

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
<code>project_id</code>		A unique identifier for the proposed project. Example: p036502
<code>project_title</code>	<ul style="list-style-type: none">••	Title of the project. Examples: Art Will Make You Happy! First Grade Fun
<code>project_grade_category</code>	<ul style="list-style-type: none">••••	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
<code>project_subject_categories</code>	<ul style="list-style-type: none">•••••••••	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
	<ul style="list-style-type: none">••	Examples: Music & The Arts Literacy & Language, Math & Science
<code>school_state</code>		State where school is located (Two-letter U.S. postal code). Example: WY
<code>project_subject_subcategories</code>	<ul style="list-style-type: none">••	One or more (comma-separated) subject subcategories for the project. Examples: Literacy Literature & Writing, Social Sciences
<code>project_resource_summary</code>	<ul style="list-style-type: none">•	An explanation of the resources needed for the project. Example: My students need hands on literacy materials to manage sensory needs!
<code>project_essay_1</code>		First application essay*
<code>project_essay_2</code>		Second application essay*
<code>project_essay_3</code>		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: <ul style="list-style-type: none"> 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 <code>project_id</code> value from the <code>train.csv</code> file. Example: p036502
description	Description 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]:

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

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
```

```

import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

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

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 [2]:

```

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

```

In [3]:

```

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

```

Number of data points in train data (109248, 17)

```

-----
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
'project_submitted_datetime' 'project_grade_category'
'project_subject_categories' 'project_subject_subcategories'
'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
'project_essay_4' 'project_resource_summary'
'teacher_number_of_previously_posted_projects' 'project_is_approved']

```

In [4]:

```

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

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

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

project_data.head(2)

```

Out[4]:

Unnamed: 0	id	teacher_id	teacher_prefix	school_state	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 [5]:

```
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[5]:

Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_subject
0	8393 p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	
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	Al
4	33679 p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5	Litera

1.2 preprocessing of project_subject_categories

In [6]:

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

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

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

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

1.3 preprocessing of project_subject_subcategories

In [7]:

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

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

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

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

1.3 Text preprocessing

In [8]:

```
# merge two column text dataframe:
project_data["essay"] = project_data["project_essay_1"].map(str) + \
    project_data["project_essay_2"].map(str) + \
    project_data["project_essay_3"].map(str) + \
    project_data["project_essay_4"].map(str)
```

In [9]:

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

Out[9]:

Unnamed: 0	id	teacher_id	teacher_prefix	school_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 [10]:

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

In [11]:

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

I have been fortunate enough to use the Fairy Tale STEM kits in my classroom as well as the STEM journals, which my students really enjoyed. I would love to implement more of the Lakeshore STEM kits in my classroom for the next school year as they provide excellent and engaging STEM lessons. My students come from a variety of backgrounds, including language and socioeconomic status. 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 instruction in engaging and meaningful ways. I can adapt the kits to my current language arts pacing guide where we already teach some of the 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 develop 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.

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 desire to def

a dangerous neighborhood. Despite these challenges, I have students who have 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.

"Life moves pretty fast. If you don't stop and look around once in awhile, you could miss it." from the movie, Ferris Bueller's Day Off. Think back...what do you remember about your grandparents? How amazing would it be to be able to flip through a book to see a day in their lives? My second graders are voracious readers! They love to read both fiction and nonfiction books. Their favorite characters include Pete the Cat, Fly Guy, Piggie and Elephant, and Mercy Watson. They also love to read about insects, space and plants. My students are hungry bookworms! My students are eager to learn and read about the world around them. My kids love to be at school and are like little sponges absorbing everything around them. Their parents work long hours and usually do not see their children. My students are usually cared for by their grandparents or a family friend. Most of my students do not have someone who speaks English at home. Thus it is difficult for my students to acquire language. Now think forward... wouldn't it mean a lot to your kids, nieces or nephews or grandchildren, to be able to see a day in your life today 30 years from now? Memories are so precious to us and being able to share these memories with future generations will be a rewarding experience. As part of our social studies curriculum, students will be learning about changes over time. Students will be studying photos to learn about how their community has changed over time. In particular, we will look at photos to study how the land, buildings, clothing, and schools have changed over time. As a culminating activity, my students will capture a slice of their history and preserve it through scrap booking. Key important events in their young lives will be documented with the date, location, and names. Students will be using photos from home and from school to create their second grade memories. Their scrap books will preserve their unique stories for future generations to enjoy. Your donation to this project will provide my second graders with an opportunity to learn about social studies in a fun and creative manner. Through their scrapbooks, children will share their story with others and have a historical document for the rest of their lives.

"A person's a person, no matter how small." (Dr. Seuss) I teach the smallest students with the biggest enthusiasm for learning. My students learn in many different ways using all of our senses and multiple intelligences. I use a wide range of techniques to help all my students succeed. 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 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. Students will gain math and literature skills as well as a life long enjoyment for healthy cooking.

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. The students look forward to their work time so they can move around the room. I would like to get rid of the

look forward to their work time so they can move around the room. I would like to get rid of the constricting desks and move toward more "fun" seating options. I am requesting various seating so my students have more options to sit. Currently, I have a stool and a papasan chair I inherited from the previous sixth-grade teacher as well as five milk crate seats I made, but I would like to give them more options and reduce the competition for the "good seats". I am also requesting two rugs as not only more seating options but to make the classroom more welcoming and appealing. In order 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 tables that we can fold up when we are not using them to leave more room for our flexible seating options.

I 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 [12]:

```
# 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 [13]:

```
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 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. \nStudents in my class come from a variety of different backgrounds which makes for wonderful sharing of experiences and cultures, including Native Americans.\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 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 classroom garden in the spring. We will also create our own cookbooks to be printed and shared with families. \nStudents will gain math and literature skills as well as a life long enjoyment for healthy cooking.nannan

=====

In [14]:

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

A person is a person, no matter how small. (Dr.Seuss) I teach the smallest students with the biggest enthusiasm for learning. My students learn in many different ways using all of our senses and

multiple intelligences. I use a wide range of techniques to help all my students succeed. 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 knowledge of where the ingredients came from as well as how it is healthy for their bodies. This project would expand our learning of nutrition and agricultural cooking recipes by having us peel our own apples to make homemade applesauce, make our own bread, and mix up healthy plants from our classroom garden in the spring. We will also create our own cookbooks to be printed and shared with families. Students will gain math and literature skills as well as a life long enjoyment for healthy cooking.annan

In [15]:

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

A person is a person no matter how small Dr Seuss I teach the smallest students with the biggest enthusiasm for learning My students learn in many different ways using all of our senses and multiple intelligences I use a wide range of techniques to help all my students succeed 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 knowledge of where the ingredients came from as well as how it is healthy for their bodies This project would expand our learning of nutrition and agricultural cooking recipes by having us peel our own apples to make homemade applesauce make our own bread and mix up healthy plants from our classroom garden in the spring We will also create our own cookbooks to be printed and shared with families Students will gain math and literature skills as well as a life long enjoyment for healthy cooking.annan

In [16]:

```
# 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', "dc
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn', "mustn't", 'shan', "shan't", 'won', "won't", 'wouldn', "wouldn't"]
```

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

In [17]:

```
# Create function that will filter sentence
def filterSentence(sentence):
    sent = decontracted(sentence)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    sent = sent.lower()
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    return sent.strip()
```

In [18]:

```
# Combining all the above students
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentence in tqdm(project_data['essay'].values):
    preprocessed_essays.append(filterSentence(sentence))
```

100%|██████████| 109248/109248 [01:09<00:00, 1569.45it/s]

In [19]:

```
# after preprocessing
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 help students succeed students class come variety different backgrounds makes wonderful sharing experiences cultures including native americans school caring community successful learners seen collaborative student project based learning classroom kindergarteners class love work hands materials many different opportunities practice skill mastered social skills work cooperatively friends crucial 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 idea create common core cooking lessons learn important math writing concepts cooking delicious healthy food snack time students grounded appreciation work went making food knowledge ingredients came well healthy bodies project would expand learning nutrition agricultural cooking recipes use peel apples make homemade applesauce make bread mix healthy plants classroom garden spring also create cookbooks printed families students gain math literature skills well life long enjoyment healthy cooking nannan'

1.4 Preprocessing of `project_title`

In [20]:

```
# similarly you can preprocess the titles also
# Combining all the above students
from tqdm import tqdm
preprocessed_titles = []
# tqdm is for printing the status bar
for sentence in tqdm(project_data['project_title'].values):
    preprocessed_titles.append(filterSentence(sentence))
```

100%|██████████| 109248/109248 [00:03<00:00, 27906.99it/s]

In [21]:

```
# after preprocessing
```

```
print(preprocessed_titles[20000])
```

health nutritional cooking kindergarten

In [22]:

```
# similarly you can preprocess the project_resource_summary also
# Combining all the above students
from tqdm import tqdm
preprocessed_resource_summary = []
# tqdm is for printing the status bar
for sentence in tqdm(project_data['project_resource_summary'].values):
    preprocessed_resource_summary.append(filterSentence(sentence))
```

```
100%|██████████| 109248/109248 [00:06<00:00, 16195.89it/s]
```

In [23]:

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

students need cooking supplies help us healthy learn nutrition mixer apple spiralizer kitchen
tools nutrition kit kid friendly healthy literature ink make cookbooks

In [24]:

```
# 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)
    clean_teacher_prefix = clean_teacher_prefix.lower()
    if clean_teacher_prefix in stopwords:
        continue
    preprocessed_teacher_prefix.append(clean_teacher_prefix.strip())
```

```
100%|██████████| 109248/109248 [00:01<00:00, 58054.17it/s]
```

In [25]:

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

Out[25]:

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

In [26]:

```
# 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)
    clean_project_grade_category = clean_project_grade_category.lower()
    if clean_project_grade_category in stopwords:
        continue
    preprocessed_project_grade_category.append(clean_project_grade_category.strip())
```

100%|██████████| 109248/109248 [00:01<00:00, 60045.47it/s]

In [27]:

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

Out[27]:

```
['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 [28]:

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

In [30]:

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

In [31]:

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

Out[31]:

Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_title
------------	----	------------	----------------	--------------	------	------------------------	---------------

Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	Project Title
0	8390	p205479	2bf07ba08945e5d8b2a3f2d9b2b3c4e3	ms	2016-04-27 00:27:36	grades prek 2	Engineering the Primary Classroom
1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	ms	2016-04-27 00:31:25	grades 3 5	Sensory Tools for Focus

1.5 Preparing data for models

In [32]:

```
project_data.columns
```

Out[32]:

```
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 [33]:

```
print(project_data.shape)

# I am taking 100% of data points for my analysis
project_data = project_data.sample(frac=1)

print(project_data.shape)
```

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

In [34]:

```
# 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[34]:

```
(109248, 17)
```

In [35]:

```
# 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 (49041, 17) (49041,)
Cross Validate Data Set (24155, 17) (24155,)
Test Data Set (36052, 17) (36052,)
```

In [36]:

```
# Commented code as per your suggestion
# # Handle imbalanced 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)
```

In [37]:

```
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 (49041, 17) (49041,)
Cross Validate Data Set (24155, 17) (24155,)
Test Data Set (36052, 17) (36052,)
```



1.5.1 Vectorizing Categorical data

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

In [38]:

```
# 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)

school_state_features = vectorizer.get_feature_names()

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)
```

```
(49041, 51) (49041,)
(24155, 51) (24155,)
(36052, 51) (36052,)
['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 [39]:

```
# 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)

clean_categories_features = vectorizer.get_feature_names()

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)
```

```
(49041, 9) (49041,)
(24155, 9) (24155,)
(36052, 9) (36052,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
*****
```

In [40]:

```
# 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)

clean_subcategories_features = vectorizer.get_feature_names()

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)
```

```
(49041, 30) (49041,)
(24155, 30) (24155,)
(36052, 30) (36052,)
['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 [41]:

```
# 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)

project_grade_category_features = vectorizer.get_feature_names()

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)

(49041, 4) (49041,)
(24155, 4) (24155,)
(36052, 4) (36052,)
['grades 3 5', 'grades 6 8', 'grades 9 12', 'grades prek 2']
*****
```

In [42]:

```
# 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)

teacher_prefix_features = vectorizer.get_feature_names()

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)
```

```
(49041, 6) (49041,)
(24155, 6) (24155,)
(36052, 6) (36052,)
['dr', 'mr', 'mrs', 'ms', 'nan', 'teacher']
*****
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [43]:

```
# Sample code for bigram extraction using TFIDF
# from sklearn.feature_extraction.text import TfidfVectorizer
# corpus = [
#     'This is the first document.',
#     'This document is the second document.',
#     'And this is the third one.',
#     'Is this the first document?',
# ]
# vectorizer = TfidfVectorizer(ngram_range=(2,2))
# X = vectorizer.fit_transform(corpus)
# print(vectorizer.get_feature_names())

# print(X.shape)
```

In [44]:

```
# - project_title : text data
print(X_train.shape, y_train.shape)
```



```

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)

clean_titles_bow_features = vectorizer.get_feature_names()

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(vectorizer.get_feature_names())
print("""*100)

```

```

(49041, 17) (49041,)
(24155, 17) (24155,)
(36052, 17) (36052,)
*****

```

```

After vectorizations
(49041, 3398) (49041,)
(24155, 3398) (24155,)
(36052, 3398) (36052,)
*****

```



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)

# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10,ngram_range=(2,2), max_features=5000)
vectorizer.fit(X_train['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_bow = vectorizer.transform(X_train['clean_essays'].values)
X_cv_essay_bow = vectorizer.transform(X_cv['clean_essays'].values)
X_test_essay_bow = vectorizer.transform(X_test['clean_essays'].values)

easy_bow_features = vectorizer.get_feature_names()

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(vectorizer.get_feature_names())
print("""*100)

```

```

(49041, 17) (49041,)
(24155, 17) (24155,)
(36052, 17) (36052,)
*****

```

```

After vectorizations
(49041, 5000) (49041,)
(24155, 5000) (24155,)
(36052, 5000) (36052,)
*****

```



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)

# 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'].values)
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)

project_resource_summary_bow_features = vectorizer.get_feature_names()

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(vectorizer.get_feature_names())
print("""*100)
```

```
(49041, 17) (49041,)
(24155, 17) (24155,)
(36052, 17) (36052,)
*****
```

```
After vectorizations
(49041, 5000) (49041,)
(24155, 5000) (24155,)
(36052, 5000) (36052,)
*****
```



1.5.2.2 TFIDF vectorizer

In [47]:

```
# - 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)

clean_titles_tfidf_features = vectorizer.get_feature_names()

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

```
(49041, 17) (49041,)
(24155, 17) (24155,)
(36052, 17) (36052,)
```

```
(36052, 17) (36052,)
*****

After vectorizations
(49041, 1988) (49041,)
(24155, 1988) (24155,)
(24155, 1988) (36052,)
*****
```

In [48]:

```
# - 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, ngram_range=(2,2), max_features=5000)
vectorizer.fit(X_train['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['clean_essays'].values)
X_cv_essay_tfidf = vectorizer.transform(X_cv['clean_essays'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['clean_essays'].values)

easy_tfidf_features = vectorizer.get_feature_names()

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)
```

```
(49041, 17) (49041,)
(24155, 17) (24155,)
(36052, 17) (36052,)
*****
```

```
After vectorizations
(49041, 5000) (49041,)
(24155, 5000) (24155,)
(36052, 5000) (36052,)
*****
```

In [49]:

```
# - 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)

project_resource_summary_tfidf_features = vectorizer.get_feature_names()

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)
```

```
(49041, 17) (49041,)
(24155, 17) (24155,)
(36052, 17) (36052,)
*****
```

After vectorizations

```
(49041, 3879) (49041,)
(24155, 3879) (24155,)
(36052, 3879) (36052,)
*****
```



1.5.2.3 Using Pretrained Models: Avg W2V

In [50]:

```
'''
# 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 preprocod_texts:
    words.extend(i.split(' '))

for i in preprocod_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[50]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef
loadGloveModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFile,\nencoding="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        model[word] = embedding\n    print ("Done.",len(model)," words loaded!")\n    return model\nmodel =
loadGloveModel('glove.42B.300d.txt')\n\n# =====\nOutput:\nLoading G
love Model\n1917495it [06:32, 4879.69it/s]\nDone. 1917495 words loaded!\n\n#
=====
words = []\nfor i in preproced_texts:\n    words.extend(i.split('\n\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]\nr
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 [51]:

```
# 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 [52]:

```
# 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['clean_essays'].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%|██████████| 49041/49041 [00:22<00:00, 2206.33it/s]

49041
300

In [53]:

```
# 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['clean_essays'].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%|██████████| 24155/24155 [00:10<00:00, 2229.70it/s]

24155
300

In [54]:

```
# 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['clean_essays'].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%|██████████| 36052/36052 [00:11<00:00, 3033.08it/s]

36052
300

In [55]:

```
# 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%|██████████| 49041/49041 [00:00<00:00, 52351.83it/s]

49041
300

In [56]:

```
# 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%|██████████| 24155/24155 [00:00<00:00, 42066.64it/s]

24155
300

In [57]:

```

# 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%|██████████| 36052/36052 [00:01<00:00, 32976.39it/s]

36052
300

In [58]:

```

# 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 stored 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%|██████████| 49041/49041 [00:02<00:00, 18033.37it/s]

49041
300

In [59]:

```
# 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%|██████████| 24155/24155 [00:01<00:00, 20727.18it/s]

24155
300

In [60]:

```
# 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%|██████████| 36052/36052 [00:01<00:00, 23690.71it/s]

36052
300

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [61]:

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

In [62]:

In [62]:

```
# 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['clean_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the 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%|██████████| 49041/49041 [02:20<00:00, 348.33it/s]

49041
300

In [63]:

```
# 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['clean_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the 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%|██████████| 24155/24155 [01:03<00:00, 382.07it/s]

24155
300

In [64]:

```
# 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['clean_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
```

```

value((sentence.count(word)/len(sentence.split()))
      tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the 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%|██████████| 36052/36052 [01:15<00:00, 480.09it/s]

36052
300

In [65]:

```

# 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 [66]:

```

# 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%|██████████| 49041/49041 [00:02<00:00, 18759.87it/s]

49041
300

In [67]:

```

# 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%|██████████| 24155/24155 [00:01<00:00, 14720.75it/s]

24155
300

In [68]:

```

# 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%|██████████| 36052/36052 [00:03<00:00, 10965.94it/s]

36052
300

In [69]:

```

# 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 [70]:

```

# 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%|██████████| 49041/49041 [00:14<00:00, 3421.29it/s]

49041
300

In [71]:

```

# 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%|██████████| 24155/24155 [00:08<00:00, 2850.75it/s]

24155
300

In [72]:

```

# 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%|██████████| 36052/36052 [00:05<00:00, 6848.64it/s]

36052
300

1.5.3 Vectorizing Numerical features

In [73]:

```

# 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
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)

In [74]:

```

# 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

```
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
```

In [75]:

```
# - 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

```
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
```

1.5.4 Merging all the above features

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

In [76]:

```
# 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

```

*****

(49041, 51) (49041,)
(24155, 51) (24155,)
(36052, 51) (36052,)
*****

(49041, 9) (49041,)
(24155, 9) (24155,)
(36052, 9) (36052,)
*****

(49041, 30) (49041,)
(24155, 30) (24155,)
(36052, 30) (36052,)
*****

(49041, 4) (49041,)
(24155, 4) (24155,)
(36052, 4) (36052,)
*****

(49041, 6) (49041,)
(24155, 6) (24155,)
(36052, 6) (36052,)
*****

```

Text Encoding Features

```

*****

(49041, 3398) (49041,)
(24155, 3398) (24155,)
(36052, 3398) (36052,)
*****

(49041, 5000) (49041,)
(24155, 5000) (24155,)
(36052, 5000) (36052,)
*****

(49041, 5000) (49041,)
(24155, 5000) (24155,)
(36052, 5000) (36052,)
*****

(49041, 1988) (49041,)
(24155, 1988) (24155,)
(24155, 1988) (36052,)
*****

(49041, 5000) (49041,)
(24155, 5000) (24155,)
(36052, 5000) (36052,)
*****

(49041, 3879) (49041,)
(24155, 3879) (24155,)
(36052, 3879) (36052,)
*****

```


(30002, 3019) (30002,)

49041

300

24155

300

36052

300

49041

300

24155

300

36052

300

49041

300

24155

300

36052

300

49041

300

24155

300

36052

300

49041

300

24155

300

36052

300

49041

300

24155

300

36052

300

Numerical Features

```
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
*****

(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
*****

(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
```

In [77]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatenating a sparse matrix and a dense matrix :)
# 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)
```

```
(49041, 17)
(24155, 17)
(36052, 17)
(49041, 26168)
(24155, 26168)
(36052, 26168)
```

Computing Sentiment Scores

In [78]:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# import nltk
# nltk.download('vader_lexicon')
```

```

sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest students w
ith the biggest enthusiasm \
for learning my students learn in many different ways using all of our senses and multiple intelli
gences i use a wide range\
of techniques to help all my students succeed students in my class come from a variety of differen
t backgrounds which makes\
for wonderful sharing of experiences and cultures including native americans our school is a carin
g 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 ki
ndergarten curriculum\
montana is the perfect place to learn about agriculture and nutrition my students love to role pla
y 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 co
oking delicious healthy \
food for snack time my students will have a grounded appreciation for the work that went into maki
ng the food and knowledge \
of where the ingredients came from as well as how it is healthy for their bodies this project woul
d expand our learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade apple
sauce make our own bread \
and mix up healthy plants from our classroom garden in the spring we will also create our own cook
books to be printed and \
shared with families students will gain math and literature skills as well as a life long enjoymen
t for healthy cooking \
nannan'
ss = sid.polarity_scores(for_sentiment)

for k in ss:
    print('{0}: {1}, '.format(k, ss[k]), end='')

# we can use these 4 things as features/attributes (neg, neu, pos, compound)
# neg: 0.0, neu: 0.753, pos: 0.247, compound: 0.93

neg: 0.01, neu: 0.745, pos: 0.245, compound: 0.9975,

```

Assignment 5: Logistic Regression




1. [Task-1] Logistic Regression(either SGDClassifier with log loss, or LogisticRegression) on these feature sets

- **Set 1:** categorical, numerical features + project_title(BOW) + preprocessed_eassay ('BOW with bi-grams' with 'min_df=10' and 'max_features=5000')
- **Set 2:** categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay ('TFIDF with bi-grams' with 'min_df=10' and 'max_features=5000')
- **Set 3:** categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
- **Set 4:** categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)

2. Hyper paramter tuning (find best hyper parameters corresponding the algorithm that you choose)

- Find the best hyper parameter which will give 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 you can also write your own for loops to do this task of hyperparameter tuning

3. Representation of results

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure. 
- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test. 
- Along with plotting ROC curve, you need to print the [confusion matrix](#) with predicted and original labels of test data points. Please visualize your confusion matrices using [seaborn heatmaps](#). 

4. [\[Task-2\] Apply Logistic Regression on the below feature set Set 5 by finding the best hyper parameter as suggested in step 2 and step 3.](#)

5. [Consider these set of features Set 5:](#)

- [school_state](#) : categorical data
- [clean_categories](#) : categorical data
- [clean_subcategories](#) : categorical data
- [project_grade_category](#) : categorical data
- [teacher_prefix](#) : categorical data
- [quantity](#) : numerical data
- [teacher_number_of_previously_posted_projects](#) : numerical data
- [price](#) : numerical data
- [sentiment score's of each of the essay](#) : numerical data
- [number of words in the title](#) : numerical data
- [number of words in the combine essays](#) : numerical data

[And apply the Logistic regression on these features by finding the best hyper paramter as suggested in step 2 and step 3](#)

6. 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-leakage, 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. Logistic Regression

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 [79]:

```
# 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 [80]:

```
# 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 [81]:

```
# 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 Applng Logistic Regression on different kind of featurization as mentioned in the instructions

Apply Logistic Regression 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 [82]:

```
#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')
```

2.4.1 Applying Logistic Regression(either SGDClassifier with log loss, or LogisticRegression) on SET 1

- **Set 1:** categorical, numerical features + project_title(BOW) + preprocessed_eassay ('BOW with bi-grams` with `min_df=10` and `max_features=5000`)

In [91]:

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

# Prepare data for BOW
X_train_bow = hstack((X_train_school_state_ohc, X_train_clean_categories_ohc,
X_train_clean_subcategories_ohc, X_train_project_grade_category_ohc, X_train_teacher_prefix_ohc, 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_ohc, X_cv_clean_categories_ohc, X_cv_clean_subcategories_ohc,
X_cv_project_grade_category_ohc, X_cv_clean_teacher_prefix_ohc, 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_ohc, X_test_clean_categories_ohc,
X_test_clean_subcategories_ohc, X_test_project_grade_category_ohc, X_test_clean_teacher_prefix_ohc
,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.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import learning_curve, GridSearchCV
```

```

"""
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.

"""

tuned_parameters = [{ 'C': [10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}]
lr=GridSearchCV(LogisticRegression(class_weight='balanced'), tuned_parameters, cv=5,
scoring='roc_auc', return_train_score=True)
lr.fit(X_train_bow, y_train);

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

plt.plot(tuned_parameters[0]['C'], train_auc, label='Train AUC')
plt.plot(tuned_parameters[0]['C'], cv_auc, label='CV AUC')

plt.scatter(tuned_parameters[0]['C'], train_auc, label='Train AUC points')
plt.scatter(tuned_parameters[0]['C'], cv_auc, label='CV AUC points')
plt.xscale('log')# we take the log in the x axis

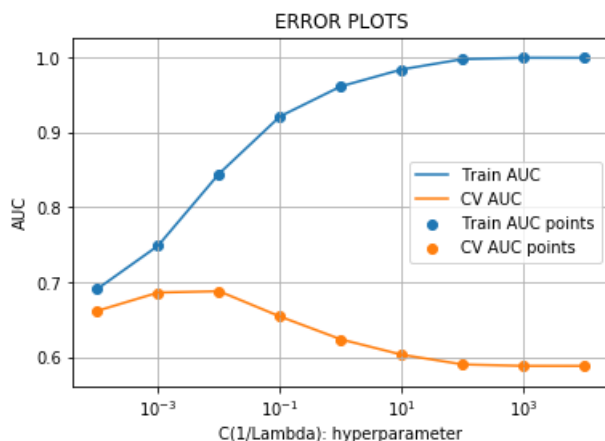
plt.legend()
plt.xlabel("C(1/Lambda): hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()

```

```

(49041, 13520) (49041,)
(24155, 13520) (24155,)
(36052, 13520) (36052,)

```



```

CPU times: user 17min 19s, sys: 12.7 s, total: 17min 32s
Wall time: 10min

```

In [92]:

```

# 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_c = 10**-3

```

In [93]:

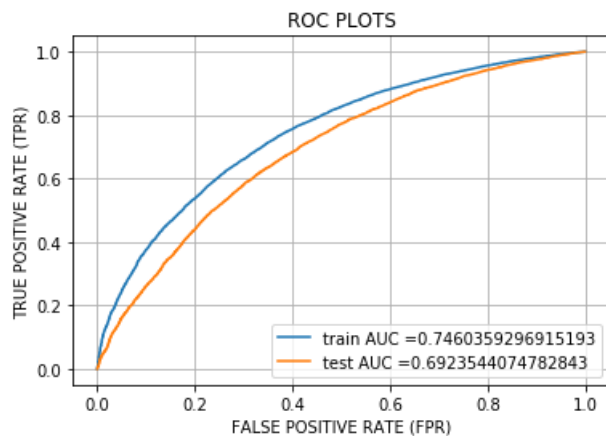
```
%%time
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

lr = LogisticRegression(C=best_c, class_weight='balanced');
lr.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 = lr.predict_proba(X_train_bow)[:,1]
y_test_pred = lr.predict_proba(X_test_bow)[:,1]

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()
```

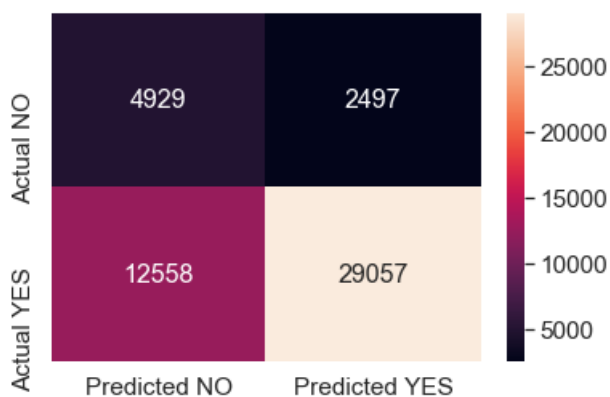


CPU times: user 1.31 s, sys: 47.2 ms, total: 1.36 s
Wall time: 878 ms

In [94]:

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

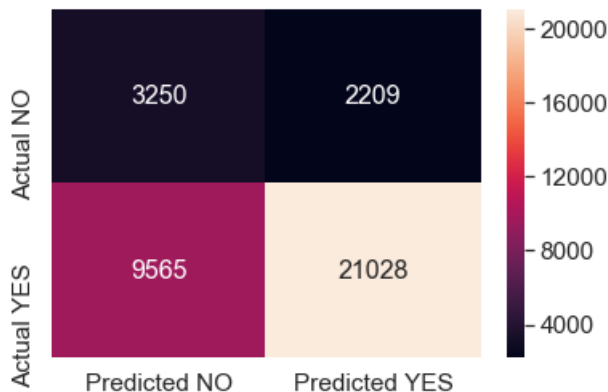
CPU times: user 111 ms, sys: 9.77 ms, total: 121 ms
Wall time: 136 ms



In [95]:

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

CPU times: user 93.8 ms, sys: 2.27 ms, total: 96.1 ms
Wall time: 97.1 ms



2.4.2 Applying Logistic Regression(either SGDClassifier with log loss, or LogisticRegression) on , SET 2

- **Set 2:** categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay ('TFIDF with bi-grams' with 'min_df=10' and 'max_features=5000')

In [96]:

```
%%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.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import learning_curve, GridSearchCV

"""
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.

"""

tuned_parameters = [{'C': [10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}]
```



```

lr=GridSearchCV(LogisticRegression(class_weight='balanced'), tuned_parameters, cv=5,
scoring='roc_auc', return_train_score=True)
lr.fit(X_train_tfidf, y_train);

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

plt.plot(tuned_parameters[0]['C'], train_auc, label='Train AUC')
plt.plot(tuned_parameters[0]['C'], cv_auc, label='CV AUC')

plt.scatter(tuned_parameters[0]['C'], train_auc, label='Train AUC points')
plt.scatter(tuned_parameters[0]['C'], cv_auc, label='CV AUC points')
plt.xscale('log')# we take the log in the x axis

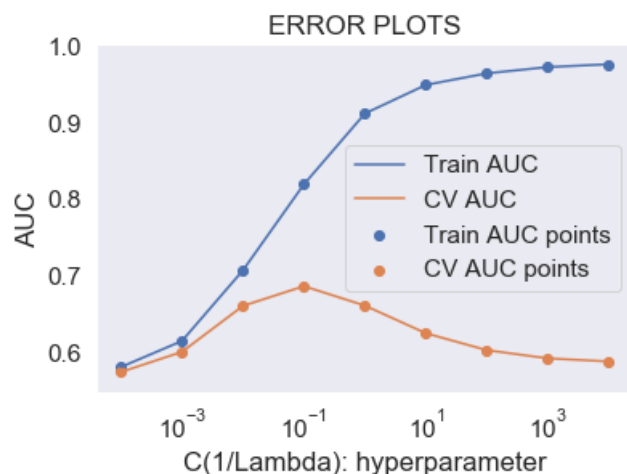
plt.legend()
plt.xlabel("C(1/Lambda): hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()

```

```

(49041, 11013)
(24155, 11013)
(36052, 11013)

```



```

CPU times: user 19min 26s, sys: 11.4 s, total: 19min 37s
Wall time: 10min 40s

```

In [97]:

```

# 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_c = 10**-2

```

In [98]:

```

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

lr = LogisticRegression(C=best_c,class_weight='balanced');
lr.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 = lr.predict_proba(X_train_tfidf)[:,1]
y_test_pred = lr.predict_proba(X_test_tfidf)[:,1]

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 2.09 s, sys: 37.3 ms, total: 2.12 s
Wall time: 1.21 s

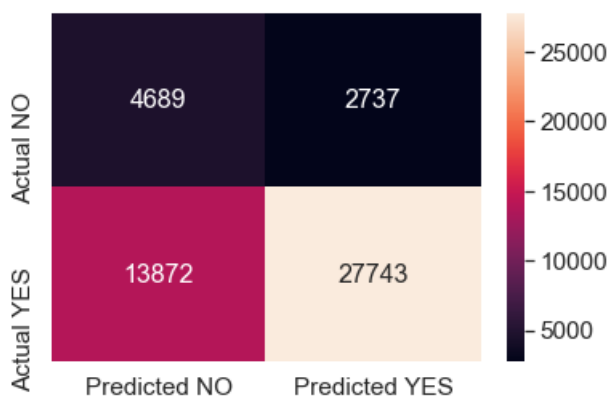
In [99]:

```

%%time
get_confusion_matrix(lr,X_train_tfidf,y_train)

```

CPU times: user 97.7 ms, sys: 3.24 ms, total: 101 ms
Wall time: 102 ms



In [100]:

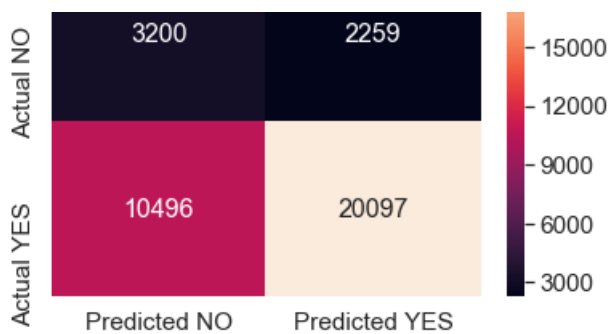
```

%%time
get_confusion_matrix(lr,X_test_tfidf,y_test)

```

CPU times: user 93.2 ms, sys: 4.59 ms, total: 97.8 ms
Wall time: 104 ms





2.4.3 Applying Logistic Regression(either SGDClassifier with log loss, or LogisticRegression) on , SET 3

- **Set 3:** categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)

In [102]:

```
%%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_oh, X_train_clean_categories_oh,
X_train_clean_subcategories_oh, X_train_project_grade_category_oh, X_train_teacher_prefix_oh, 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_oh, X_cv_clean_categories_oh,
X_cv_clean_subcategories_oh, X_cv_project_grade_category_oh, X_cv_clean_teacher_prefix_oh,
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_oh, X_test_clean_categories_oh,
X_test_clean_subcategories_oh, X_test_project_grade_category_oh, X_test_clean_teacher_prefix_oh
, 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.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import learning_curve, GridSearchCV

"""
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.

"""

tuned_parameters = [{ 'C': [10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}]
lr=GridSearchCV(LogisticRegression(class_weight='balanced'), tuned_parameters, cv=5,
scoring='roc_auc', return_train_score=True)
lr.fit(X_train_avgw2v, y_train);

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

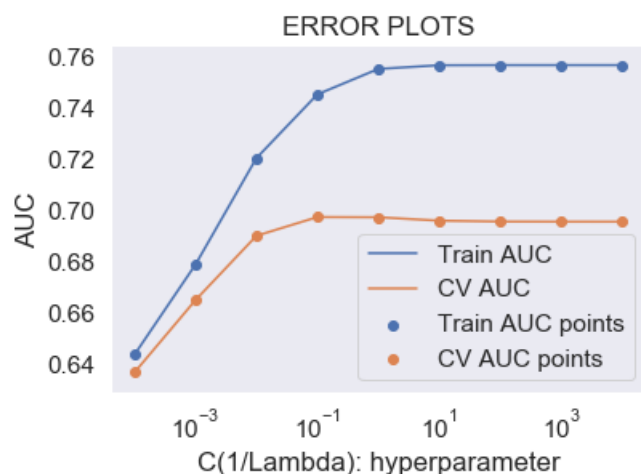
plt.plot(tuned_parameters[0][ 'C'], train_auc, label='Train AUC')
```

```
plt.plot(tuned_parameters[0]['C'], cv_auc, label='CV AUC')

plt.scatter(tuned_parameters[0]['C'], train_auc, label='Train AUC points')
plt.scatter(tuned_parameters[0]['C'], cv_auc, label='CV AUC points')
plt.xscale('log')# we take the log in the x axis

plt.legend()
plt.xlabel("C(1/Lambda): hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
(49041, 1002)
(24155, 1002)
(36052, 1002)
```



```
CPU times: user 38min 58s, sys: 38.2 s, total: 39min 36s
Wall time: 46min 48s
```

In [103]:

```
# 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_c = 10**-2
```

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

lr = LogisticRegression(C=best_c,class_weight='balanced');
lr.fit(X_train_avg2v ,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 = lr.predict_proba(X_train_avg2v)[:,-1]
y_test_pred = lr.predict_proba(X_test_avg2v)[:,-1]

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()
```

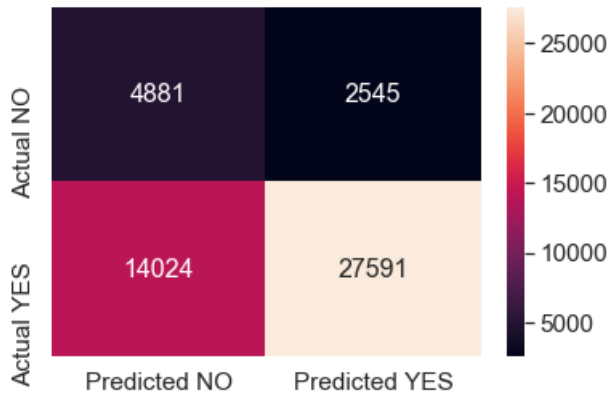


CPU times: user 14.3 s, sys: 808 ms, total: 15.1 s
Wall time: 16.6 s

In [105]:

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

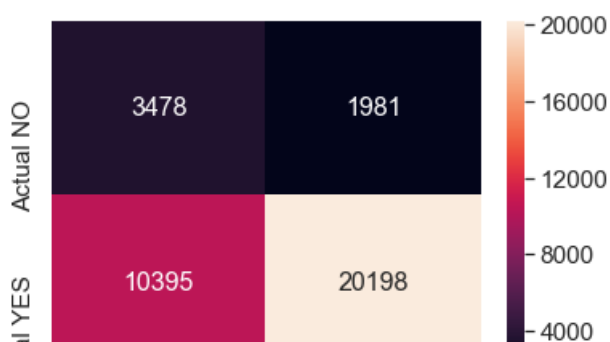
CPU times: user 216 ms, sys: 43.6 ms, total: 260 ms
Wall time: 344 ms



In [106]:

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

CPU times: user 170 ms, sys: 6.37 ms, total: 177 ms
Wall time: 191 ms



2.4.4 Applying Logistic Regression(either SGDClassifier with log loss, or LogisticRegression) on , SET 4

- **Set 4:** categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)

In [107]:

```
%%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_tfidf_w2v = 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_tfidf_w2v = 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_tfidf_w2v = 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_tfidf_w2v.shape)
print(X_cv_tfidf_w2v.shape)
print(X_test_tfidf_w2v.shape)

import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import learning_curve, GridSearchCV

"""
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.

"""

tuned_parameters = [{'C': [10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}]
lr=GridSearchCV(LogisticRegression(class_weight='balanced'), tuned_parameters, cv=3,
scoring='roc_auc', return_train_score=True)
lr.fit(X_train_tfidf_w2v, y_train);

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

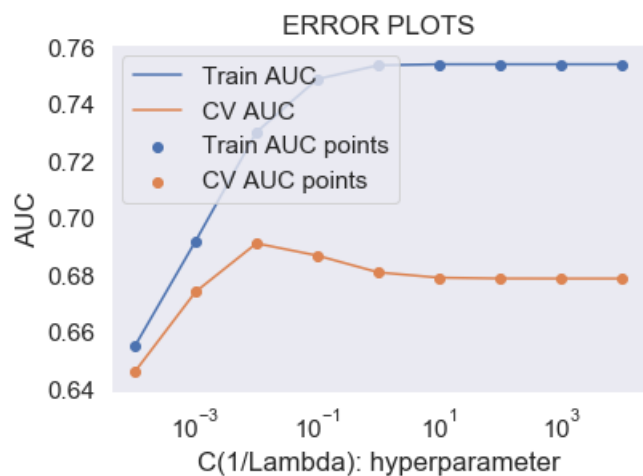
plt.plot(tuned_parameters[0]['C'], train_auc, label='Train AUC')
plt.plot(tuned_parameters[0]['C'], cv_auc, label='CV AUC')

plt.scatter(tuned_parameters[0]['C'], train_auc, label='Train AUC points')
plt.scatter(tuned_parameters[0]['C'], cv_auc, label='CV AUC points')
plt.xscale('log')# we take the log in the x axis

plt.legend()
plt.xlabel("C(1/Lambda) : hyperparameter")
plt.ylabel("AUC")
```

```
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
(49041, 1002)
(24155, 1002)
(36052, 1002)
```



CPU times: user 15min 54s, sys: 27.8 s, total: 16min 21s
Wall time: 20min 40s

In [108]:

```
# 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_c = 10**-2
```

In [109]:

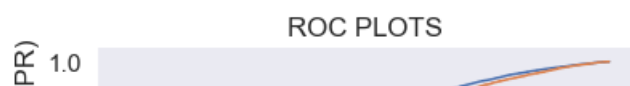
```
%%time
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

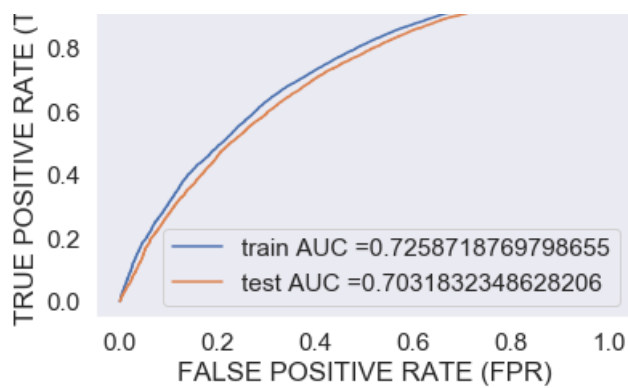
lr = LogisticRegression(C=best_c, class_weight='balanced');
lr.fit(X_train_tfidfw2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = lr.predict_proba(X_train_tfidfw2v)[:,1]
y_test_pred = lr.predict_proba(X_test_tfidfw2v)[:,1]

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()
```



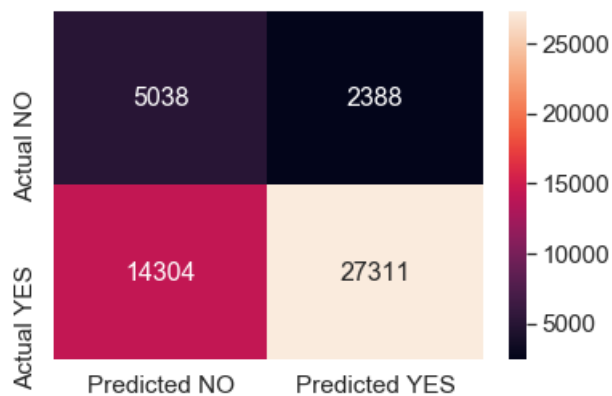


CPU times: user 9.58 s, sys: 930 ms, total: 10.5 s
Wall time: 11.5 s

In [111]:

```
%%time
get_confusion_matrix(lr,X_train_tfidf2v,y_train)
```

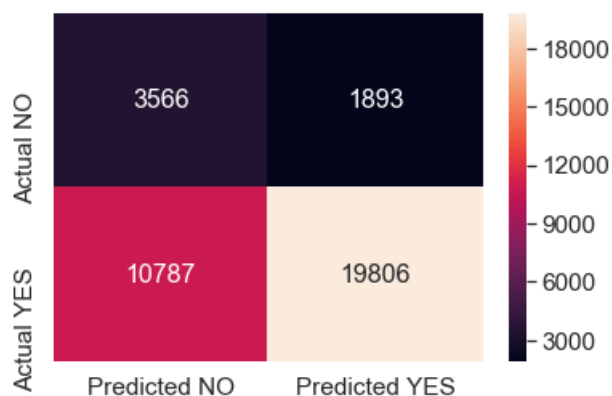
CPU times: user 170 ms, sys: 9.56 ms, total: 179 ms
Wall time: 198 ms



In [112]:

```
%%time
get_confusion_matrix(lr,X_test_tfidf2v,y_test)
```

CPU times: user 144 ms, sys: 7.87 ms, total: 152 ms
Wall time: 170 ms



2.5 Logistic Regression with added Features `Set 5`

[Task-2] Apply Logistic Regression on the below feature set Set 5 by finding the best hyper parameter as suggested in step 2 and step 3. Consider these set of features Set 5:

step 3. Consider these set of features Set 5 :

```
school_state : categorical data
clean_categories : categorical data
clean_subcategories : categorical data
project_grade_category : categorical data
teacher_prefix : categorical data
quantity : numerical data
teacher_number_of_previously_posted_projects : numerical data
price : numerical data
sentiment score's of each of the essay : numerical data
number of words in the title : numerical data
number of words in the combine essays : numerical data
```

And apply the Logistic regression on these features by finding the best hyper paramter as suggested in step 2 and step 3

In [83]:

```
# 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 [84]:

```
# Collect sentiment score
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

sid = SentimentIntensityAnalyzer()

# Process train data
# the sentiment score for each sentence/review is stored in these lists
sentiment_score_essay_pos_train = []
sentiment_score_essay_neg_train = []
sentiment_score_essay_compound_train = [];
for sentence in tqdm(X_train_real['clean_essays']): # for each review/sentence
    ss = sid.polarity_scores(sentence)
    sentiment_score_essay_pos_train.append(ss['pos'])
    sentiment_score_essay_neg_train.append(ss['neg'])
    sentiment_score_essay_compound_train.append(ss['compound'])

sentiment_score_essay_pos_train = np.array(sentiment_score_essay_pos_train)
sentiment_score_essay_neg_train = np.array(sentiment_score_essay_neg_train)
sentiment_score_essay_compound_train = np.array(sentiment_score_essay_compound_train)

print(len(sentiment_score_essay_pos_train))
print(len(sentiment_score_essay_neg_train))
print(len(sentiment_score_essay_compound_train))

# Process cv data
# the sentiment score for each sentence/review is stored in these lists
sentiment_score_essay_pos_cv = []
sentiment_score_essay_neg_cv = []
sentiment_score_essay_compound_cv = [];
for sentence in tqdm(X_cv_real['clean_essays']): # for each review/sentence
    ss = sid.polarity_scores(sentence)
    sentiment_score_essay_pos_cv.append(ss['pos'])
    sentiment_score_essay_neg_cv.append(ss['neg'])
    sentiment_score_essay_compound_cv.append(ss['compound'])

sentiment_score_essay_pos_cv = np.array(sentiment_score_essay_pos_cv)
sentiment_score_essay_neg_cv = np.array(sentiment_score_essay_neg_cv)
sentiment_score_essay_compound_cv = np.array(sentiment_score_essay_compound_cv)
```

```

print(len(sentiment_score_essay_pos_cv))
print(len(sentiment_score_essay_neg_cv))
print(len(sentiment_score_essay_compound_cv))

# Process test data
# the sentiment score for each sentence/review is stored in these lists
sentiment_score_essay_pos_test = []
sentiment_score_essay_neg_test = []
sentiment_score_essay_compound_test = [];
for sentence in tqdm(X_test_real['clean_essays']): # for each review/sentence
    ss = sid.polarity_scores(sentence)
    sentiment_score_essay_pos_test.append(ss['pos'])
    sentiment_score_essay_neg_test.append(ss['neg'])
    sentiment_score_essay_compound_test.append(ss['compound'])

sentiment_score_essay_pos_test = np.array(sentiment_score_essay_pos_test)
sentiment_score_essay_neg_test = np.array(sentiment_score_essay_neg_test)
sentiment_score_essay_compound_test = np.array(sentiment_score_essay_compound_test)

print(len(sentiment_score_essay_pos_test))
print(len(sentiment_score_essay_neg_test))
print(len(sentiment_score_essay_compound_test))

```

```

100%|██████████| 49041/49041 [01:44<00:00, 470.73it/s]
0%|          | 64/24155 [00:00<00:38, 632.95it/s]

```

```

49041
49041
49041

```

```

100%|██████████| 24155/24155 [00:37<00:00, 638.32it/s]
0%|          | 67/36052 [00:00<00:54, 661.92it/s]

```

```

24155
24155
24155

```

```

100%|██████████| 36052/36052 [00:55<00:00, 648.10it/s]

```

```

36052
36052
36052

```

Number of words in the title for each record

In [85]:

```

title_no_of_words_train = []
for title in tqdm(X_train_real['clean_titles']):
    title_no_of_words_train.append(len(title.split()))

title_no_of_words_train = np.array(title_no_of_words_train)

title_no_of_words_cv = []
for title in tqdm(X_cv_real['clean_titles']):
    title_no_of_words_cv.append(len(title.split()))

title_no_of_words_cv = np.array(title_no_of_words_cv)

title_no_of_words_test = []
for title in tqdm(X_test_real['clean_titles']):
    title_no_of_words_test.append(len(title.split()))

title_no_of_words_test = np.array(title_no_of_words_test)

print(len(title_no_of_words_train))
print(len(title_no_of_words_cv))

```

```
print(len(title_no_of_words_test))
```

```
100%|██████████| 49041/49041 [00:00<00:00, 887379.80it/s]
100%|██████████| 24155/24155 [00:00<00:00, 959370.98it/s]
100%|██████████| 36052/36052 [00:00<00:00, 1095905.55it/s]
```

```
49041
24155
36052
```

Number of words in the essay for every record

In [86]:

```
essay_no_of_words_train = []
for essay in tqdm(X_train_real['clean_essays']):
    essay_no_of_words_train.append(len(essay.split()))

essay_no_of_words_train = np.array(essay_no_of_words_train)

essay_no_of_words_cv = []
for essay in tqdm(X_cv_real['clean_essays']):
    essay_no_of_words_cv.append(len(essay.split()))

essay_no_of_words_cv = np.array(essay_no_of_words_cv)

essay_no_of_words_test = []
for essay in tqdm(X_test_real['clean_essays']):
    essay_no_of_words_test.append(len(essay.split()))

essay_no_of_words_test = np.array(essay_no_of_words_test)

print(len(essay_no_of_words_train))
print(len(essay_no_of_words_cv))
print(len(essay_no_of_words_test))
```

```
100%|██████████| 49041/49041 [00:00<00:00, 114604.32it/s]
100%|██████████| 24155/24155 [00:00<00:00, 112209.39it/s]
100%|██████████| 36052/36052 [00:00<00:00, 115146.51it/s]
```

```
49041
24155
36052
```

Finally merge all data

In [88]:

```
X_train_without_text = hstack((X_train_school_state_one, X_train_clean_categories_one,
X_train_clean_subcategories_one, X_train_project_grade_category_one, X_train_teacher_prefix_one, X
_train_quantity_norm, X_train_teacher_number_of_previously_posted_projects_norm,
X_train_price_norm, title_no_of_words_train.reshape(-1,1), essay_no_of_words_train.reshape(-1, 1),
sentiment_score_essay_pos_train.reshape(-1, 1), sentiment_score_essay_neg_train.reshape(-1, 1), sen
timent_score_essay_compound_train.reshape(-1, 1))).tocsr()
X_cv_without_text = hstack((X_cv_school_state_one, X_cv_clean_categories_one,
X_cv_clean_subcategories_one, X_cv_project_grade_category_one, X_cv_clean_teacher_prefix_one,
X_cv_quantity_norm, X_cv_teacher_number_of_previously_posted_projects_norm, X_cv_price_norm,
title_no_of_words_cv.reshape(-1,1), essay_no_of_words_cv.reshape(-1, 1),
sentiment_score_essay_pos_cv.reshape(-1, 1), sentiment_score_essay_neg_cv.reshape(-1, 1), sentiment
_score_essay_compound_cv.reshape(-1, 1))).tocsr()
X_test_without_text = hstack((X_test_school_state_one, X_test_clean_categories_one,
X_test_clean_subcategories_one, X_test_project_grade_category_one, X_test_clean_teacher_prefix_one
, X_test_quantity_norm, X_test_teacher_number_of_previously_posted_projects_norm,
X_test_price_norm, title_no_of_words_test.reshape(-1,1), essay_no_of_words_test.reshape(-1, 1), sen
```

```
timent_score_essay_pos_test.reshape(-1, 1), sentiment_score_essay_neg_test.reshape(-1, 1), sentiment_score_essay_compound_test.reshape(-1, 1))).tocsr()
```

```
print('*'*100)
print('Shape Without Text')
print(X_train_without_text.shape)
print(X_cv_without_text.shape)
print(X_test_without_text.shape)
```

Shape Without Text

```
(49041, 108)
(24155, 108)
(36052, 108)
```

In [89]:

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

import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import learning_curve, GridSearchCV

"""
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.

"""

tuned_parameters = [{'C': [10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}]
lr=GridSearchCV(LogisticRegression(class_weight='balanced'), tuned_parameters, cv=3,
scoring='roc_auc', return_train_score=True)
lr.fit(X_train_without_text, y_train);

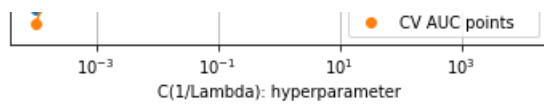
train_auc= lr.cv_results_['mean_train_score']
train_auc_std= lr.cv_results_['std_train_score']
cv_auc = lr.cv_results_['mean_test_score']
cv_auc_std= lr.cv_results_['std_test_score']

plt.plot(tuned_parameters[0]['C'], train_auc, label='Train AUC')
plt.plot(tuned_parameters[0]['C'], cv_auc, label='CV AUC')

plt.scatter(tuned_parameters[0]['C'], train_auc, label='Train AUC points')
plt.scatter(tuned_parameters[0]['C'], cv_auc, label='CV AUC points')
plt.xscale('log')# we take the log in the x axis

plt.legend()
plt.xlabel("C(1/Lambda): hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```





CPU times: user 12.1 s, sys: 355 ms, total: 12.4 s
Wall time: 13.3 s

In [90]:

```
# 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_c = 10**-2
```

In [91]:

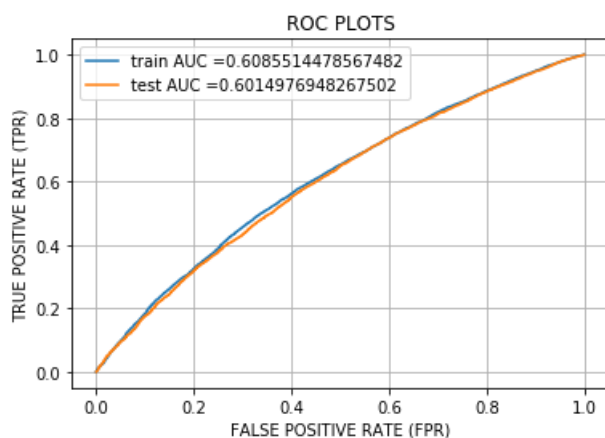
```
%%time
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

lr = LogisticRegression(C=best_c,class_weight='balanced');
lr.fit(X_train_without_text ,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 = lr.predict_proba(X_train_without_text)[: ,1]
y_test_pred = lr.predict_proba(X_test_without_text)[: ,1]

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()
```

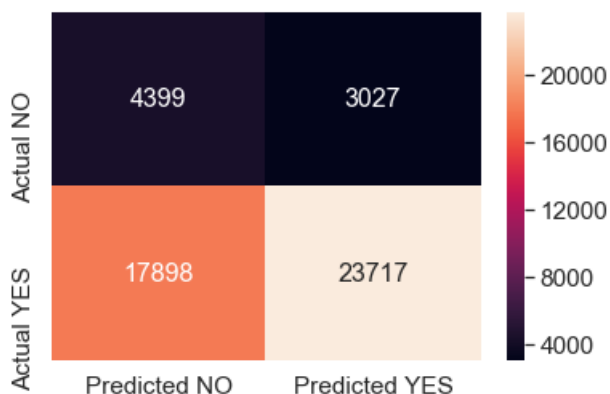


CPU times: user 451 ms, sys: 11.8 ms, total: 463 ms
Wall time: 496 ms

In [92]:

```
%%time
get_confusion_matrix(lr,X_train_without_text,y_train)
```

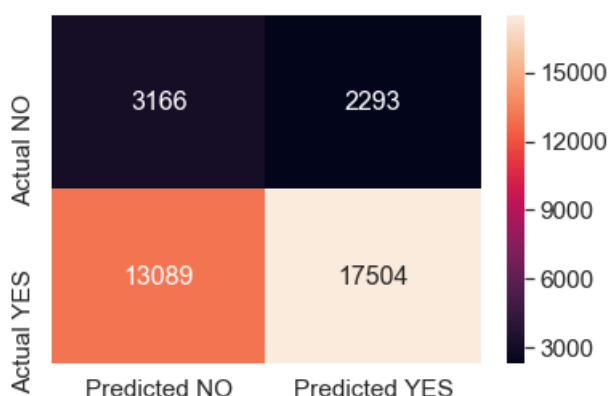
CPU times: user 108 ms, sys: 11.2 ms, total: 119 ms
Wall time: 121 ms



In [93]:

```
%%time  
get_confusion_matrix(lr,X_test_without_text,y_test)
```

CPU times: user 74.6 ms, sys: 2.76 ms, total: 77.3 ms
Wall time: 77.9 ms



3. Conclusions

In [94]:

```
# Please compare all your models using Prettytable library  
# 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(C=1/Lambda)", "AUC"]  
x.add_row(["Bag of Words", "LogisticRegression", 10**-3, 0.69])  
x.add_row(["TFIDF", "LogisticRegression", 10**-2, 0.67])  
x.add_row(["AVG W2V", "LogisticRegression", 10**-2, 0.70])  
x.add_row(["TFIDF W2V", "LogisticRegression", 10**-2, 0.70])  
x.add_row(["Without Text Data", "LogisticRegression", 10**-2, 0.60])  
  
print(x)
```

Vectorizer	Model	Hyperparameter(C=1/Lambda)	AUC
Bag of Words	LogisticRegression	0.001	0.69
TFIDF	LogisticRegression	0.01	0.67
AVG W2V	LogisticRegression	0.01	0.7
TFIDF W2V	LogisticRegression	0.01	0.7
Without Text Data	LogisticRegression	0.01	0.6

Without Text Data LogisticRegression	0.01	0.6
+-----+-----+-----+		

Observations

1. Model with Text Data performed better than features with only numerical and categorical data.

In []: