DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located (Two-letter U.S. postal code). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy` • `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • `My students need hands on literacy materials to manage sensory needs!`
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*

Feature	Description
`project_essay_4`	Fourth application essay
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
`teacher_id`	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same teacher. **Example:** `2`

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

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

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

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

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

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

Notes on the Essay Data

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

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

```
# Citation https://www.kaggle.com/shashank49/donors-choose-knn
# I referenced few parts of my code from above link
```

In [2]:

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

import sqlite3
```

```
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

import time
from tqdm import tqdm
import os
import pickle

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

1.1 Reading Data

In [3]:

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

In [4]:

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

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

In [5]:

```
# 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[5]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	

In [6]:

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

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

# Join two data frames
project_data = pd.merge(project_data, price_data, on='id', how='left')
project_data.head(5)
```

```
Number of data points in train data (1541272, 4)
['id' 'description' 'quantity' 'price']
        id                                           description  quantity  \
0  p233245  LC652 - Lakeshore Double-Space Mobile Drying Rack         1
1  p069063          Bouncy Bands for Desks (Blue support pipes)         3

    price
0  149.00
1   14.95
```

Out[6]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_subje
0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	M
1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	
2	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	2016-04-27 00:46:53	Grades PreK-2	Litera
3	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2	Ap
4	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5	Litera

1.2 preprocessing of

1.2 preprocessing of `project_subject_categories`

In [7]:

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

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

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [8]:

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

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

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

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

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

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

1.3 Text preprocessing

In [9]:

```
# 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 [10]:

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

Out[10]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_title
0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	Engineering STEAM into the Primary Classroom
1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	Sensory Tools for Focus

In [11]:

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

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[10000])
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 j
ournals, which my students really enjoyed.  I would love to implement more of the Lakeshore STEM k
its 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 statu
s.  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 projects.  I would use the kits and robot to help guide my science i
nstruction in engaging and meaningful ways.  I can adapt the kits to my current language arts paci
ng 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 taught in the next school year where I will
implement these kits: magnets, motion, sink vs. float, robots.  I often get to these units and don
't know If I am teaching the right way or using the right materials.    The kits will give me
additional ideas, strategies, and lessons to prepare my students in science.It is challenging to d
evelop high quality science activities.  These kits give me the materials I need to provide my
students with science activities that will go along with the curriculum in my classroom.  Although
I have some things (like magnets) in my classroom, I don't know how to use them effectively.  The
kits will provide me with the right amount of materials and show me how to use them in an
appropriate way
```

appropriate way.
==========
I teach high school English to students with learning and behavioral disabilities. My students all vary in their ability level. However, the ultimate goal is to increase all students literacy levels. This includes their reading, writing, and communication levels.I teach a really dynamic group of students. However, my students face a lot of challenges. My students all live in poverty and in a dangerous neighborhood. Despite these challenges, I have students who have the the desire to defeat these challenges. My students all have learning disabilities and currently all are performing below grade level. My students are visual learners and will benefit from a classroom that fulfills their preferred learning style.The materials I am requesting will allow my students to be prepared for the classroom with the necessary supplies. Too often I am challenged with students who come to school unprepared for class due to economic challenges. I want my students to be able to focus on learning and not how they will be able to get school supplies. The supplies will last all year. Students will be able to complete written assignments and maintain a classroom journal. The chart paper will be used to make learning more visual in class and to create posters to aid students in their learning. The students have access to a classroom printer. The toner will be used to print student work that is completed on the classroom Chromebooks.I want to try and remove all barriers for the students learning and create opportunities for learning. One of the biggest barriers is the students not having the resources to get pens, paper, and folders. My students will be able to increase their literacy skills because of this project.
==========
Take a minute and think about how you like to work. Do you prefer to sit in a chair at a table, or would you rather curl up on the couch with your book or computer in your lap? Everyone prefers to learn and work in different ways! Kindergarten and 1st Grade students are the same way!\r\n\r\nMy students are 5 and 6 year olds who are eager to come to school and learn! Many of my kindergarten students have never attended school before that first day that they step into my classroom.\r\nThese students are energetic and need to be moving constantly throughout the day. We have a high population of English Language Learners and they need many opportunities to communicate and collaborate with their peers.\r\nMy students will be able to use these different seating options throughout our classroom. I have chosen cushions with a cart because these cushions can give students the option to work wherever they want and ensure that they are comfortable while doing it. The cart will be necessary to efficiently store the cushions so that they will last us longer. The large pillows will provide a comfortable place for my students to sit and work. They will also be able to use them to lay down and read or do their work. My students will have choices of where they want to sit in the classroom, as well as what they want to sit on. These cushions and pillows will benefit students during every subject area throughout the day.\r\n\r\nThese seating options can change my students' lives by helping them have choices in their learning experience.\r\nIt will help them become more engaged and invested in their learning. Wouldn't you be excited about learning if you could choose the way that's best for you?!nannan
==========
\"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 middle school next year. My job is to get them ready to make this transition 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- choice on where to sit and work, the order to complete assignments, choice of 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 because we ALL share it together. Because my time with them is limited, I want to ensure they get the most of this time and enjoy it to the best of their 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 to choose their own spots while they are working independently or in groups. 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 making them stay at their desks! It is because of this that I am looking toward the “Flexible Seating” option for my classroom.\r\n The students look forward to their work time so they can move around the room. I would like to get rid of the c

look forward to their work time so they can move around the room. I would like to get rid of the c
onstricting desks and move toward more “fun” seating options. I am requesting various seating so m
y students have more options to sit. Currently, I have a stool and a papasan chair I inherited fro
m the previous sixth-grade teacher as well as five milk crate seats I made, but I would like to gi
ve them more options and reduce the competition for the “good seats”. I am also requesting two rug
s as not only more seating options but to make the classroom more welcoming and appealing. In orde
r for my students 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 t
ables that we can fold up when we are not using them to leave more room for our flexible seating o
ptions.\r\nI know that with more seating options, they will be that much more excited about 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 smallest students with the b
iggest enthusiasm for learning. My students learn in many different ways using all of our senses a
nd multiple intelligences. I use a wide range of techniques to help all my students succeed. \r\nS
tudents 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 su
ccessful learners which can be seen through collaborative student project based learning in and ou
t 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 wor
k 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 coo
king 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 knowled
ge of where the ingredients came from as well as how it is healthy for their bodies. This project
would expand our learning of nutrition and agricultural cooking recipes by having us peel our own
apples to make homemade applesauce, make our own bread, and mix up healthy plants from our classro
om garden in the spring. We will also create our own cookbooks to be printed and shared with famil
ies. \r\nStudents will gain math and literature skills as well as a life long enjoyment for health
y 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 big
gest 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. 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 community of successful learners which can be seen through collaborative student project based learning in a nd 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 t o 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 p retend 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 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 knowled ge of where the ingredients came from as well as how it is healthy for their bodies. This project would expand our learning of nutrition and agricultural cooking recipes by having us peel our own apples to make homemade applesauce, make our own bread, and mix up healthy plants from our classro om garden in the spring. We will also create our own cookbooks to be printed and shared with famil ies.   Students will gain math and literature skills as well as a life long enjoyment 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 students with the biggest enthusiasm for learning My students learn in many different ways using all of our senses and multi ple intelligences I use a wide range 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 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 knowled ge of where the ingredients came from as well as how it is healthy for their bodies This project w ould expand our learning of nutrition and agricultural cooking recipes by having us peel our own a pples 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 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've",
\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "d
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
```

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

In [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.lower() not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████| 109248/109248 [01:01<00:00, 1765.25it/s]
```

In [19]:

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

Out[19]:

```
'person person no matter small dr seuss teach smallest students biggest enthusiasm learning
students learn many different ways using senses multiple intelligences use wide range techniques h
elp students succeed students class come variety different backgrounds makes wonderful sharing exp
eriences cultures including native americans school caring community successful learners seen coll
aborative student project based learning classroom kindergarteners class love work hands materials
many different opportunities practice skill mastered social skills work cooperatively friends cruc
ial aspect kindergarten curriculum montana perfect place learn agriculture nutrition students love
role play pretend kitchen early childhood classroom several kids ask try cooking real food take id
ea create common core cooking lessons learn important math writing concepts cooking delicious heal
thy food snack time students grounded appreciation work went making food knowledge ingredients cam
e well healthy bodies project would expand learning nutrition agricultural cooking recipes us peel
apples make homemade applesauce make bread mix healthy plants classroom garden spring also create
cookbooks printed shared families students gain math literature skills well life long enjoyment he
althy cooking nannan'
```

1.4 Preprocessing of `project_title`

In [20]:

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

```
100%|██████████| 109248/109248 [00:02<00:00, 37362.09it/s]
```

In [21]:

```
# after preprocessing
print(preprocessed_titles[20000])
```

```
health nutritional cooking kindergarten
```

In [22]:

```
# Preprocess teacher_prefix
from tqdm import tqdm
preprocessed_teacher_prefix = []
# tqdm is for printing the status bar
for teacher_prefix in tqdm(project_data['teacher_prefix'].values):
    teacher_prefix = str(teacher_prefix)
    clean_teacher_prefix = decontracted(teacher_prefix)
    clean_teacher_prefix = clean_teacher_prefix.replace('\\r', ' ')
    clean_teacher_prefix = clean_teacher_prefix.replace('\\"', ' ')
    clean_teacher_prefix = clean_teacher_prefix.replace('\\n', ' ')
    clean_teacher_prefix = re.sub('[^A-Za-z0-9]+', ' ', clean_teacher_prefix)
    if clean_teacher_prefix in stopwords:
        continue
    preprocessed_teacher_prefix.append(clean_teacher_prefix.lower().strip())
```

```
100%|██████████| 109248/109248 [00:01<00:00, 59723.95it/s]
```

In [23]:

```
preprocessed_teacher_prefix[0:10]
```

Out[23]:

```
['mrs', 'ms', 'mrs', 'mrs', 'mrs', 'mrs', 'mrs', 'ms', 'ms', 'mrs']
```

In [24]:

```
# Preprocess project_grade_category
from tqdm import tqdm
preprocessed_project_grade_category = []
# tqdm is for printing the status bar
for project_grade_category in tqdm(project_data['project_grade_category'].values):
    project_grade_category = str(project_grade_category)
    clean_project_grade_category = decontracted(project_grade_category)
    clean_project_grade_category = clean_project_grade_category.replace('\\r', ' ')
    clean_project_grade_category = clean_project_grade_category.replace('\\"', ' ')
    clean_project_grade_category = clean_project_grade_category.replace('\\n', ' ')
    clean_project_grade_category = re.sub('[^A-Za-z0-9]+', ' ', clean_project_grade_category)
    if clean_project_grade_category in stopwords:
        continue
    preprocessed_project_grade_category.append(clean_project_grade_category.lower().strip())
```

```
100%|██████████| 109248/109248 [00:01<00:00, 65029.43it/s]
```

In [25]:

```
preprocessed_project_grade_category[0:10]
```

Out[25]:

```
['grades prek 2',
 'grades 3 5',
 'grades prek 2',
 'grades prek 2',
 'grades 3 5',
 'grades 3 5',
 'grades 3 5',
 'grades 3 5',
 'grades prek 2',
 'grades 3 5']
```

In [26]:

```
# Replace original columns with preprocessed column values
project_data['clean_essays'] = preprocessed_essays
```

```
project_data['clean_titles'] = preprocessed_titles
project_data['teacher_prefix'] = preprocessed_teacher_prefix
# project_data['project_grade_category'] = preprocessed_project_grade_category
# Drop essays column
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
project_data.drop(['project_essay_4'], axis=1, inplace=True)
```

In [27]:

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

Out[27]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_title
0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	mrs	CA	2016-04-27 00:27:36	Grades PreK-2	Engineering STEAM into the Primary Classroom
1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	ms	UT	2016-04-27 00:31:25	Grades 3-5	Sensory Tools for Focus

In [28]:

```
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna('null')
```

In [29]:

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

Out[29]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_title
0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	mrs	CA	2016-04-27 00:27:36	Grades PreK-2	Engineering STEAM into the Primary Classroom
1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	ms	UT	2016-04-27 00:31:25	Grades 3-5	Sensory Tools for Focus

1.5 Preparing data for models

In [30]:

```
project_data.columns
```

Out[30]:

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

```
       'clean_essays', '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
```

In [31]:

```
print(project_data.shape)

# I am taking 50% of data points for my analysis
project_data = project_data.sample(frac=0.2)

print(project_data.shape)
```

```
(109248, 18)
(21850, 18)
```

In [32]:

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

Out[32]:

```
(21850, 17)
```

In [33]:

```
# Split Train, CV and Test data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)

print('Train Data Set', X_train.shape, y_train.shape)
print('Cross Validate Data Set', X_cv.shape, y_cv.shape)
print('Test Data Set', X_test.shape, y_test.shape)
```

```
Train Data Set (9808, 17) (9808,)
Cross Validate Data Set (4831, 17) (4831,)
Test Data Set (7211, 17) (7211,)
```

In [34]:

```
# Handle imblanced data set
from imblearn.over_sampling import RandomOverSampler
from collections import Counter

ros = RandomOverSampler(sampling_strategy='minority', random_state=42)
X_train, y_train = ros.fit_resample(X_train, y_train)
print('Resampled Dataset Shape %s ' %Counter(y_train))
```

```
X_train = pd.DataFrame(X_train, columns=X.columns)
X_train.head(2)
```

```
Resampled Dataset Shape Counter({1: 8321, 0: 8321})
```

Out[34]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_title
0	130482	p095253	bf89bebdad18cedccaf6f6c8878042fb	mrs	IN	2016-05-17 13:59:37	Grades PreK-2	This is One Fine Project
1	74646	p019631	b46629abcbb2c9b5b510bcde98f8eb53	mrs	SC	2017-01-01 21:17:18	Grades 3-5	Enhance Learning With Lego Blocks and a Chrome...

In [35]:

```
print('Train Data Set', X_train.shape, y_train.shape)
print('Cross Validate Data Set', X_cv.shape, y_cv.shape)
print('Test Data Set', X_test.shape, y_test.shape)
print('*'*100)
```

```
Train Data Set (16642, 17) (16642,)
Cross Validate Data Set (4831, 17) (4831,)
Test Data Set (7211, 17) (7211,)
****************************************************************************************************
```

1.5.1 Vectorizing Categorical data

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

In [36]:

```
# One hot encoding of Categorical Feature
# - school_state : categorical data
vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values)# Fit has to happen only on train data

X_train_school_state_ohe = vectorizer.transform(X_train['school_state'].values)
X_cv_school_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_school_state_ohe = vectorizer.transform(X_test['school_state'].values)

print(X_train_school_state_ohe.shape, y_train.shape)
print(X_cv_school_state_ohe.shape, y_cv.shape)
print(X_test_school_state_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print('*'*100)
```

```
(16642, 51) (16642,)
(4831, 51) (4831,)
(7211, 51) (7211,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'k
s', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm',
'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv
', 'wy']
****************************************************************************************************
```

In [37]:

```
# One hot encoding of Categorical Feature
# - clean_categories : categorical data
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values)# Fit has to happen only on train data

X_train_clean_categories_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_cv_clean_categories_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_clean_categories_ohe = vectorizer.transform(X_test['clean_categories'].values)

print(X_train_clean_categories_ohe.shape, y_train.shape)
print(X_cv_clean_categories_ohe.shape, y_cv.shape)
print(X_test_clean_categories_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print('*'*100)
```

```
(16642, 9) (16642,)
(4831, 9) (4831,)
(7211, 9) (7211,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
****************************************************************************************************
```

In [38]:

```
# One hot encoding of Categorical Feature
# - clean_subcategories : categorical data
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values)# Fit has to happen only on train data

X_train_clean_subcategories_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
X_cv_clean_subcategories_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_clean_subcategories_ohe = vectorizer.transform(X_test['clean_subcategories'].values)

print(X_train_clean_subcategories_ohe.shape, y_train.shape)
print(X_cv_clean_subcategories_ohe.shape, y_cv.shape)
print(X_test_clean_subcategories_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print('*'*100)
```

```
(16642, 30) (16642,)
(4831, 30) (4831,)
(7211, 30) (7211,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience',
'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness',
'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'm
athematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socia
lsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
****************************************************************************************************
```

In [39]:

```
# One hot encoding of Categorical Feature
# - project_grade_category : categorical data
# Convert one hot encoding for project grade category
vectorizer = CountVectorizer(vocabulary=set(preprocessed_project_grade_category), lowercase=False,
binary=True)
vectorizer.fit(X_train['project_grade_category'].values)# Fit has to happen only on train data

X_train_project_grade_category_ohe = vectorizer.transform(X_train['project_grade_category'].values
)
X_cv_project_grade_category_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_project_grade_category_ohe = vectorizer.transform(X_test['project_grade_category'].values)

print(X_train_project_grade_category_ohe.shape, y_train.shape)
print(X_cv_project_grade_category_ohe.shape, y_cv.shape)
print(X_test_project_grade_category_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print('*'*100)
```

```
(16642, 4) (16642,)
```

```
(4831, 4) (4831,)
(7211, 4) (7211,)
['grades 3 5', 'grades 6 8', 'grades 9 12', 'grades prek 2']
**************************************************************************************************
```

In [40]:

```
# One hot encoding of Categorical Feature
# - teacher_prefix : categorical data
vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values)# Fit has to happen only on train data

X_train_teacher_prefix_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
X_cv_clean_teacher_prefix_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_clean_teacher_prefix_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

print(X_train_teacher_prefix_ohe.shape, y_train.shape)
print(X_cv_clean_teacher_prefix_ohe.shape, y_cv.shape)
print(X_test_clean_teacher_prefix_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print('*'*100)
```

```
(16642, 4) (16642,)
(4831, 4) (4831,)
(7211, 4) (7211,)
['mr', 'mrs', 'ms', 'teacher']
**************************************************************************************************
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [41]:

```
# - project_title : text data
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("*"*100)

# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
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_title_bow = vectorizer.transform(X_train['clean_titles'].values)
X_cv_title_bow = vectorizer.transform(X_cv['clean_titles'].values)
X_test_title_bow = vectorizer.transform(X_test['clean_titles'].values)

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

```
(16642, 17) (16642,)
(4831, 17) (4831,)
(7211, 17) (7211,)
**************************************************************************************************

After vectorizations
(16642, 1808) (16642,)
(4831, 1808) (4831,)
(7211, 1808) (7211,)
**************************************************************************************************
```

In [42]:

```
# - text : text data
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("*"*100)

# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
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)
```

```
(16642, 17) (16642,)
(4831, 17) (4831,)
(7211, 17) (7211,)
****************************************************************************************************

After vectorizations
(16642, 5000) (16642,)
(4831, 5000) (4831,)
(7211, 5000) (7211,)
****************************************************************************************************
```

In [43]:

```
# - project_resource_summary: text data (optinal)
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("*"*100)

# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['project_resource_summary'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_project_resource_summary_bow = vectorizer.transform(X_train['project_resource_summary'].va
lues)
X_cv_project_resource_summary_bow = vectorizer.transform(X_cv['project_resource_summary'].values)
X_test_project_resource_summary_bow =
vectorizer.transform(X_test['project_resource_summary'].values)

print("After vectorizations")
print(X_train_project_resource_summary_bow.shape, y_train.shape)
print(X_cv_project_resource_summary_bow.shape, y_cv.shape)
print(X_test_project_resource_summary_bow.shape, y_test.shape)
print("*"*100)
```

```
(16642, 17) (16642,)
(4831, 17) (4831,)
(7211, 17) (7211,)
****************************************************************************************************

After vectorizations
(16642, 5000) (16642,)
(4831, 5000) (4831,)
(7211, 5000) (7211,)
****************************************************************************************************
```

1.5.2.2 TFIDF vectorizer

In [44]:

```
# - project_title : text data
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("*"*100)

from sklearn.feature_extraction.text import TfidfVectorizer
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = TfidfVectorizer(min_df=10)
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_title_tfidf = vectorizer.transform(X_train['clean_titles'].values)
X_cv_title_tfidf = vectorizer.transform(X_cv['clean_titles'].values)
X_test_title_tfidf = vectorizer.transform(X_test['clean_titles'].values)

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

```
(16642, 17) (16642,)
(4831, 17) (4831,)
(7211, 17) (7211,)
****************************************************************************************************

After vectorizations
(16642, 1014) (16642,)
(4831, 1014) (4831,)
(4831, 1014) (7211,)
****************************************************************************************************
```

In [45]:

```
# - text : text data
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("*"*100)

from sklearn.feature_extraction.text import TfidfVectorizer
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = TfidfVectorizer(min_df=10)
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)

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

```
(16642, 17) (16642,)
(4831, 17) (4831,)
(7211, 17) (7211,)
****************************************************************************************************

After vectorizations
(16642, 8243) (16642,)
(4831, 8243) (4831,)
(7211, 8243) (7211,)
```

```
**************************************************************************************************************
```

In [46]:

```
# - project_resource_summary: text data (optinal)
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("*"*100)

from sklearn.feature_extraction.text import TfidfVectorizer
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['project_resource_summary'].values) # fit has to happen only on train data

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

print("After vectorizations")
print(X_train_project_resource_summary_tfidf.shape, y_train.shape)
print(X_cv_project_resource_summary_tfidf.shape, y_cv.shape)
print(X_test_project_resource_summary_tfidf.shape, y_test.shape)
print("*"*100)
```

```
(16642, 17) (16642,)
(4831, 17) (4831,)
(7211, 17) (7211,)
**************************************************************************************************************

After vectorizations
(16642, 2333) (16642,)
(4831, 2333) (4831,)
(7211, 2333) (7211,)
**************************************************************************************************************
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [47]:

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

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

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

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

words = []
for i in preproced_texts:
```

```
    words.extend(i.split(' '))

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

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

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


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

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


'''
```

Out[47]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef
loadGloveModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFile,\'r\',
encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n        splitLine = line.split()\n
word = splitLine[0]\n        embedding = np.array([float(val) for val in splitLine[1:]])\n        m
odel[word] = embedding\n    print ("Done.",len(model)," words loaded!")\n    return model\nmodel =
loadGloveModel(\'glove.42B.300d.txt\')\n\n# ============================\nOutput:\n    \nLoading G
love Model\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n#
============================\n\nwords = []\nfor i in preproced_texts:\n    words.extend(i.split(\'
\'))\n\nfor i in preproced_titles:\n    words.extend(i.split(\' \'))\nprint("all the words in the
coupus", len(words))\nwords = set(words)\nprint("the unique words in the coupus",
len(words))\n\ninter_words = set(model.keys()).intersection(words)\nprint("The number of words tha
t are present in both glove vectors and our coupus",       len(inter_words),"
(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nwords_glove =
set(model.keys())\nfor i in words:\n    if i in words_glove:\n        words_courpus[i] = model[i]\n
print("word 2 vec length", len(words_courpus))\n\n\n# stronging variables into pickle files python
: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimport pic
kle\nwith open(\'glove_vectors\', \'wb\') as f:\n    pickle.dump(words_courpus, f)\n\n\n'
```

In [48]:

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

In [49]:

```
# average Word2Vec for train text
# compute average word2vec for each review.
avg_w2v_vectors_text_train = []; # 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(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_text_train.append(vector)
```

```
print(len(avg_w2v_vectors_text_train))
print(len(avg_w2v_vectors_text_train[0]))
```

```
100%|██████████| 16642/16642 [00:07<00:00, 2305.35it/s]
```

```
16642
300
```

In [50]:

```
# average Word2Vec for CV text
# compute average word2vec for each review.
avg_w2v_vectors_text_cv = []; # 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(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_text_cv.append(vector)

print(len(avg_w2v_vectors_text_cv))
print(len(avg_w2v_vectors_text_cv[0]))
```

```
100%|██████████| 4831/4831 [00:02<00:00, 1836.91it/s]
```

```
4831
300
```

In [51]:

```
# average Word2Vec for test text
# compute average word2vec for each review.
avg_w2v_vectors_text_test = []; # 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(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_text_test.append(vector)

print(len(avg_w2v_vectors_text_test))
print(len(avg_w2v_vectors_text_test[0]))
```

```
100%|██████████| 7211/7211 [00:03<00:00, 2155.02it/s]
```

```
7211
300
```

In [52]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_title_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
```

```
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_title_train.append(vector)

print(len(avg_w2v_vectors_title_train))
print(len(avg_w2v_vectors_title_train[0]))
```

```
100%|██████████| 16642/16642 [00:00<00:00, 51025.13it/s]
```

```
16642
300
```

In [53]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_title_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_title_cv.append(vector)

print(len(avg_w2v_vectors_title_cv))
print(len(avg_w2v_vectors_title_cv[0]))
```

```
100%|██████████| 4831/4831 [00:00<00:00, 37458.35it/s]
```

```
4831
300
```

In [54]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_title_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_title_test.append(vector)

print(len(avg_w2v_vectors_title_test))
print(len(avg_w2v_vectors_title_test[0]))
```

```
100%|██████████| 7211/7211 [00:00<00:00, 30858.69it/s]
```

```
7211
300
```

In [55]:

```
# Similarly you can vectorize for project_resource_summary also
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_project_resource_summary_train = []; # the avg-w2v for each sentence/review is sto
red in this list
for sentence in tqdm(X_train['project_resource_summary']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_project_resource_summary_train.append(vector)

print(len(avg_w2v_vectors_project_resource_summary_train))
print(len(avg_w2v_vectors_project_resource_summary_train[0]))
```

```
100%|██████████| 16642/16642 [00:01<00:00, 10891.03it/s]
```

```
16642
300
```

In [56]:

```
# Similarly you can vectorize for project_resource_summary also
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_project_resource_summary_cv = []; # the avg-w2v for each sentence/review is stored
in this list
for sentence in tqdm(X_cv['project_resource_summary']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_project_resource_summary_cv.append(vector)

print(len(avg_w2v_vectors_project_resource_summary_cv))
print(len(avg_w2v_vectors_project_resource_summary_cv[0]))
```

```
100%|██████████| 4831/4831 [00:00<00:00, 18333.49it/s]
```

```
4831
300
```

In [57]:

```
# Similarly you can vectorize for project_resource_summary also
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_project_resource_summary_test = []; # the avg-w2v for each sentence/review is stor
ed in this list
for sentence in tqdm(X_test['project_resource_summary']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
```

```
    avg_w2v_vectors_project_resource_summary_test.append(vector)

print(len(avg_w2v_vectors_project_resource_summary_test))
print(len(avg_w2v_vectors_project_resource_summary_test[0]))
```

```
100%|██████████| 7211/7211 [00:00<00:00, 19095.56it/s]
```

```
7211
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [58]:

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

In [59]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_text_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_text_train.append(vector)

print(len(tfidf_w2v_vectors_text_train))
print(len(tfidf_w2v_vectors_text_train[0]))
```

```
100%|██████████| 16642/16642 [01:04<00:00, 256.80it/s]
```

```
16642
300
```

In [60]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_text_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
```

```
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_text_cv.append(vector)

print(len(tfidf_w2v_vectors_text_cv))
print(len(tfidf_w2v_vectors_text_cv[0]))
```

```
100%|██████████| 4831/4831 [00:22<00:00, 217.34it/s]
```

```
4831
300
```

In [61]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_text_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_text_test.append(vector)

print(len(tfidf_w2v_vectors_text_test))
print(len(tfidf_w2v_vectors_text_test[0]))
```

```
100%|██████████| 7211/7211 [00:43<00:00, 167.59it/s]
```

```
7211
300
```

In [62]:

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

In [63]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_title_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
```

```
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_title_train.append(vector)

print(len(tfidf_w2v_vectors_title_train))
print(len(tfidf_w2v_vectors_title_train[0]))
```

```
100%|██████████| 16642/16642 [00:00<00:00, 26602.35it/s]
```

```
16642
300
```

In [64]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_title_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_title_cv.append(vector)

print(len(tfidf_w2v_vectors_title_cv))
print(len(tfidf_w2v_vectors_title_cv[0]))
```

```
100%|██████████| 4831/4831 [00:00<00:00, 27672.07it/s]
```

```
4831
300
```

In [65]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_title_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_title_test.append(vector)

print(len(tfidf_w2v_vectors_title_test))
print(len(tfidf_w2v_vectors_title_test[0]))
```

```
100%|██████████| 7211/7211 [00:00<00:00, 22475.29it/s]
```

```
7211
300
```

In [66]:

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

In [67]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_project_resource_summary_train = []; # the avg-w2v for each sentence/review is s
tored in this list
for sentence in tqdm(X_train['project_resource_summary']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_project_resource_summary_train.append(vector)

print(len(tfidf_w2v_vectors_project_resource_summary_train))
print(len(tfidf_w2v_vectors_project_resource_summary_train[0]))
```

```
100%|██████████| 16642/16642 [00:02<00:00, 6343.99it/s]
```

```
16642
300
```

In [68]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_project_resource_summary_cv = []; # the avg-w2v for each sentence/review is stor
ed in this list
for sentence in tqdm(X_cv['project_resource_summary']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_project_resource_summary_cv.append(vector)

print(len(tfidf_w2v_vectors_project_resource_summary_cv))
print(len(tfidf_w2v_vectors_project_resource_summary_cv[0]))
```

```
100%|██████████| 4831/4831 [00:00<00:00, 7376.34it/s]
```

```
100%|██████████| 4831/4831 [00:00<00:00, 7576.34it/s]
```

```
4831
300
```

In [69]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_project_resource_summary_test = []; # the avg-w2v for each sentence/review is st
ored in this list
for sentence in tqdm(X_test['project_resource_summary']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_project_resource_summary_test.append(vector)

print(len(tfidf_w2v_vectors_project_resource_summary_test))
print(len(tfidf_w2v_vectors_project_resource_summary_test[0]))
```

```
100%|██████████| 7211/7211 [00:01<00:00, 3701.95it/s]
```

```
7211
300
```

1.5.3 Vectorizing Numerical features

In [70]:

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

X_train_quantity_norm = normalizer.transform(X_train['quantity'].values.reshape(-1,1))
X_cv_quantity_norm = normalizer.transform(X_cv['quantity'].values.reshape(-1,1))
X_test_quantity_norm = normalizer.transform(X_test['quantity'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_quantity_norm.shape, y_train.shape)
print(X_cv_quantity_norm.shape, y_cv.shape)
print(X_test_quantity_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(16642, 1) (16642,)
(4831, 1) (4831,)
(7211, 1) (7211,)
====================================================================================================
```

In [71]:

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

X_train_teacher_number_of_previously_posted_projects_norm =
normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_cv_teacher_number_of_previously_posted_projects_norm =
normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_test_teacher_number_of_previously_posted_projects_norm =
normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_teacher_number_of_previously_posted_projects_norm.shape, y_train.shape)
print(X_cv_teacher_number_of_previously_posted_projects_norm.shape, y_cv.shape)
print(X_test_teacher_number_of_previously_posted_projects_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(16642, 1) (16642,)
(4831, 1) (4831,)
(7211, 1) (7211,)
====================================================================================================
```

In [72]:

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

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

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

```
After vectorizations
(16642, 1) (16642,)
(4831, 1) (4831,)
(7211, 1) (7211,)
====================================================================================================
```

1.5.4 Merging all the above features

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

In [73]:

```
# print(categories_one_hot.shape)
# print(sub_categories_one_hot.shape)
# print(text_bow.shape)
# print(price_standardized.shape)
print('Categorical Features')
print('*'*100)
print(X_train_school_state_ohe.shape, y_train.shape)
print(X_cv_school_state_ohe.shape, y_cv.shape)
print(X_test_school_state_ohe.shape, y_test.shape)
print('*'*100)
print(X_train_clean_categories_ohe.shape, y_train.shape)
print(X_cv_clean_categories_ohe.shape, y_cv.shape)
print(X_test_clean_categories_ohe.shape, y_test.shape)
print('*'*100)
print(X_train_clean_subcategories_ohe.shape, y_train.shape)
print(X_cv_clean_subcategories_ohe.shape, y_cv.shape)
print(X_test_clean_subcategories_ohe.shape, y_test.shape)
print('*'*100)
print(X_train_project_grade_category_ohe.shape, y_train.shape)
print(X_cv_project_grade_category_ohe.shape, y_cv.shape)
print(X_test_project_grade_category_ohe.shape, y_test.shape)
print('*'*100)
print(X_train_teacher_prefix_ohe.shape, y_train.shape)
print(X_cv_clean_teacher_prefix_ohe.shape, y_cv.shape)
print(X_test_clean_teacher_prefix_ohe.shape, y_test.shape)
print('*'*100)
print('Text Encoding Features')
print('*'*100)
print(X_train_title_bow.shape, y_train.shape)
print(X_cv_title_bow.shape, y_cv.shape)
print(X_test_title_bow.shape, y_test.shape)
print('*'*100)
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)
print(X_train_project_resource_summary_bow.shape, y_train.shape)
print(X_cv_project_resource_summary_bow.shape, y_cv.shape)
print(X_test_project_resource_summary_bow.shape, y_test.shape)
print('*'*100)
print(X_train_title_tfidf.shape, y_train.shape)
print(X_cv_title_tfidf.shape, y_cv.shape)
print(X_cv_title_tfidf.shape, y_test.shape)
print('*'*100)
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print('*'*100)
print(X_train_project_resource_summary_tfidf.shape, y_train.shape)
print(X_cv_project_resource_summary_tfidf.shape, y_cv.shape)
print(X_test_project_resource_summary_tfidf.shape, y_test.shape)
print('*'*100)
print(len(avg_w2v_vectors_text_train))
print(len(avg_w2v_vectors_text_train[0]))
print('*'*100)
print(len(avg_w2v_vectors_text_cv))
print(len(avg_w2v_vectors_text_cv[0]))
print('*'*100)
print(len(avg_w2v_vectors_text_test))
print(len(avg_w2v_vectors_text_test[0]))
print('*'*100)
print(len(avg_w2v_vectors_title_train))
print(len(avg_w2v_vectors_title_train[0]))
print('*'*100)
print(len(avg_w2v_vectors_title_cv))
print(len(avg_w2v_vectors_title_cv[0]))
print('*'*100)
print(len(avg_w2v_vectors_title_test))
print(len(avg_w2v_vectors_title_test[0]))
print('*'*100)
print(len(avg_w2v_vectors_project_resource_summary_train))
print(len(avg_w2v_vectors_project_resource_summary_train[0]))
print('*'*100)
print(len(avg_w2v_vectors_project_resource_summary_cv))
print(len(avg_w2v_vectors_project_resource_summary_cv[0]))
print('*'*100)
```

```
print(len(avg_w2v_vectors_project_resource_summary_test))
print(len(avg_w2v_vectors_project_resource_summary_test[0]))
print('*'*100)
print(len(tfidf_w2v_vectors_text_train))
print(len(tfidf_w2v_vectors_text_train[0]))
print('*'*100)
print(len(tfidf_w2v_vectors_text_cv))
print(len(tfidf_w2v_vectors_text_cv[0]))
print('*'*100)
print(len(tfidf_w2v_vectors_text_test))
print(len(tfidf_w2v_vectors_text_test[0]))
print('*'*100)
print(len(tfidf_w2v_vectors_title_train))
print(len(tfidf_w2v_vectors_title_train[0]))
print('*'*100)
print(len(tfidf_w2v_vectors_title_cv))
print(len(tfidf_w2v_vectors_title_cv[0]))
print('*'*100)
print(len(tfidf_w2v_vectors_title_test))
print(len(tfidf_w2v_vectors_title_test[0]))
print('*'*100)
print(len(tfidf_w2v_vectors_project_resource_summary_train))
print(len(tfidf_w2v_vectors_project_resource_summary_train[0]))
print('*'*100)
print(len(tfidf_w2v_vectors_project_resource_summary_cv))
print(len(tfidf_w2v_vectors_project_resource_summary_cv[0]))
print('*'*100)
print(len(tfidf_w2v_vectors_project_resource_summary_test))
print(len(tfidf_w2v_vectors_project_resource_summary_test[0]))
print('*'*100)
print('Numerical Features')
print('*'*100)
print(X_train_quantity_norm.shape, y_train.shape)
print(X_cv_quantity_norm.shape, y_cv.shape)
print(X_test_quantity_norm.shape, y_test.shape)
print('*'*100)
print(X_train_teacher_number_of_previously_posted_projects_norm.shape, y_train.shape)
print(X_cv_teacher_number_of_previously_posted_projects_norm.shape, y_cv.shape)
print(X_test_teacher_number_of_previously_posted_projects_norm.shape, y_test.shape)
print('*'*100)
print(X_train_price_norm.shape, y_train.shape)
print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
```

```
Categorical Features
****************************************************************************************************

(16642, 51) (16642,)
(4831, 51) (4831,)
(7211, 51) (7211,)
****************************************************************************************************

(16642, 9) (16642,)
(4831, 9) (4831,)
(7211, 9) (7211,)
****************************************************************************************************

(16642, 30) (16642,)
(4831, 30) (4831,)
(7211, 30) (7211,)
****************************************************************************************************

(16642, 4) (16642,)
(4831, 4) (4831,)
(7211, 4) (7211,)
****************************************************************************************************

(16642, 4) (16642,)
(4831, 4) (4831,)
(7211, 4) (7211,)
****************************************************************************************************

Text Encoding Features
****************************************************************************************************

(16642, 1808) (16642,)
(4831, 1808) (4831,)
```

(4831, 1808) (4831,)
(7211, 1808) (7211,)
****

(16642, 5000) (16642,)
(4831, 5000) (4831,)
(7211, 5000) (7211,)
****

(16642, 5000) (16642,)
(4831, 5000) (4831,)
(7211, 5000) (7211,)
****

(16642, 1014) (16642,)
(4831, 1014) (4831,)
(4831, 1014) (7211,)
****

(16642, 8243) (16642,)
(4831, 8243) (4831,)
(7211, 8243) (7211,)
****

(16642, 2333) (16642,)
(4831, 2333) (4831,)
(7211, 2333) (7211,)
****

16642
300
****

4831
300
****

7211
300
****

16642
300
****

4831
300
****

7211
300
****

16642
300
****

4831
300
****

7211
300
****

16642
300
****

4831
300
****

7211
300
****

16642

```
16642
300
**********************************************************************************************************

4831
300
**********************************************************************************************************

7211
300
**********************************************************************************************************

16642
300
**********************************************************************************************************

4831
300
**********************************************************************************************************

7211
300
**********************************************************************************************************

Numerical Features
**********************************************************************************************************

(16642, 1) (16642,)
(4831, 1) (4831,)
(7211, 1) (7211,)
**********************************************************************************************************

(16642, 1) (16642,)
(4831, 1) (4831,)
(7211, 1) (7211,)
**********************************************************************************************************

(16642, 1) (16642,)
(4831, 1) (4831,)
(7211, 1) (7211,)
```

In [74]:

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

X_train_real = X_train
X_cv_real = X_cv
X_test_real = X_test

X_train = hstack((X_train_school_state_ohe, X_train_clean_categories_ohe,
X_train_clean_subcategories_ohe, X_train_project_grade_category_ohe, X_train_teacher_prefix_ohe, X
_train_title_bow, X_train_essay_bow, X_train_project_resource_summary_bow, X_train_title_tfidf,
X_train_essay_tfidf, X_train_project_resource_summary_tfidf, avg_w2v_vectors_text_train,
avg_w2v_vectors_title_train, avg_w2v_vectors_project_resource_summary_train,
tfidf_w2v_vectors_text_train, tfidf_w2v_vectors_title_train,
tfidf_w2v_vectors_project_resource_summary_train, X_train_quantity_norm,
X_train_teacher_number_of_previously_posted_projects_norm, X_train_price_norm)).tocsr()
X_cv = hstack((X_cv_school_state_ohe, X_cv_clean_categories_ohe, X_cv_clean_subcategories_ohe,
X_cv_project_grade_category_ohe, X_cv_clean_teacher_prefix_ohe, X_cv_title_bow, X_cv_essay_bow, X_c
v_project_resource_summary_bow, X_cv_title_tfidf, X_cv_essay_tfidf,
X_cv_project_resource_summary_tfidf, avg_w2v_vectors_text_cv, avg_w2v_vectors_title_cv,
avg_w2v_vectors_project_resource_summary_cv, tfidf_w2v_vectors_text_cv, tfidf_w2v_vectors_title_cv
, tfidf_w2v_vectors_project_resource_summary_cv, X_cv_quantity_norm,
X_cv_teacher_number_of_previously_posted_projects_norm, X_cv_price_norm)).tocsr()
X_test = hstack((X_test_school_state_ohe, X_test_clean_categories_ohe,
X_test_clean_subcategories_ohe, X_test_project_grade_category_ohe, X_test_clean_teacher_prefix_ohe
,X_test_title_bow, X_test_essay_bow, X_test_project_resource_summary_bow, X_test_title_tfidf,
X_test_essay_tfidf, X_test_project_resource_summary_tfidf, avg_w2v_vectors_text_test,
avg_w2v_vectors_title_test, avg_w2v_vectors_project_resource_summary_test,
tfidf_w2v_vectors_text_test, tfidf_w2v_vectors_title_test,
tfidf_w2v_vectors_project_resource_summary_test, X_test_quantity_norm,
X_test_teacher_number_of_previously_posted_projects_norm, X_test_price_norm)).tocsr()
```

```
print(X_train_real.shape)
print(X_cv_real.shape)
print(X_test_real.shape)
print(X_train.shape)
print(X_cv.shape)
print(X_test.shape)
```

```
(16642, 17)
(4831, 17)
(7211, 17)
(16642, 25299)
(4831, 25299)
(7211, 25299)
```

Assignment 3: Apply KNN

1. **[Task-1] Apply KNN(brute force version) on these feature sets**
   - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)
   - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF)
   - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_essay (AVG W2V)
   - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)
2. **Hyper paramter tuning to find best K**
   - Find the best hyper parameter which results in the maximum AUC value
   - Find the best hyper paramter using k-fold cross validation (or) simple cross validation data
   - Use gridsearch-cv or randomsearch-cv or write your own for loops to do this task
3. **Representation of results**
   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, as shown in the figure
   - Once you find the best hyper parameter, you need to train your model-M using the best hyper-param. Now, find the AUC on test data and plot the ROC curve on both train and test using model-M.
   - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points
4. **[Task-2]**
   - Select top 2000 features from feature Set 2 using `SelectKBest` and then apply KNN on top of these features
     - ```
       from sklearn.datasets import load_digits
       from sklearn.feature_selection import SelectKBest, chi2
       X, y = load_digits(return_X_y=True)
       X.shape
       X_new = SelectKBest(chi2, k=20).fit_transform(X, y)
       X_new.shape
       ========
       output:
       (1797, 64)
       (1797, 20)
       ```
   - Repeat the steps 2 and 3 on the data matrix after feature selection
5. **Conclusion**
   - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
       ```

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

2. K Nearest Neighbor

Note: I already completed steps 2.1, 2.2 & 2.3 previously, So I didn't copy code in below cells.

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

In [75]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [76]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# make sure you featurize train and test data separatly

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

2.3 Make Data Model Ready: encoding eassay, and project_title

In [77]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# make sure you featurize train and test data separatly

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

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

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

In [78]:

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

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

In [79]:

```
#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')
```

In [80]:

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

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

    return y_data_pred
```

2.4.1 Applying KNN brute force on BOW, SET 1

In [88]:

```
%%time
# Please write all the code with proper documentation

# Prepare data for BOW
X_train_bow = hstack((X_train_school_state_ohe, X_train_clean_categories_ohe,
X_train_clean_subcategories_ohe, X_train_project_grade_category_ohe, X_train_teacher_prefix_ohe, X
_train_title_bow, X_train_essay_bow, X_train_project_resource_summary_bow, X_train_quantity_norm,
X_train_teacher_number_of_previously_posted_projects_norm, X_train_price_norm)).tocsr()
X_cv_bow = hstack((X_cv_school_state_ohe, X_cv_clean_categories_ohe, X_cv_clean_subcategories_ohe,
X_cv_project_grade_category_ohe, X_cv_clean_teacher_prefix_ohe, X_cv_title_bow, X_cv_essay_bow, X_c
v_project_resource_summary_bow, X_cv_quantity_norm,
X_cv_teacher_number_of_previously_posted_projects_norm, X_cv_price_norm)).tocsr()
X_test_bow = hstack((X_test_school_state_ohe, X_test_clean_categories_ohe,
X_test_clean_subcategories_ohe, X_test_project_grade_category_ohe, X_test_clean_teacher_prefix_ohe
,X_test_title_bow, X_test_essay_bow, X_test_project_resource_summary_bow, X_test_quantity_norm,
X_test_teacher_number_of_previously_posted_projects_norm, X_test_price_norm)).tocsr()

print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)

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

```
y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [1, 5, 11, 15, 31, 41, 51,71,101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X_train_bow, y_train)

    y_train_pred = batch_predict(neigh, X_train_bow)
    y_cv_pred = batch_predict(neigh, X_cv_bow)

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

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

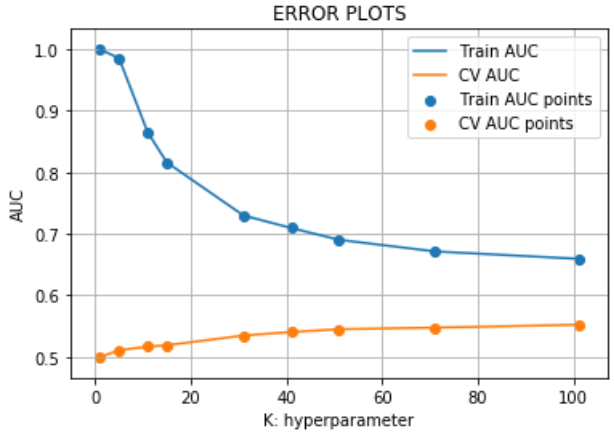
plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

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

```
  0%|          | 0/9 [00:00<?, ?it/s]
```

```
(41692, 14453) (41692,)
(12078, 14453) (12078,)
(18026, 14453) (18026,)
```

```
100%|██████████| 9/9 [54:51<00:00, 351.88s/it]
```




```
CPU times: user 2h 5min 8s, sys: 8min 52s, total: 2h 14min 1s
Wall time: 54min 53s
```

In [233]:

```
import pickle



pickle_out = open("clf1.pickle","wb")
pickle.dump(clf1, pickle_out)
```

```
pickle_out.close()
```

In [99]:

```
# from the error plot we choose K such that, we will have maximum AUC on cv data and gap between t
he train and cv is less
# Note: based on the method you use you might get different hyperparameter values as best one
# so, you choose according to the method you choose, you use gridsearch if you are having more com
puting power and note it will take more time
# if you increase the cv values in the GridSearchCV you will get more rebust results.

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

In [104]:

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


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

y_train_pred = batch_predict(neigh, X_train_bow)
y_test_pred = batch_predict(neigh, X_test_bow)

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

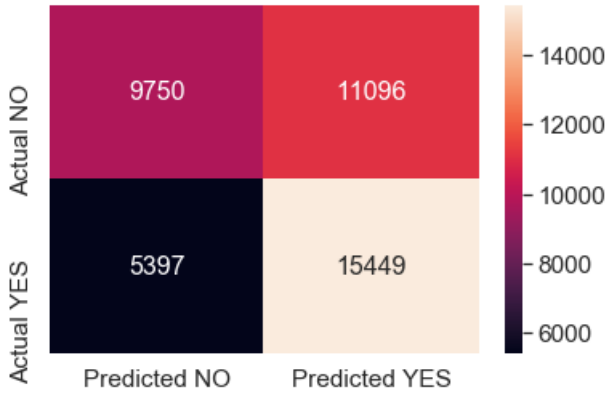
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE (FPR)")
plt.ylabel("TRUE POSITIVE RATE (TPR)")
plt.title("ROC PLOTS")
plt.grid()
plt.show()
```


```
CPU times: user 14min 10s, sys: 1min 4s, total: 15min 15s
Wall time: 6min 20s
```

In [101]:

```
%%time
get_confusion_matrix(neigh,X_train_bow,y_train)
```

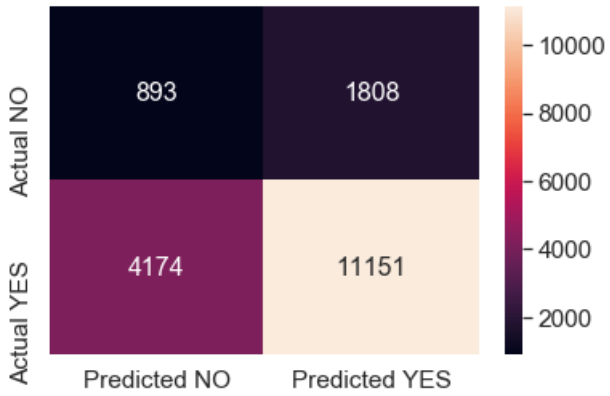
```
CPU times: user 10min 35s, sys: 41.7 s, total: 11min 17s
Wall time: 4min 21s
```



In [102]:

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

```
CPU times: user 4min 8s, sys: 16.3 s, total: 4min 24s
Wall time: 1min 39s
```



2.4.2 Applying KNN brute force on TFIDF, SET 2

In [103]:

```
%%time
# Please write all the code with proper documentation
# Prepare data for TFIDF
X_train_tfidf = hstack((X_train_school_state_ohe, X_train_clean_categories_ohe,
X_train_clean_subcategories_ohe, X_train_project_grade_category_ohe, X_train_teacher_prefix_ohe, X
_train_title_tfidf, X_train_essay_tfidf, X_train_project_resource_summary_tfidf,
X_train_quantity_norm, X_train_teacher_number_of_previously_posted_projects_norm,
X_train_price_norm)).tocsr()
X_cv_tfidf = hstack((X_cv_school_state_ohe, X_cv_clean_categories_ohe,
X_cv_clean_subcategories_ohe, X_cv_project_grade_category_ohe, X_cv_clean_teacher_prefix_ohe,
X_cv_title_tfidf, X_cv_essay_tfidf, X_cv_project_resource_summary_tfidf, X_cv_quantity_norm,
X_cv_teacher_number_of_previously_posted_projects_norm, X_cv_price_norm)).tocsr()
X_test_tfidf = hstack((X_test_school_state_ohe, X_test_clean_categories_ohe,
X_test_clean_subcategories_ohe, X_test_project_grade_category_ohe, X_test_clean_teacher_prefix_ohe
, X_test_title_tfidf, X_test_essay_tfidf, X_test_project_resource_summary_tfidf,
X_test_quantity_norm, X_test_teacher_number_of_previously_posted_projects_norm, X_test_price_norm)
).tocsr()

print(X_train_tfidf.shape)
print(X_cv_tfidf.shape)
print(X_test_tfidf.shape)

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
```

```
true binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [1, 5, 11, 15, 31, 41, 51,71,101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X_train_tfidf, y_train)

    y_train_pred = batch_predict(neigh, X_train_tfidf)
    y_cv_pred = batch_predict(neigh, X_cv_tfidf)

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

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

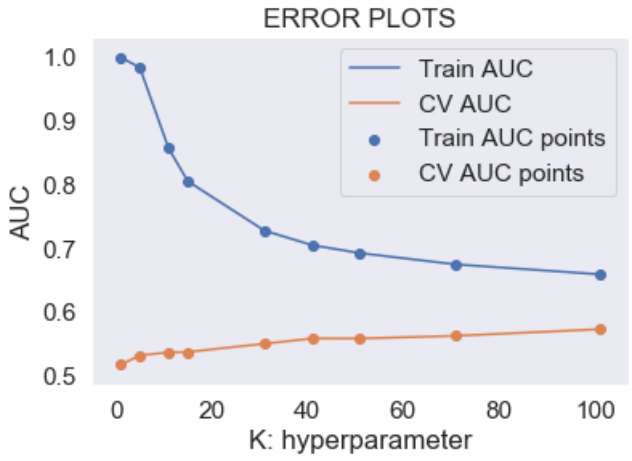
plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

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

```
  0%|          | 0/9 [00:00<?, ?it/s]
```

```
(41692, 17913)
(12078, 17913)
(18026, 17913)
```

```
 11%|█         | 1/9 [03:27<27:36, 207.05s/it]
 22%|██        | 2/9 [06:57<24:15, 207.97s/it]
 33%|███       | 3/9 [10:49<21:31, 215.22s/it]
 44%|████      | 4/9 [14:50<18:35, 223.02s/it]
 56%|█████     | 5/9 [19:05<15:30, 232.57s/it]
 67%|██████    | 6/9 [23:41<12:16, 245.55s/it]
 78%|███████   | 7/9 [27:49<08:12, 246.34s/it]
 89%|████████  | 8/9 [32:50<04:22, 262.85s/it]
100%|██████████| 9/9 [36:52<00:00, 256.42s/it]
```




```
CPU times: user 1h 21min 58s, sys: 6min 55s, total: 1h 28min 53s
```

```
Wall time: 36min 53s
```

In [105]:

```
# from the error plot we choose K such that, we will have maximum AUC on cv data and gap between t
he train and cv is less
# Note: based on the method you use you might get different hyperparameter values as best one
# so, you choose according to the method you choose, you use gridsearch if you are having more com
puting power and note it will take more time
# if you increase the cv values in the GridSearchCV you will get more rebust results.

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

In [110]:

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


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

y_train_pred = batch_predict(neigh, X_train_tfidf)
y_test_pred = batch_predict(neigh, X_test_tfidf)

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

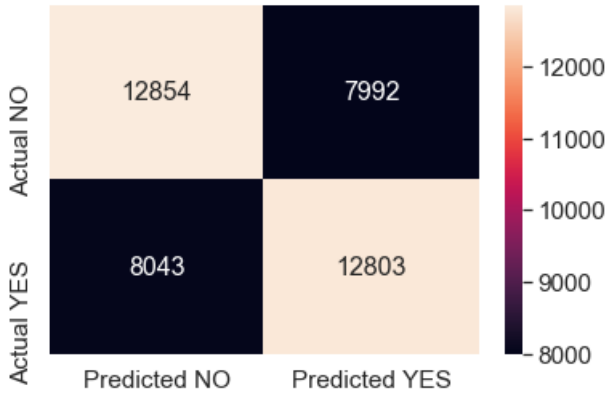
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE (FPR)")
plt.ylabel("TRUE POSITIVE RATE (TPR)")
plt.title("ROC PLOTS")
plt.grid()
plt.show()
```


```
CPU times: user 13min 39s, sys: 59.9 s, total: 14min 39s
Wall time: 6min 9s
```

In [111]:

```
%%time
get_confusion_matrix(neigh,X_train_tfidf,y_train)
```

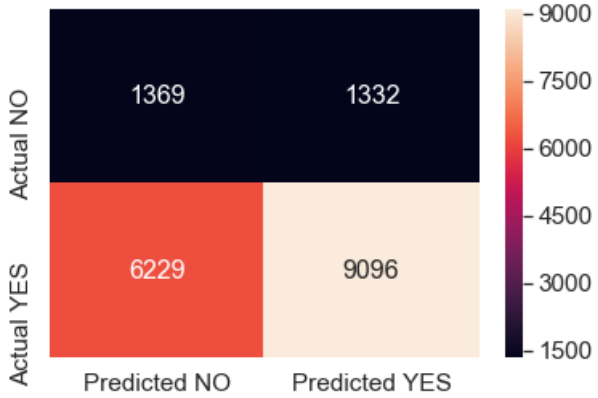
```
CPU times: user 8min 9s, sys: 37.1 s, total: 8min 46s
Wall time: 3min 16s
```



In [112]:

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

```
CPU times: user 3min 23s, sys: 15.2 s, total: 3min 39s
Wall time: 1min 17s
```



2.4.3 Applying KNN brute force on AVG W2V, SET 3

In [234]:

```
%%time
# I reduced data points to perform AVG W2V for 10K data points, For large data points it was not c
ompleted within 12hrs.
# Please write all the code with proper documentation
# Prepare data for BOW
X_train_avgw2v = hstack((X_train_school_state_ohe, X_train_clean_categories_ohe,
X_train_clean_subcategories_ohe, X_train_project_grade_category_ohe, X_train_teacher_prefix_ohe, a
vg_w2v_vectors_text_train, avg_w2v_vectors_title_train,
avg_w2v_vectors_project_resource_summary_train, X_train_quantity_norm,
X_train_teacher_number_of_previously_posted_projects_norm, X_train_price_norm)).tocsr()
X_cv_avgw2v = hstack((X_cv_school_state_ohe, X_cv_clean_categories_ohe,
X_cv_clean_subcategories_ohe, X_cv_project_grade_category_ohe, X_cv_clean_teacher_prefix_ohe,
avg_w2v_vectors_text_cv, avg_w2v_vectors_title_cv, avg_w2v_vectors_project_resource_summary_cv, X_
cv_quantity_norm, X_cv_teacher_number_of_previously_posted_projects_norm, X_cv_price_norm)).tocsr(
)
X_test_avgw2v = hstack((X_test_school_state_ohe, X_test_clean_categories_ohe,
X_test_clean_subcategories_ohe, X_test_project_grade_category_ohe, X_test_clean_teacher_prefix_ohe
, avg_w2v_vectors_text_test, avg_w2v_vectors_title_test,
avg_w2v_vectors_project_resource_summary_test, X_test_quantity_norm,
X_test_teacher_number_of_previously_posted_projects_norm, X_test_price_norm)).tocsr()


print(X_train_avgw2v.shape)
print(X_cv_avgw2v.shape)
print(X_test_avgw2v.shape)

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

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

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [1, 5, 11, 15, 31, 41, 51,71,101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X_train_avgw2v, y_train)

    y_train_pred = batch_predict(neigh, X_train_avgw2v)
    y_cv_pred = batch_predict(neigh, X_cv_avgw2v)

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

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

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

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

```
  0%|          | 0/9 [00:00<?, ?it/s]
```

```
(16634, 1001)
(4831, 1001)
(7211, 1001)
```

```
 11%|▉         | 1/9 [15:15<2:02:03, 915.44s/it]

 22%|██        | 2/9 [43:57<2:15:01, 1157.29s/it]

 33%|███       | 3/9 [1:06:08<2:00:56, 1209.44s/it]

 44%|████      | 4/9 [1:33:04<1:50:57, 1331.50s/it]

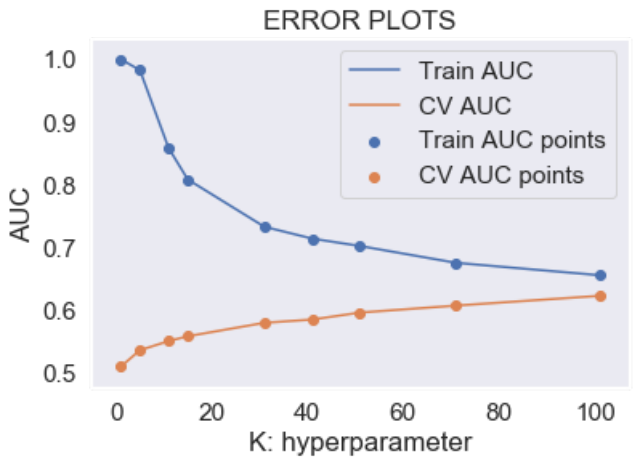
 56%|█████     | 5/9 [1:57:29<1:31:26, 1371.62s/it]

 67%|██████    | 6/9 [2:16:25<1:05:02, 1300.94s/it]

 78%|███████   | 7/9 [2:28:54<37:50, 1135.22s/it]

 89%|████████  | 8/9 [2:37:49<15:55, 955.34s/it]
```

```
100%|██████████| 9/9 [2:50:10<00:00, 890.94s/it]
```




```
CPU times: user 7h 35s, sys: 6min 8s, total: 7h 6min 44s
Wall time: 2h 50min 13s
```

In [235]:

```
# from the error plot we choose K such that, we will have maximum AUC on cv data and gap between t
he train and cv is less
# Note: based on the method you use you might get different hyperparameter values as best one
# so, you choose according to the method you choose, you use gridsearch if you are having more com
puting power and note it will take more time
# if you increase the cv values in the GridSearchCV you will get more rebust results.

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

In [236]:

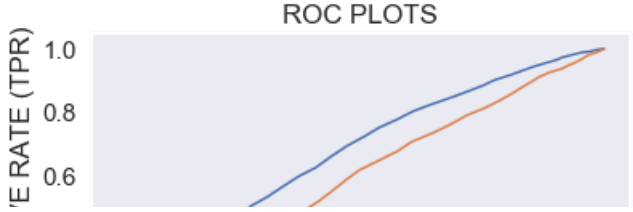
```
%%time
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

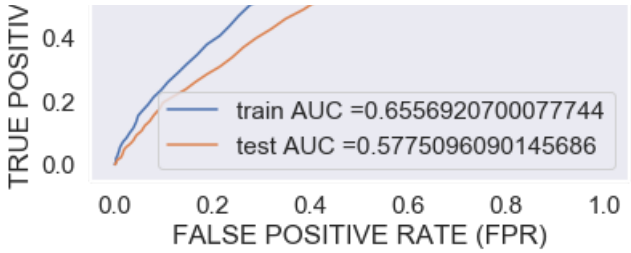

neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(X_train_avgw2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_train_avgw2v)
y_test_pred = batch_predict(neigh, X_test_avgw2v)

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE (FPR)")
plt.ylabel("TRUE POSITIVE RATE (TPR)")
plt.title("ROC PLOTS")
plt.grid()
plt.show()
```



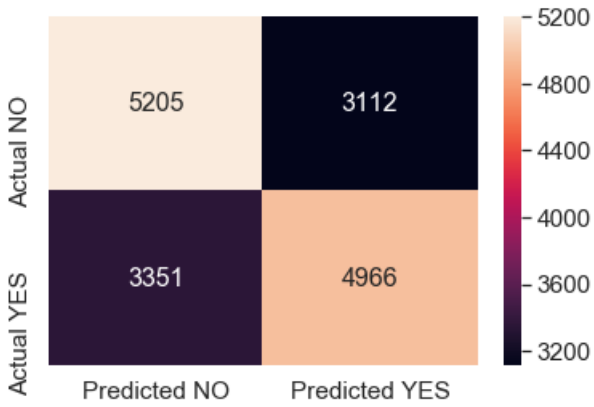



```
CPU times: user 30min 8s, sys: 20.7 s, total: 30min 29s
Wall time: 9min 31s
```

In [237]:

```
%%time
get_confusion_matrix(neigh,X_train_avgw2v,y_train)
```

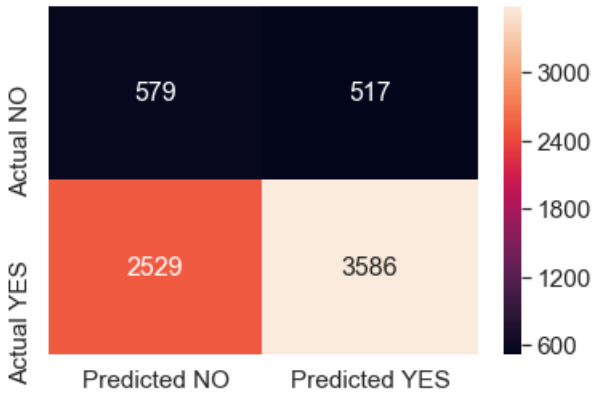
```
CPU times: user 22min 7s, sys: 12.7 s, total: 22min 19s
Wall time: 7min 8s
```



In [238]:

```
%%time
get_confusion_matrix(neigh,X_test_avgw2v,y_test)
```

```
CPU times: user 8min 52s, sys: 4.57 s, total: 8min 57s
Wall time: 2min 40s
```



2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

In [239]:

```
%%time
# I reduced data points to perform AVG W2V for 10K data points, For large data points it was not c
ompleted within 12hrs.
# Please write all the code with proper documentation
```

```
# Prepare data for BOW
X_train_tfidfw2v = hstack((X_train_school_state_ohe, X_train_clean_categories_ohe,
X_train_clean_subcategories_ohe, X_train_project_grade_category_ohe, X_train_teacher_prefix_ohe, t
fidf_w2v_vectors_text_train, tfidf_w2v_vectors_title_train,
tfidf_w2v_vectors_project_resource_summary_train, X_train_quantity_norm,
X_train_teacher_number_of_previously_posted_projects_norm, X_train_price_norm)).tocsr()
X_cv_tfidfw2v = hstack((X_cv_school_state_ohe, X_cv_clean_categories_ohe,
X_cv_clean_subcategories_ohe, X_cv_project_grade_category_ohe, X_cv_clean_teacher_prefix_ohe,
tfidf_w2v_vectors_text_cv, tfidf_w2v_vectors_title_cv,
tfidf_w2v_vectors_project_resource_summary_cv, X_cv_quantity_norm,
X_cv_teacher_number_of_previously_posted_projects_norm, X_cv_price_norm)).tocsr()
X_test_tfidfw2v = hstack((X_test_school_state_ohe, X_test_clean_categories_ohe,
X_test_clean_subcategories_ohe, X_test_project_grade_category_ohe, X_test_clean_teacher_prefix_ohe
, tfidf_w2v_vectors_text_test, tfidf_w2v_vectors_title_test,
tfidf_w2v_vectors_project_resource_summary_test, X_test_quantity_norm,
X_test_teacher_number_of_previously_posted_projects_norm, X_test_price_norm)).tocsr()


print(X_train_tfidfw2v.shape)
print(X_cv_tfidfw2v.shape)
print(X_test_tfidfw2v.shape)

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

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [1, 5, 11, 15, 31, 41, 51,71,101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X_train_tfidfw2v, y_train)

    y_train_pred = batch_predict(neigh, X_train_tfidfw2v)
    y_cv_pred = batch_predict(neigh, X_cv_tfidfw2v)

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

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

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

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

```
  0%|          | 0/9 [00:00<?, ?it/s]
```

```
(16634, 1001)
(4831, 1001)
(7211, 1001)
```

```
 11%|█         | 1/9 [08:00<1:04:01, 480.17s/it]

 22%|██        | 2/9 [24:17<1:13:25, 629.35s/it]

 33%|███       | 3/9 [31:14<56:33, 565.62s/it]

 44%|████      | 4/9 [1:22:56<1:50:32, 1326.53s/it]

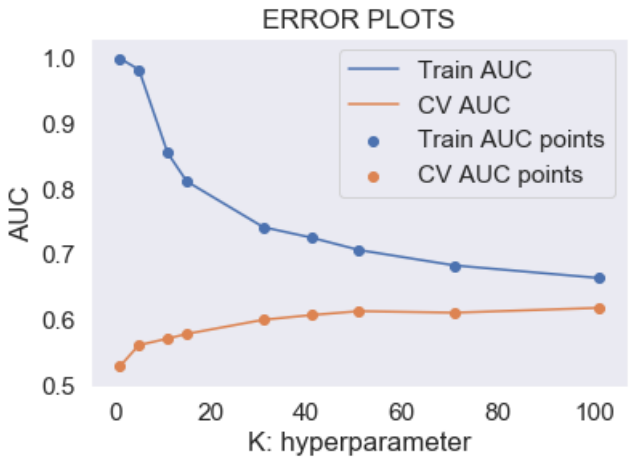
 56%|█████     | 5/9 [1:28:30<1:08:35, 1028.75s/it]

 67%|██████    | 6/9 [1:34:02<40:59, 819.68s/it]

 78%|███████   | 7/9 [1:46:36<26:39, 799.98s/it]

 89%|████████  | 8/9 [1:59:32<13:12, 792.92s/it]

100%|██████████| 9/9 [2:08:38<00:00, 718.79s/it]
```




```
CPU times: user 3h 49min 38s, sys: 2min 49s, total: 3h 52min 27s
Wall time: 2h 8min 41s
```

In [240]:

```
# from the error plot we choose K such that, we will have maximum AUC on cv data and gap between t
he train and cv is less
# Note: based on the method you use you might get different hyperparameter values as best one
# so, you choose according to the method you choose, you use gridsearch if you are having more com
puting power and note it will take more time
# if you increase the cv values in the GridSearchCV you will get more rebust results.

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

In [241]:

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


neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(X_train_tfidfw2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
```

```
y_train_pred = batch_predict(neigh, X_train_tfidfw2v)
y_test_pred = batch_predict(neigh, X_test_tfidfw2v)

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

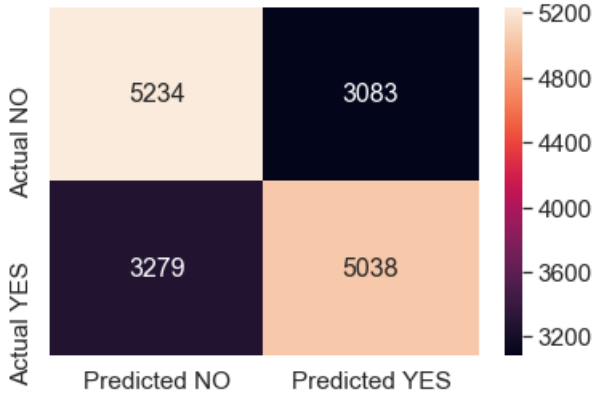
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE (FPR)")
plt.ylabel("TRUE POSITIVE RATE (TPR)")
plt.title("ROC PLOTS")
plt.grid()
plt.show()
```


```
CPU times: user 26min 52s, sys: 21.6 s, total: 27min 14s
Wall time: 8min 29s
```

In [242]:

```
%%time
get_confusion_matrix(neigh,X_train_tfidfw2v,y_train)
```

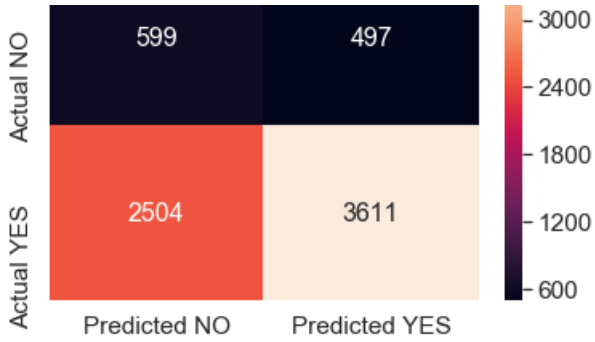
```
CPU times: user 22min 31s, sys: 18.1 s, total: 22min 49s
Wall time: 7min 56s
```



In [243]:

```
%%time
get_confusion_matrix(neigh,X_test_tfidfw2v,y_test)
```

```
CPU times: user 9min 39s, sys: 6.77 s, total: 9min 46s
Wall time: 3min 15s
```



2.5 Feature selection with `SelectKBest`

In [246]:

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

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

In [247]:

```
%%time
# print(X_train_real.shape)
# print(X_train_real.head(2))
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train_real['clean_essays'].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_real['clean_essays'].values)
X_cv_essay_tfidf = vectorizer.transform(X_cv_real['clean_essays'].values)
X_test_essay_tfidf = vectorizer.transform(X_test_real['clean_essays'].values)


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

```
(16634, 2000)
(4831, 2000)
(7211, 2000)
CPU times: user 4.73 s, sys: 351 ms, total: 5.09 s
Wall time: 5.38 s
```

In [248]:

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

vectorizer = TfidfVectorizer(min_df=5)
vectorizer.fit(X_train_real['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_real['clean_titles'].values)
X_cv_titles_tfidf = vectorizer.transform(X_cv_real['clean_titles'].values)
```

```
X_test_titles_tfidf = vectorizer.transform(X_test_real['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: (16634, 1830)
CV shape: (4831, 1830)
Test shape: (7211, 1830)
```

In [249]:

```
vectorizer = TfidfVectorizer(min_df=5)
vectorizer.fit(X_train_real['project_resource_summary'].values) # fit has to happen only on train
datadata

# we use the fitted CountVectorizer to convert the text to vector
X_train_summary_tfidf = vectorizer.transform(X_train_real['project_resource_summary'].values)
X_cv_summary_tfidf = vectorizer.transform(X_cv_real['project_resource_summary'].values)
X_test_summary_tfidf = vectorizer.transform(X_test_real['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
(16634, 3563) (16634,)
(4831, 3563) (4831,)
(7211, 3563) (7211,)
====================================================================================================
```

In [250]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((X_train_school_state_ohe, X_train_clean_categories_ohe,
X_train_clean_subcategories_ohe, X_train_project_grade_category_ohe, X_train_teacher_prefix_ohe, X
_train_essay_2000, X_train_titles_tfidf, X_train_summary_tfidf, X_train_quantity_norm,
X_train_teacher_number_of_previously_posted_projects_norm, X_train_price_norm)).tocsr()
X_cr = hstack((X_cv_school_state_ohe, X_cv_clean_categories_ohe, X_cv_clean_subcategories_ohe,
X_cv_project_grade_category_ohe, X_cv_clean_teacher_prefix_ohe, X_cv_essay_2000, X_cv_titles_tfidf
, X_cv_summary_tfidf, X_cv_quantity_norm, X_cv_teacher_number_of_previously_posted_projects_norm,
X_cv_price_norm)).tocsr()
X_te = hstack((X_test_school_state_ohe, X_test_clean_categories_ohe,
X_test_clean_subcategories_ohe, X_test_project_grade_category_ohe, X_test_clean_teacher_prefix_ohe
, X_test_essay_2000, X_test_titles_tfidf, X_test_summary_tfidf, X_test_quantity_norm,
X_test_teacher_number_of_previously_posted_projects_norm, X_test_price_norm)).tocsr()

# X_tr =
hstack((X_train_essay_2000,X_train_titles_tfidf,X_train_summary_tfidf,X_train_clean_cat_ohe,X_trai
an_subcat_ohe, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe,
X_train_price_std,X_train_projects_std,X_train_qty_std)).tocsr()
# X_cr =
hstack((X_cv_essay_2000,X_cv_titles_tfidf,X_cv_summary_tfidf,X_cv_clean_cat_ohe,X_cv_clean_subcat_
X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe,
X_cv_price_std,X_cv_projects_std,X_cv_qty_std)).tocsr()
# X_te =
hstack((X_test_essay_2000,X_test_titles_tfidf,X_test_summary_tfidf,X_test_clean_cat_ohe,X_test_cle
bcat_ohe, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe,
X_test_price_std,X_test_projects_std,X_test_qty_std)).tocsr()

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

```
Final Data matrix
(16634, 7494) (16634,)
(4831, 7494) (4831,)
(7211, 7494) (7211,)
```

==========

In [251]:

```
%%time
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

neigh = KNeighborsClassifier()
parameters = {'n_neighbors':[1, 5, 11, 15, 21, 31, 41, 51, 55, 61,71]}
clf2 = GridSearchCV(neigh, parameters, cv=10, scoring='roc_auc', n_jobs=-1, return_train_score=True
)
clf2.fit(X_tr, y_train)

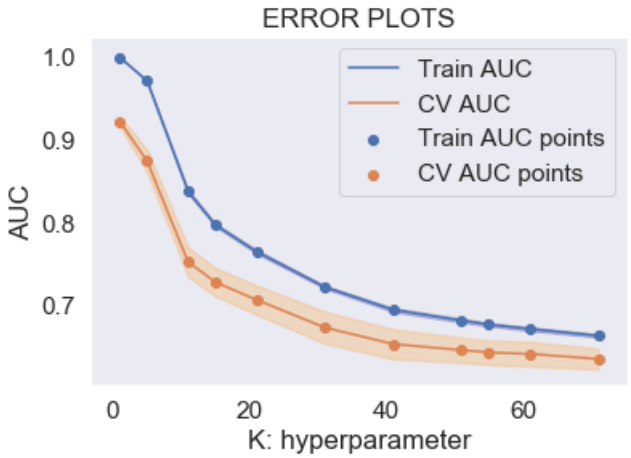
train_auc= clf2.cv_results_['mean_train_score']
train_auc_std= clf2.cv_results_['std_train_score']
cv_auc = clf2.cv_results_['mean_test_score']
cv_auc_std= clf2.cv_results_['std_test_score']

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

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

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


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




```
CPU times: user 884 ms, sys: 366 ms, total: 1.25 s
Wall time: 20min 39s
```

In [252]:

```
import pickle

pickle_out = open("clf2.pickle","wb")
pickle.dump(clf2, pickle_out)
pickle_out.close()
```

In [253]:

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


neigh = KNeighborsClassifier(n_neighbors=51,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 positive
class
# not the predicted outputs

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

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

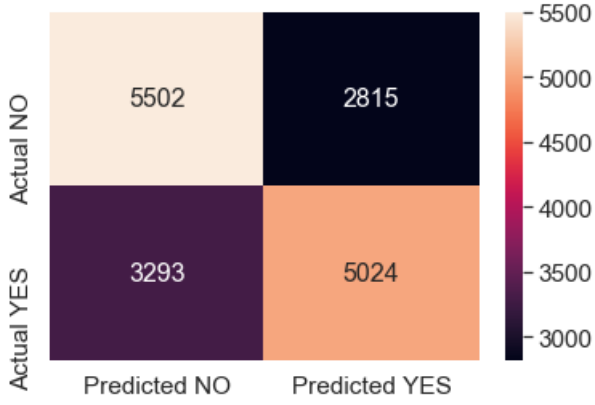
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE (FPR)")
plt.ylabel("TRUE POSITIVE RATE (TPR)")
plt.title("ROC PLOTS")
plt.grid()
plt.show()
```


```
CPU times: user 44.7 s, sys: 5.7 s, total: 50.4 s
Wall time: 24.6 s
```

In [254]:

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

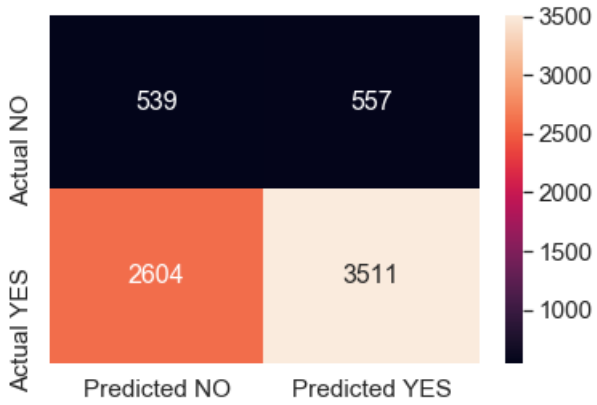
```
CPU times: user 35.8 s, sys: 4.62 s, total: 40.4 s
Wall time: 18.9 s
```



In [255]:

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

```
CPU times: user 15.3 s, sys: 1.98 s, total: 17.3 s
Wall time: 8.33 s
```



3. Conclusions

In [256]:

```
# Please compare all your models using Prettytable library
#http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyperparameter", "AUC"]
x.add_row(["Bag of Words", "Brute-Force", 101, 0.55])
x.add_row(["TFIDF", "Brute-Force", 101, 0.57])
x.add_row(["AvgW2V", "Brute-Force", 101, 0.58])
x.add_row(["TFIDFW2V", "Brute-Force", 101, 0.58])
x.add_row(["TFIDF Top 2K Features", "Brute-Force", 51, 0.55])

print(x)
```

```
+-----------------------+-------------+-----------------+------+
|       Vectorizer      |    Model    | Hyperparameter | AUC  |
+-----------------------+-------------+-----------------+------+
|      Bag of Words     | Brute-Force |      101        | 0.55 |
|          TFIDF        | Brute-Force |      101        | 0.57 |
|          AvgW2V       | Brute-Force |      101        | 0.58 |
|        TFIDFW2V       | Brute-Force |      101        | 0.58 |
| TFIDF Top 2K Features | Brute-Force |       51        | 0.55 |
+-----------------------+-------------+-----------------+------+
```

In [ ]:

```
```