
In [67]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
====================================================================================================
========================
the maximum value of tpr*(1-fpr) 0.3556202301009778 for threshold 0.495
Train confusion matrix
[[1355  921]
 [ 877 1301]]
Test confusion matrix
[[945 729]
 [672 930]]
```

In [68]:

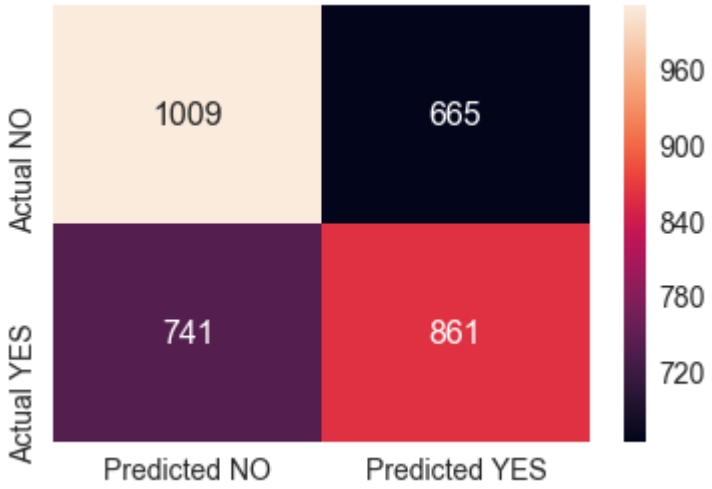
```
get_confusion_matrix(neigh,X_tr,y_train)
```



In [69]:

```
%%time
get_confusion_matrix(neigh,X_te,y_test)
```

```
Wall time: 4.39 s
```



Using Pretrained Models: Avg W2V

In [70]:

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

In [71]:

```
# average Word2Vec
# compute average word2vec for each review.
train_w2v_vectors_essays = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_train['essay'].values): # for each essay in training data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_w2v_vectors_essays.append(vector)
print("train  vector")
print(len(train_w2v_vectors_essays))
print(len(train_w2v_vectors_essays[0]))
print('='*50)

# average Word2Vec
# compute average word2vec for each review.
test_w2v_vectors_essays = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_test['essay'].values): # for each essay in training data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_w2v_vectors_essays.append(vector)

print("Test vec")
print(len(test_w2v_vectors_essays))
print(len(test_w2v_vectors_essays[0]))
print('='*50)

# average Word2Vec
# compute average word2vec for each review.
cv_w2v_vectors_essays = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_cv['essay'].values): # for each essay in training data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_w2v_vectors_essays.append(vector)

print("CV vec")
print(len(cv_w2v_vectors_essays))
print(len(cv_w2v_vectors_essays[0]))
print('='*50)
```

```
100%|████████████████████████████████████| 4454/4454 [00:04<00:00, 1109.00i
t/s]
```

```
train  vector
4454
50
==================================================

100%|████████████████████████████████████| 3276/3276 [00:02<00:00, 1118.02i
t/s]

Test vec
3276
50
==================================================

100%|████████████████████████████████████| 2195/2195 [00:02<00:00, 1095.79i
t/s]

CV vec
2195
50
==================================================
```

In [72]:

```
# Changing list to numpy arrays
train_w2v_vectors_essays = np.array(train_w2v_vectors_essays)
test_w2v_vectors_essays = np.array(test_w2v_vectors_essays)
cv_w2v_vectors_essays = np.array(cv_w2v_vectors_essays)
```

In [73]:

```
# average Word2Vec
# compute average word2vec for each review.
train_w2v_vectors_titles = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_train['clean_titles'].values): # for each essay in training data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_w2v_vectors_titles.append(vector)
print("train  vector")
print(len(train_w2v_vectors_titles))
print(len(train_w2v_vectors_titles[0]))
print('='*50)

# average Word2Vec
# compute average word2vec for each review.
test_w2v_vectors_titles = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_test['clean_titles'].values): # for each essay in training data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_w2v_vectors_titles.append(vector)

print("Test vec")
print(len(test_w2v_vectors_titles))
print(len(test_w2v_vectors_titles[0]))
print('='*50)

# average Word2Vec
# compute average word2vec for each review.
cv_w2v_vectors_titles = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_cv['clean_titles'].values): # for each essay in training data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_w2v_vectors_titles.append(vector)

print("CV vec")
print(len(cv_w2v_vectors_titles))
print(len(cv_w2v_vectors_titles[0]))
print('='*50)
```

```
100%|████████████████████████████████████| 4454/4454 [00:00<00:00, 27663.00i
t/s]
```

```
train  vector
4454
50
==================================================

100%|████████████████████████████████████| 3276/3276 [00:00<00:00, 20732.99i
t/s]

Test vec
3276
50
==================================================

100%|████████████████████████████████████| 2195/2195 [00:00<00:00, 26129.51i
t/s]

CV vec
2195
50
==================================================
```

In [74]:

```
# Changing list to numpy arrays
train_w2v_vectors_titles = np.array(train_w2v_vectors_titles)
test_w2v_vectors_titles = np.array(test_w2v_vectors_titles)
cv_w2v_vectors_titles = np.array(cv_w2v_vectors_titles)
```

In [75]:

```
# average Word2Vec
# compute average word2vec for each review.
train_w2v_vectors_summary = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_train['project_resource_summary'].values): # for each essay in train
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_w2v_vectors_summary.append(vector)
print("train  vector")
print(len(train_w2v_vectors_summary))
print(len(train_w2v_vectors_summary[0]))
print('='*50)

# average Word2Vec
# compute average word2vec for each review.
test_w2v_vectors_summary = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_test['project_resource_summary'].values): # for each essay in traini
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_w2v_vectors_summary.append(vector)

print("Test vec")
print(len(test_w2v_vectors_summary))
print(len(test_w2v_vectors_summary[0]))
print('='*50)

# average Word2Vec
# compute average word2vec for each review.
cv_w2v_vectors_summary = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_cv['project_resource_summary'].values): # for each essay in training
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_w2v_vectors_summary.append(vector)

print("CV vec")
print(len(cv_w2v_vectors_summary))
print(len(cv_w2v_vectors_summary[0]))
print('='*50)
```

```
100%|████████████████████████████████████| 4454/4454 [00:00<00:00, 11449.21i
t/s]
```

```
train  vector
4454
50
==================================================

100%|████████████████████████████████████| 3276/3276 [00:00<00:00, 10740.37i
t/s]

Test vec
3276
50
==================================================

100%|████████████████████████████████████| 2195/2195 [00:00<00:00, 11141.50i
t/s]

CV vec
2195
50
==================================================
```

In [76]:

```
# Changing list to numpy arrays
train_w2v_vectors_summary = np.array(train_w2v_vectors_summary)
test_w2v_vectors_summary = np.array(test_w2v_vectors_summary)
cv_w2v_vectors_summary = np.array(cv_w2v_vectors_summary)
```

In [77]:

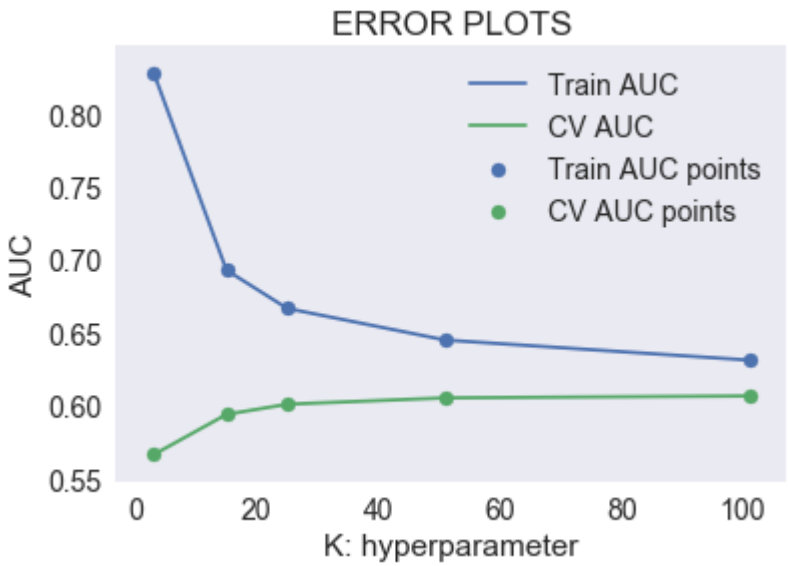
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((train_w2v_vectors_essays,train_w2v_vectors_titles,train_w2v_vectors_summary,
X_cr = hstack((cv_w2v_vectors_essays,cv_w2v_vectors_titles,cv_w2v_vectors_summary,X_cv_clea
X_te = hstack((test_w2v_vectors_essays,test_w2v_vectors_titles,test_w2v_vectors_summary,X_t

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

```
Final Data matrix
(4454, 322) (4454,)
(2195, 322) (2195,)
(3276, 322) (3276,)
====================================================================================
========================
```

In [78]:

```
train_auc = []
cv_auc = []
K = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X_tr, y_train)

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

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # 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%|██████████████████████████████████████████████████| 5/5 [01:06<00:00, 13.18s/
it]
```

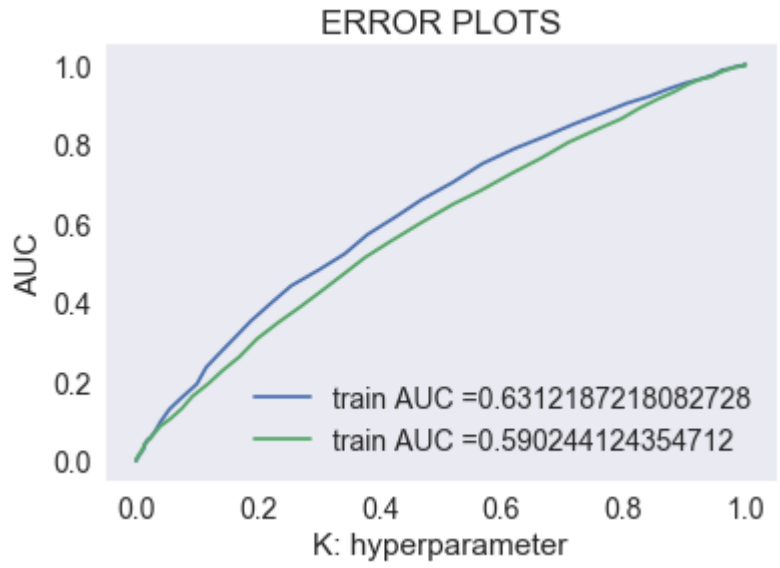


In [79]:

```
best_k = 101
```

In [80]:

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


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

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

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="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()
```



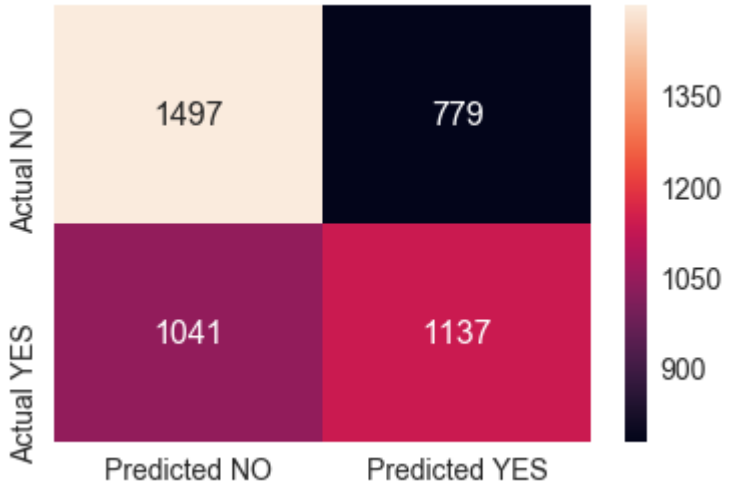
In [81]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
====================================================================================================
========================
the maximum value of tpr*(1-fpr) 0.35409293445720996 for threshold 0.495
Train confusion matrix
[[1411  865]
 [ 934 1244]]
Test confusion matrix
[[974 700]
 [714 888]]
```

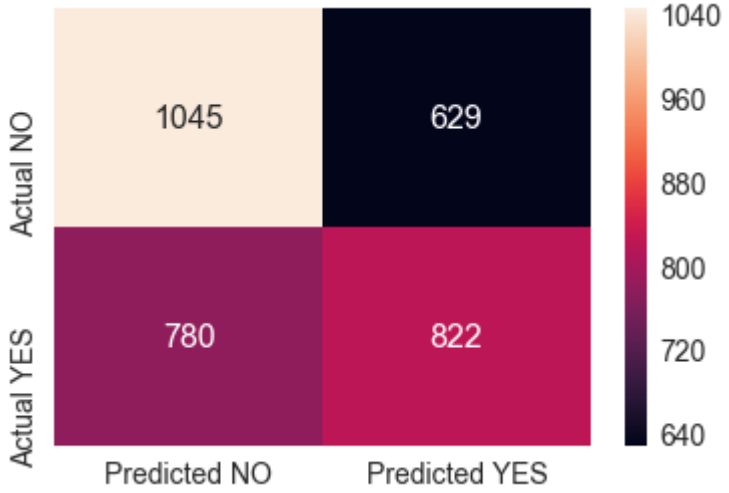
In [82]:

```
get_confusion_matrix(neigh,X_tr,y_train)
```



In [83]:

```
get_confusion_matrix(neigh,X_te,y_test)
```



TFIDF W2V

In [84]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['essay'].values)
# 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())


# average Word2Vec
# compute average word2vec for each review.
train_tfidf_w2v_essays = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essay'].values): # for each review/sentence
    vector = np.zeros(50) # 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][:50] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_tfidf_w2v_essays.append(vector)

print("Train matrix:")
print(len(train_tfidf_w2v_essays))
print(len(train_tfidf_w2v_essays[0]))
print('='*50)

cv_tfidf_w2v_essays = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['essay'].values): # for each review/sentence
    vector = np.zeros(50) # 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][:50] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    cv_tfidf_w2v_essays.append(vector)

print("CV matrix:")
print(len(cv_tfidf_w2v_essays))
print(len(cv_tfidf_w2v_essays[0]))
print('='*50)

test_tfidf_w2v_essays = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay'].values): # for each review/sentence
    vector = np.zeros(50) # 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][:50] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
```

```
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_tfidf_w2v_essays.append(vector)

print("Test matrix:")
print(len(test_tfidf_w2v_essays))
print(len(test_tfidf_w2v_essays[0]))
print('='*50)
```

```
100%|████████████████████████████████████| 4454/4454 [00:40<00:00, 111.24i
t/s]

Train matrix:
4454
50
==================================================

100%|████████████████████████████████████| 2195/2195 [00:19<00:00, 113.02i
t/s]

CV matrix:
2195
50
==================================================

100%|████████████████████████████████████| 3276/3276 [00:29<00:00, 112.21i
t/s]

Test matrix:
3276
50
==================================================
```

In [85]:

```
# Changing list to numpy arrays
train_tfidf_w2v_essays = np.array(train_tfidf_w2v_essays)
test_tfidf_w2v_essays = np.array(test_tfidf_w2v_essays)
cv_tfidf_w2v_essays = np.array(cv_tfidf_w2v_essays)
```

In [86]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['clean_titles'].values)
# 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())


# average Word2Vec
# compute average word2vec for each review.
train_tfidf_w2v_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_titles'].values): # for each review/sentence
    vector = np.zeros(50) # 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][:50] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_tfidf_w2v_titles.append(vector)

print("Train matrix:")
print(len(train_tfidf_w2v_titles))
print(len(train_tfidf_w2v_titles[0]))
print('='*50)

cv_tfidf_w2v_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_titles'].values): # for each review/sentence
    vector = np.zeros(50) # 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][:50] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    cv_tfidf_w2v_titles.append(vector)

print("CV matrix:")
print(len(cv_tfidf_w2v_titles))
print(len(cv_tfidf_w2v_titles[0]))
print('='*50)

test_tfidf_w2v_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_titles'].values): # for each review/sentence
    vector = np.zeros(50) # 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][:50] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
```

```
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_tfidf_w2v_titles.append(vector)

print("Test matrix:")
print(len(test_tfidf_w2v_titles))
print(len(test_tfidf_w2v_titles[0]))
print('='*50)
```

```
100%|████████████████████████████████████| 4454/4454 [00:00<00:00, 13060.83i
t/s]

Train matrix:
4454
50
==================================================

100%|████████████████████████████████████| 2195/2195 [00:00<00:00, 13064.72i
t/s]

CV matrix:
2195
50
==================================================

100%|████████████████████████████████████| 3276/3276 [00:00<00:00, 13480.70i
t/s]

Test matrix:
3276
50
==================================================
```

In [87]:

```
# Changing list to numpy arrays
train_tfidf_w2v_titles = np.array(train_tfidf_w2v_titles)
test_tfidf_w2v_titles = np.array(test_tfidf_w2v_titles)
cv_tfidf_w2v_titles = np.array(cv_tfidf_w2v_titles)
```

In [88]:

```
5# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['project_resource_summary'].values)
# 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())


# average Word2Vec
# compute average word2vec for each review.
train_tfidf_w2v_summary = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['project_resource_summary'].values): # for each review/sentenc
    vector = np.zeros(50) # 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][:50] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_tfidf_w2v_summary.append(vector)

print("Train matrix:")
print(len(train_tfidf_w2v_summary))
print(len(train_tfidf_w2v_summary[0]))
print('='*50)

cv_tfidf_w2v_summary = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['project_resource_summary'].values): # for each review/sentence
    vector = np.zeros(50) # 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][:50] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    cv_tfidf_w2v_summary.append(vector)

print("CV matrix:")
print(len(cv_tfidf_w2v_summary))
print(len(cv_tfidf_w2v_summary[0]))
print('='*50)

test_tfidf_w2v_summary = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['project_resource_summary'].values): # for each review/sentence
    vector = np.zeros(50) # 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][:50] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
```

```
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_tfidf_w2v_summary.append(vector)

print("Test matrix:")
print(len(test_tfidf_w2v_summary))
print(len(test_tfidf_w2v_summary[0]))
print('='*50)
```

```
100%|████████████████████████████████████| 4454/4454 [00:01<00:00, 3856.06i
t/s]

Train matrix:
4454
50
==================================================

100%|████████████████████████████████████| 2195/2195 [00:00<00:00, 3726.44i
t/s]

CV matrix:
2195
50
==================================================

100%|████████████████████████████████████| 3276/3276 [00:00<00:00, 3975.50i
t/s]

Test matrix:
3276
50
==================================================
```

In [89]:

```
# Changing list to numpy arrays
train_tfidf_w2v_summary = np.array(train_tfidf_w2v_summary)
test_tfidf_w2v_summary = np.array(test_tfidf_w2v_summary)
cv_tfidf_w2v_summary = np.array(cv_tfidf_w2v_summary)
```

In [90]:

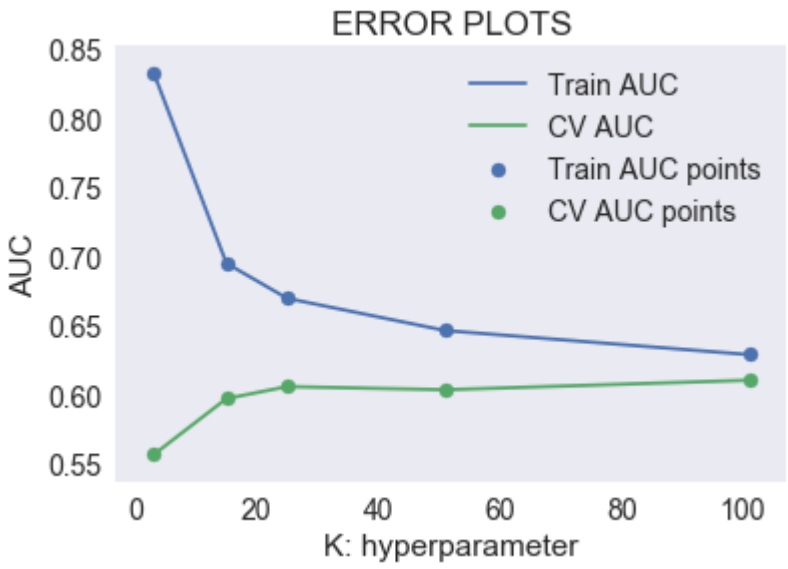
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((train_tfidf_w2v_essays,train_tfidf_w2v_titles,train_tfidf_w2v_summary,X_trai
X_cr = hstack((cv_tfidf_w2v_essays,cv_tfidf_w2v_titles,cv_tfidf_w2v_summary,X_cv_clean_cat_
X_te = hstack((test_tfidf_w2v_essays,test_tfidf_w2v_titles,test_tfidf_w2v_summary,X_test_cl

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

```
Final Data matrix
(4454, 322) (4454,)
(2195, 322) (2195,)
(3276, 322) (3276,)
====================================================================================
========================
```

In [91]:

```
train_auc = []
cv_auc = []
K = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X_tr, y_train)

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

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # 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%|██████████████████████████████████████████████████| 5/5 [01:08<00:00, 13.69s/
it]
```

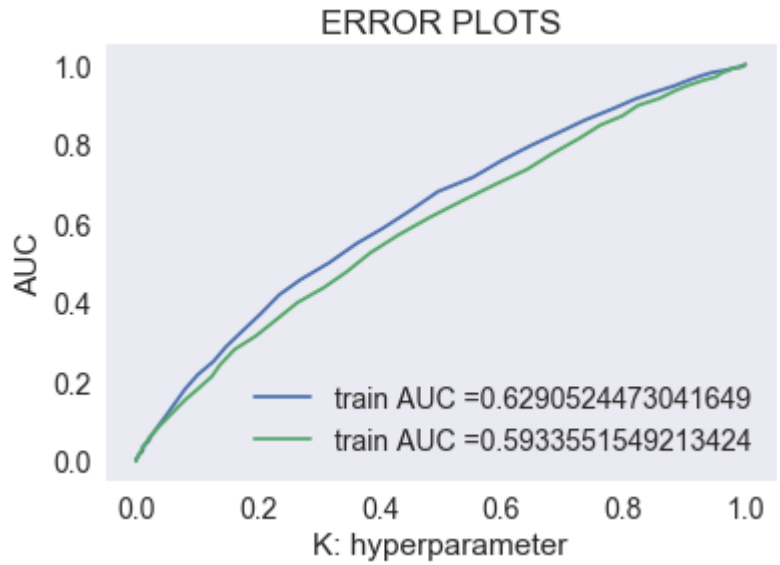


In [92]:

```
best_k = 101
```

In [93]:

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


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

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

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="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()
```



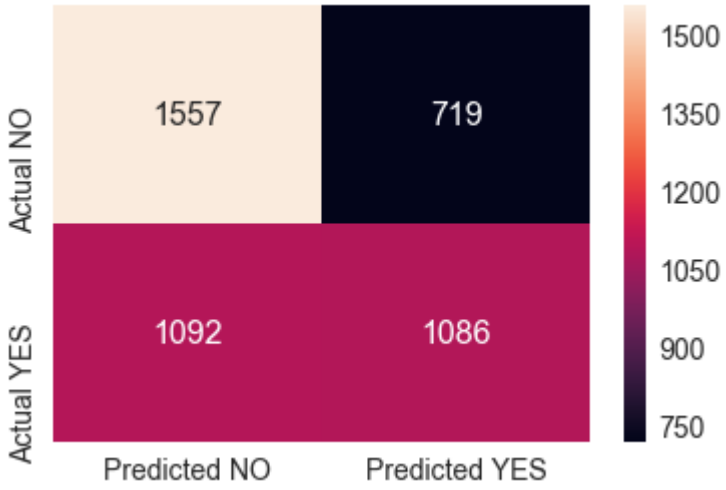
In [94]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
====================================================================================================
========================
the maximum value of tpr*(1-fpr) 0.34983966522550963 for threshold 0.495
Train confusion matrix
[[1450  826]
 [ 982 1196]]
Test confusion matrix
[[1031  643]
 [ 762  840]]
```

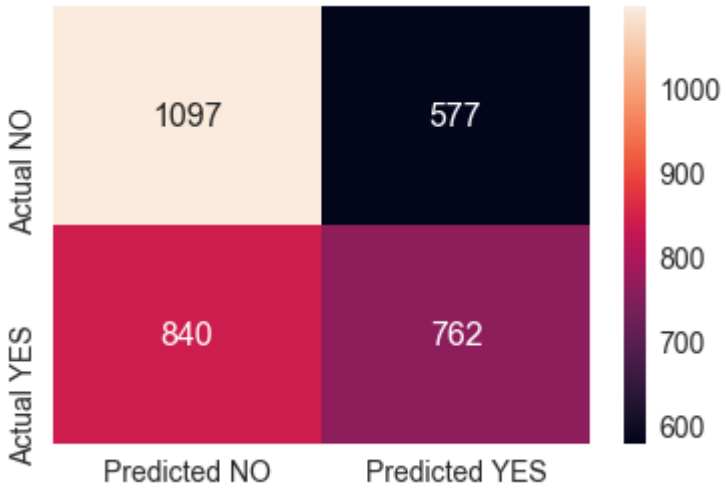
In [95]:

```
get_confusion_matrix(neigh,X_tr,y_train)
```



In [96]:

```
get_confusion_matrix(neigh,X_te,y_test)
```



Conclusion

In [97]:

```
#http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer", "Hyperparameter", "AUC"]
x.add_row(["Bag of Words", 95, 0.61])
x.add_row(["TFIDF", 95, 0.64])
x.add_row(["TFIDF", 101, 0.64])
x.add_row(["TFIDF", 101, 0.64])

print(x)
```

```
+--------------+----------------+------+
|  Vectorizer  | Hyperparameter | AUC  |
+--------------+----------------+------+
| Bag of Words |       95       | 0.61 |
|    TFIDF     |       95       | 0.64 |
|    TFIDF     |      101       | 0.64 |
|    TFIDF     |      101       | 0.64 |
+--------------+----------------+------+
```

In [ ]: