DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

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

- Literature & Writing, Social Sciences

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

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

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

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

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

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

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

Notes on the Essay Data

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

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

your neighborhood, and your school are all helpful."

- __project_essay_2:__ "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

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

In [1]:

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

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

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

from tqdm import tqdm
import os

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

1.1 Reading Data

In [2]:

```
project_data = pd.read_csv(r'C:\Users\utsav94\Desktop\train_data.csv')
resource_data = pd.read_csv(r'C:\Users\utsav94\Desktop\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]:

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

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

Out[4]:

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

1.2 preprocessing of `project_subject_categories`

In [5]:

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

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

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [6]:

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

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

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

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

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

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

1.3 Text preprocessing

In [7]:

```
# 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 [8]:

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

Out[8]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Gra
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Gra

In [9]:

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

In [10]:

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

```
My students are English learners that are working on English as their second or third languages. W
e are a melting pot of refugees, immigrants, and native-born Americans bringing the gift of langua
ge to our school. \r\n\r\n We have over 24 languages represented in our English Learner program wi
th students at every level of mastery.  We also have over 40 countries represented with the
families within our school.  Each student brings a wealth of knowledge and experiences to us that
open our eyes to new cultures, beliefs, and respect.\"The limits of your language are the limits o
f your world.\"-Ludwig Wittgenstein  Our English learner's have a strong support system at home th
at begs for more resources.  Many times our parents are learning to read and speak English along s
ide of their children.  Sometimes this creates barriers for parents to be able to help their child
learn phonetics, letter recognition, and other reading skills.\r\n\r\nBy providing these dvd's and
players, students are able to continue their mastery of the English language even if no one at hom
e is able to assist.  All families with students within the Level 1 proficiency status, will be a
offered to be a part of this program.  These educational videos will be specially chosen by the En
glish Learner Teacher and will be sent home regularly to watch.  The videos are to help the child
develop early reading skills.\r\n\r\nParents that do not have access to a dvd player will have the
opportunity to check out a dvd player to use for the year.  The plan is to use these videos and ed
ucational dvd's for the years to come for other EL students.\r\nnannan
==================================================
The 51 fifth grade students that will cycle through my classroom this year all love learning, at l
east most of the time. At our school, 97.3% of the students receive free or reduced price lunch. O
f the 560 students, 97.3% are minority students. \r\nThe school has a vibrant community that loves
to get together and celebrate. Around Halloween there is a whole school parade to show off the bea
utiful costumes that students wear. On Cinco de Mayo we put on a big festival with crafts made by
the students, dances, and games. At the end of the year the school hosts a carnival to celebrate t
he hard work put in during the school year, with a dunk tank being the most popular activity.My st
udents will use these five brightly colored Hokki stools in place of regular, stationary, 4-legged
chairs. As I will only have a total of ten in the classroom and not enough for each student to hav
e an individual one, they will be used in a variety of ways. During independent reading time they
will be used as special chairs students will each use on occasion. I will utilize them in place of
chairs at my small group tables during math and reading times. The rest of the day they will be us
ed by the students who need the highest amount of movement in their life in order to stay focused
on school.\r\n\r\nWhenever asked what the classroom is missing, my students always say more Hokki
Stools. They can't get their fill of the 5 stools we already have. When the students are sitting i
n group with me on the Hokki Stools, they are always moving, but at the same time doing their
work. Anytime the students get to pick where they can sit, the Hokki Stools are the first to be ta
ken. There are always students who head over to the kidney table to get one of the stools who are
disappointed as there are not enough of them. \r\n\r\nWe ask a lot of students to sit for 7 hours
a day. The Hokki stools will be a compromise that allow my students to do desk work and move at th
e same time. These stools will help students to meet their 60 minutes a day of movement by
allowing them to activate their core muscles for balance while they sit. For many of my students,
these chairs will take away the barrier that exists in schools for a child who can't sit
still.nannan
==================================================
How do you remember your days of school? Was it in a sterile environment with plain walls, rows of
desks, and a teacher in front of the room? A typical day in our room is nothing like that. I work
hard to create a warm inviting themed room for my students look forward to coming to each day.\r\n
\r\nMy class is made up of 28 wonderfully unique boys and girls of mixed races in
Arkansas.\r\nThey attend a Title I school, which means there is a high enough percentage of free a
nd reduced-price lunch to qualify. Our school is an \"open classroom\" concept, which is very uniq
ue as there are no walls separating the classrooms. These 9 and 10 year-old students are very eage
r learners; they are like sponges, absorbing all the information and experiences and keep on wanti
ng more.With these resources such as the comfy red throw pillows and the whimsical nautical hangin
g decor and the blue fish nets, I will be able to help create the mood in our classroom setting to
be one of a themed nautical environment. Creating a classroom environment is very important in the
success in each and every child's education. The nautical photo props will be used with each child
as they step foot into our classroom for the first time on Meet the Teacher evening. I'll take pic
tures of each child with them, have them developed, and then hung in our classroom ready for their
first day of 4th grade.  This kind gesture will set the tone before even the first day of school!
The nautical thank you cards will be used throughout the year by the students as they create thank
you cards to their team groups.\r\n\r\nYour generous donations will help me to help make our
classroom a fun, inviting, learning environment from day one.\r\n\r\nIt costs lost of money out of
my own pocket on resources to get our classroom ready. Please consider helping with this project t
o make our new school year a very successful one. Thank you!nannan
==================================================
My kindergarten students have varied disabilities ranging from speech and language delays, cogniti
ve delays, gross/fine motor delays, to autism. They are eager beavers and always strive to work th
eir hardest working past their limitations. \r\n\r\nThe materials we have are the ones I seek out
```

```
for my students. I teach in a Title I school where most of the students receive free or reduced pr
ice lunch.  Despite their disabilities and limitations, my students love coming to school and come
eager to learn and explore.Have you ever felt like you had ants in your pants and you needed to gr
oove and move as you were in a meeting? This is how my kids feel all the time. The want to be able
to move as they learn or so they say.Wobble chairs are the answer and I love then because they dev
elop their core, which enhances gross motor and in Turn fine motor skills. \r\nThey also want to l
earn through games, my kids don't want to sit and do worksheets. They want to learn to count by ju
mping and playing. Physical engagement is the key to our success. The number toss and color and sh
ape mats can make that happen. My students will forget they are doing work and just have the fun a
6 year old deserves.nannan
==================================================
The mediocre teacher tells. The good teacher explains. The superior teacher demonstrates. The grea
t teacher inspires. -William A. Ward\r\n\r\nMy school has 803 students which is makeup is 97.6% Af
rican-American, making up the largest segment of the student body. A typical school in Dallas is m
ade up of 23.2% African-American students. Most of the students are on free or reduced lunch. We a
ren't receiving doctors, lawyers, or engineers children from rich backgrounds or neighborhoods. As
an educator I am inspiring minds of young children and we focus not only on academics but one smar
t, effective, efficient, and disciplined students with good character.In our classroom we can util
ize the Bluetooth for swift transitions during class. I use a speaker which doesn't amplify the so
und enough to receive the message. Due to the volume of my speaker my students can't hear videos o
r books clearly and it isn't making the lessons as meaningful. But with the bluetooth speaker my s
tudents will be able to hear and I can stop, pause and replay it at any time.\r\nThe cart will all
ow me to have more room for storage of things that are needed for the day and has an extra part to
it I can use.  The table top chart has all of the letter, words and pictures for students to learn
about different letters and it is more accessible.nannan
==================================================
```

In [11]:

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

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

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

In [12]:

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

```
My kindergarten students have varied disabilities ranging from speech and language delays, cogniti
ve delays, gross/fine motor delays, to autism. They are eager beavers and always strive to work th
eir hardest working past their limitations. \r\n\r\nThe materials we have are the ones I seek out
for my students. I teach in a Title I school where most of the students receive free or reduced pr
ice lunch.  Despite their disabilities and limitations, my students love coming to school and come
eager to learn and explore.Have you ever felt like you had ants in your pants and you needed to gr
oove and move as you were in a meeting? This is how my kids feel all the time. The want to be able
to move as they learn or so they say.Wobble chairs are the answer and I love then because they dev
elop their core, which enhances gross motor and in Turn fine motor skills. \r\nThey also want to l
earn through games, my kids do not want to sit and do worksheets. They want to learn to count by j
umping and playing. Physical engagement is the key to our success. The number toss and color and s
hape mats can make that happen. My students will forget they are doing work and just have the fun
a 6 year old deserves.nannan
==================================================
```

In [13]:

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

```
My kindergarten students have varied disabilities ranging from speech and language delays, cogniti
ve delays, gross/fine motor delays, to autism. They are eager beavers and always strive to work th
eir hardest working past their limitations.     The materials we have are the ones I seek out for
my students. I teach in a Title I school where most of the students receive free or reduced price
lunch.  Despite their disabilities and limitations, my students love coming to school and come eag
er to learn and explore.Have you ever felt like you had ants in your pants and you needed to groov
e and move as you were in a meeting? This is how my kids feel all the time. The want to be able to
move as they learn or so they say.Wobble chairs are the answer and I love then because they develo
p their core, which enhances gross motor and in Turn fine motor skills.   They also want to learn t
hrough games, my kids do not want to sit and do worksheets. They want to learn to count by jumping
and playing. Physical engagement is the key to our success. The number toss and color and shape ma
ts can make that happen. My students will forget they are doing work and just have the fun a 6 yea
r old deserves.nannan
```

In [14]:

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

```
My kindergarten students have varied disabilities ranging from speech and language delays cognitiv
e delays gross fine motor delays to autism They are eager beavers and always strive to work their
hardest working past their limitations The materials we have are the ones I seek out for my studen
ts I teach in a Title I school where most of the students receive free or reduced price lunch
Despite their disabilities and limitations my students love coming to school and come eager to lea
rn and explore Have you ever felt like you had ants in your pants and you needed to groove and mov
e as you were in a meeting This is how my kids feel all the time The want to be able to move as th
ey learn or so they say Wobble chairs are the answer and I love then because they develop their co
re which enhances gross motor and in Turn fine motor skills They also want to learn through games
my kids do not want to sit and do worksheets They want to learn to count by jumping and playing Ph
ysical engagement is the key to our success The number toss and color and shape mats can make that
happen My students will forget they are doing work and just have the fun a 6 year old deserves nan
nan
```

In [15]:

```
# 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', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [16]:

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

```
100%|█████████████████████████████████| 109248/109248 [02:00<00:00, 904.70it/s]
```

In [17]:

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

Out[17]:

```
'my kindergarten students varied disabilities ranging speech language delays cognitive delays gros
s fine motor delays autism they eager beavers always strive work hardest working past limitations
the materials ones i seek students i teach title i school students receive free reduced price lunc
h despite disabilities limitations students love coming school come eager learn explore have ever
felt like ants pants needed groove move meeting this kids feel time the want able move learn say w
obble chairs answer i love develop core enhances gross motor turn fine motor skills they also want
learn games kids not want sit worksheets they want learn count jumping playing physical engagement
key success the number toss color shape mats make happen my students forget work fun 6 year old de
serves nannan'
```

1.4 Preprocessing of `project_title`

In [18]:

```
# similarly you can preprocess the titles also
# similarly you can preprocess the titles also
print(project_data['project_title'].values[0])
print("="*50)
print(project_data['project_title'].values[150])
print("="*50)
print(project_data['project_title'].values[1000])
print("="*50)
print(project_data['project_title'].values[20000])
print("="*50)
print(project_data['project_title'].values[99999])
print("="*50)
```

```
Educational Support for English Learners at Home
==================================================
More Movement with Hokki Stools
==================================================
Sailing Into a Super 4th Grade Year
==================================================
We Need To Move It While We Input It!
==================================================
Inspiring Minds by Enhancing the Educational Experience
==================================================
```

In [19]:

```
sent_1 = decontracted(project_data['project_title'].values[500])
print(sent_1)
print("="*50)

preprocessed_titles = []
# tqdm is for printing the status bar
for sentance_1 in tqdm(project_data['project_title'].values):
    sent_1 = decontracted(sentance_1)
    sent_1 = sent_1.replace('\\r', ' ')
    sent_1 = sent_1.replace('\\r', ' ')
    sent_1 = sent_1.replace('\\"', ' ')
    sent_1 = sent_1.replace('\\n', ' ')
    sent_1 = re.sub('[^A-Za-z0-9]+', ' ', sent_1)
    # https://gist.github.com/sebleier/554280
    sent_1 = ' '.join(e for e in sent_1.split() if e not in stopwords)
    preprocessed_titles.append(sent_1.lower().strip())
```

```
Classroom Chromebooks for College Bound Seniors!
==================================================
```

```
100%|██████████████████████████████| 109248/109248 [00:05<00:00, 20057.01it/s]
```

In [20]:

```
preprocessed_titles[13143]
```

Out[20]:

```
'engaging stem laboratory activities'
```

1.5 Preparing data for models

In [21]:

```
project_data.columns
```

Out[21]:

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

we are going to consider

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

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

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

1.5.1 Vectorizing Categorical data

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

In [22]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True
)
categories_one_hot = vectorizer.fit_transform(project_data['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",categories_one_hot.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig  (109248, 9)
```

In [23]:

```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=
True)
sub_categories_one_hot = vectorizer.fit_transform(project_data['clean_subcategories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",sub_categories_one_hot.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig  (109248, 30)
```

In [24]:

```
# you can do the similar thing with state, teacher_prefix and project_grade_category also
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [25]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10)
text_bow = vectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",text_bow.shape)
```

```
Shape of matrix after one hot encodig  (109248, 16623)
```

In [26]:

```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
```

1.5.2.2 TFIDF vectorizer

In [27]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
text_tfidf = vectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",text_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (109248, 16623)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [28]:

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

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

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

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

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

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

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

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


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

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


'''
```

Out[28]:

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

In [29]:

```
# 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
# 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 [30]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    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.append(vector)

print(len(avg_w2v_vectors))
print(len(avg_w2v_vectors[0]))
```

```
100%|████████████████████████████████| 109248/109248 [01:01<00:00, 1773.92it/s]
```

```
109248
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [31]:

```
# 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 [32]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the 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.append(vector)

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

```
100%|████████████████████████████████| 109248/109248 [06:31<00:00, 279.30it/s]
```

```
109248
300
```

In [33]:

```
# Similarly you can vectorize for title also
```

1.5.3 Vectorizing Numerical features

In [34]:

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

In [35]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-
learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.
73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)

price_scalar = StandardScaler()
price_scalar.fit(project_data['price'].values.reshape(-1,1)) # finding the mean and standard
deviation of this data
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
price_standardized = price_scalar.transform(project_data['price'].values.reshape(-1, 1))
```

```
Mean : 298.1193425966608, Standard deviation : 367.49634838483496
```

In [36]:

```
price_standardized
```

Out[36]:

```
array([[-0.3905327 ],
       [ 0.00239637],
       [ 0.59519138],
       ...,
       [-0.15825829],
       [-0.61243967],
       [-0.51216657]])
```

1.5.4 Merging all the above features

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

In [37]:

```
print(categories_one_hot.shape)
print(sub_categories_one_hot.shape)
print(text_bow.shape)
print(price_standardized.shape)
```

```
(109248, 9)
(109248, 30)
(109248, 16623)
(109248, 1)
```

In [38]:

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

Out[38]:

```
(109248, 16663)
```

In [39]:

```
ase = project_data['project_is_approved']
project_data['teacher_prefix'] = project_data['teacher_prefix'].replace(np.nan, 'teacher')
project_data['project_grade_category'] = project_data['project_grade_category'].str.replace(' ','')
project_data['project_grade_category']
project_data['project_grade_category'] = project_data['project_grade_category'].str.replace("-", "_
")
```

In [40]:

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

#https://stackoverflow.com/questions/49984905/count-number-of-words-per-row/49984998
project_data['essay_count']=project_data['preprocessed_essays'].str.split().str.len()
project_data['title_count']=project_data['preprocessed_titles'].str.split().str.len()

#https://www.geeksforgeeks.org/python-textblob-sentiment-method/
#https://textblob.readthedocs.io/en/dev/quickstart.html#quickstart
from textblob import TextBlob
project_data['essay_sentiment'] = [ TextBlob(tb).sentiment.polarity for tb in project_data['essay']
]

project_data
```

Out[40]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetin
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	2016-08-31 12:03:56
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY	2016-10-06 21:16:17
4	172407	p104768	be1f7507a41f8479dc06f047086a39ec	Mrs.	TX	2016-07-11 01:10:09
5	141660	p154343	a50a390e8327a95b77b9e495b58b9a6e	Mrs.	FL	2017-04-08 22:40:43

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetim
6	21147	p099819	9b40170bfa65e399981717ee8731efc3	Mrs.	CT	2017-02-17 19:58:56
7	94142	p092424	5bfd3d12fae3d2fe88684bbac570c9d2	Ms.	GA	2016-09-01 00:02:15
8	112489	p045029	487448f5226005d08d36bdd75f095b31	Mrs.	SC	2016-09-25 17:00:26
9	158561	p001713	140eeac1885c820ad5592a409a3a8994	Ms.	NC	2016-11-17 18:18:56
10	43184	p040307	363788b51d40d978fe276bcb1f8a2b35	Mrs.	CA	2017-01-04 16:40:30
11	127083	p251806	4ba7c721133ef651ca54a03551746708	Ms.	CA	2016-11-14 22:57:28
12	19090	p051126	5e52c92b7e3c472aad247a239d345543	Mrs.	NY	2016-05-23 15:46:02
13	15126	p003874	178f6ae765cd4e0fb143a77c47fd65e2	Mrs.	OK	2016-10-17 09:49:27
14	62232	p233127	424819801de22a60bba7d0f4354d0258	Ms.	MA	2017-02-14 16:29:10
15	67303	p132832	bb6d6d054824fa01576ab38dfa2be160	Ms.	TX	2016-10-05 21:05:38
16	127215	p174627	4ad7e280fddff889e1355cc9f29c3b89	Mrs.	FL	2017-01-18 10:59:05
17	157771	p152491	e39abda057354c979c5b075cffbe5f88	Ms.	NV	2016-11-23 17:14:17
18	122186	p196421	fcd9b003fc1891383f340a89da02a1a6	Mrs.	GA	2016-08-28 15:04:42
19	146331	p058343	8e07a98deb1bc74c75b97521e05b1691	Ms.	OH	2016-08-06 13:05:20

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetim
20	75560	p052326	e0c1aad1f71badeff703fadc15f57680	Mrs.	PA	2016-10-07 18:27:02
21	132078	p187097	2d4a4d2d774e5c2fdd25b2ba0e7341f8	Mrs.	NC	2016-05-17 19:45:13
22	84810	p165540	30f08fbe02eba5453c4ce2e857e88eb4	Ms.	CA	2016-09-01 10:09:15
23	8636	p219330	258ef2e6ab5ce007ac6764ce15d261ba	Mr.	AL	2017-01-10 11:41:06
24	21478	p126524	74f8690562c44fc88f65f845b9fe61d0	Mrs.	FL	2017-03-31 12:34:44
25	20142	p009037	b8bf3507cee960d5fedcb27719df2d59	Mrs.	AL	2017-03-09 15:36:20
26	33903	p040091	7a0a5de5ed94e7036946b1ac3eaa99d0	Ms.	TX	2016-09-18 22:10:40
27	1156	p161033	efdc3cf14d136473c9f62becc00d4cec	Teacher	LA	2016-11-06 16:02:31
28	35430	p085706	22c8184c4660f1c589bea061d14b7f35	Mrs.	GA	2017-01-27 12:34:59
29	22088	p032018	45f16a103f1e00b7439861d4e0728a59	Mrs.	VA	2016-07-15 12:58:40
...	...	...	...	...	...	...
109218	127181	p077978	91f5c69bf72c82edb9bc1f55596d8d95	Mrs.	IL	2017-01-10 14:08:28
109219	65838	p042022	9a6784108c76576565f46446594f99c4	Teacher	FL	2016-07-26 22:43:52
109220	21062	p064087	19c622a38a0cd76c2e9dbcc40541fabd	Mrs.	WI	2016-09-18 13:15:13

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetim
109221	81490	p117254	031e299278ac511616b2950fc1312a55	Teacher	NY	2016-07-03 23:09:29
109222	69138	p152194	6f6e951e435aa9dc966091945414bcc4	Ms.	NC	2016-12-01 20:29:04
109223	5110	p041136	6db62616b4ef6efc2310088f7ea0ae14	Ms.	GA	2017-02-15 14:07:07
109224	109630	p257774	651866d8215616f65934aafcbee21bf5	Ms.	NY	2016-05-23 20:36:51
109225	177841	p079425	c628dff071aa8028b08a5d4972bef2a1	Mrs.	NC	2016-11-14 21:04:43
109226	65359	p085810	1d286ff10ee3982b2b47813f1e415ef2	Ms.	CA	2016-08-12 09:19:22
109227	55643	p146149	e15cd063caa1ce11a45f2179535105f2	Mrs.	NY	2016-10-19 10:10:04
109228	103666	p191845	d0603199630760d8d0eb003108208998	Mrs.	LA	2016-10-14 18:05:17
109229	121219	p055363	523f95270c6aec82bee90e3931ceeeca	Mrs.	CO	2016-09-06 23:19:17
109230	117282	p235512	ee59900af64d9244487e7ed87d0bc423	Ms.	NY	2016-08-09 21:06:33
109231	170085	p248898	9d7a4dae637d1a170778e2db1515e574	Mrs.	AZ	2016-09-17 09:58:59
109232	36083	p204774	c116af7435274872bea9ff123a69cf6a	Mrs.	MD	2017-03-14 19:59:52
109233	155847	p120664	b90258ab009b84e0dc11a7186d597141	Ms.	AZ	2016-12-21 16:36:26
109234	52918	p057638	dd68d9fbae85933c0173c13f66291cbe	Ms.	NY	2017-03-29 20:06:10

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetim
109235	69971	p105083	9636fcacbf65eb393133a94c83c4a0d4	Mrs.	TX	2017-01-07 14:50:08
109236	120581	p254202	2950019dd34581dbcddcae683e74207a	Mrs.	OH	2016-08-14 08:27:24
109237	115336	p056813	07fd2c09f8dfcc74dbb161e1ec3df1fe	Mrs.	IN	2016-05-05 13:03:58
109238	32628	p143363	5b42211690ca8418c7c839436d0b7e49	Mrs.	WI	2016-08-01 21:17:33
109239	156548	p103958	8b9a9dc5bd4aa0301b0ff416e2ed29f6	Mrs.	MN	2016-08-15 17:01:00
109240	93971	p257729	58c112dcb2f1634a4d4236bf0dcdcb31	Mrs.	MD	2016-08-25 13:09:19
109241	36517	p180358	3e5c98480f4f39d465837b2955df6ae0	Mrs.	MD	2016-06-24 11:48:12
109242	34811	p080323	fe10e79b7aeb570dfac87eeea7e9a8f1	Mrs.	SC	2017-03-09 20:00:33
109243	38267	p048540	fadf72d6cd83ce6074f9be78a6fcd374	Mr.	MO	2016-06-17 12:02:31
109244	169142	p166281	1984d915cc8b91aa16b4d1e6e39296c6	Ms.	NJ	2017-01-11 12:49:39
109245	143653	p155633	cdbfd04aa041dc6739e9e576b1fb1478	Mrs.	NJ	2016-08-25 17:11:32
109246	164599	p206114	6d5675dbfafa1371f0e2f6f1b716fe2d	Mrs.	NY	2016-07-29 17:53:15

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetin
109247	128381	p191189	ca25d5573f2bd2660f7850a886395927	Ms.	VA	2016-06-29 09:17:01

109248 rows × 25 columns

Computing Sentiment Scores

In [41]:

```
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
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\utsav94\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
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`) `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

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

In [42]:

```
# 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

#https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6

# ============================== loading libraries ===========================================
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn import model_selection
# ============================================================================================

y13 = project_data['project_is_approved']
X_train, X_test, y_train, y_test = train_test_split(project_data, y13, stratify=y13, test_size=0.3)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [43]:

```
# 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

from sklearn.feature_extraction.text import CountVectorizer
vectorizer_train_tr_1 = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False,
binary=True)
categories_one_hot_train = vectorizer_train_tr_1.fit(X_train['clean_categories'].values)
#print(vectorizer_train_tr_1.get_feature_names())
#print("Shape of matrix after one hot encodig ",categories_one_hot_train_tr.shape)

categories_one_hot_train_tr =
categories_one_hot_train.transform(X_train['clean_categories'].values)
#print(vectorizer_train_tr_1.get_feature_names())
print("Shape of matrix after one hot encodig ",categories_one_hot_train_tr.shape)

#feature names encoding for X_test
#from sklearn.feature_extraction.text import CountVectorizer
#vectorizer_test_1 = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, bin
ary=True)
categories_one_hot_test = categories_one_hot_train.transform(X_test['clean_categories'].values)
#print(vectorizer_test_1.get_feature_names())
print("Shape of matrix after one hot encodig ",categories_one_hot_test.shape)



# we use count vectorizer to convert the values into one for X_train
vectorizer_train_tr = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False
, binary=True)
sub_categories_one_hot_train = vectorizer_train_tr.fit(X_train['clean_subcategories'].values)
#print(vectorizer_train_tr.get_feature_names())
#print("Shape of matrix after one hot encodig ",sub_categories_one_hot_train_tr.shape)

sub_categories_one_hot_train_tr =
sub_categories_one_hot_train.transform(X_train['clean_subcategories'].values)
#print(vectorizer_train_tr.get_feature_names())
print("Shape of matrix after one hot encodig ",sub_categories_one_hot_train_tr.shape)

# we use count vectorizer to convert the values into one for X_test
vectorizer_test = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, bi
vectorizer_test = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, bi
nary=True)
sub_categories_one_hot_test = sub_categories_one_hot_train.transform(X_test['clean_subcategories']
.values)
#print(vectorizer_test.get_feature_names())
print("Shape of matrix after one hot encodig ",sub_categories_one_hot_test.shape)


#school state for Xtrain
vectorizer_1_train_tr = CountVectorizer(lowercase=False, binary=True)
vectorizer_1_train_tr.fit(X_train['school_state'].values)
#print(vectorizer_1_train_tr.get_feature_names())
categories_state_1_train = vectorizer_1_train_tr.fit(X_train['school_state'].values)
#print("Shape of matrix after one hot encodig ",categories_state_1_train_tr.shape)

categories_state_1_train_tr = categories_state_1_train.transform(X_train['school_state'].values)
print("Shape of matrix after one hot encodig ",categories_state_1_train_tr.shape)

#school state for Xtest
vectorizer_1_test = CountVectorizer(lowercase=False, binary=True)
vectorizer_1_test.fit(X_test['school_state'].values)
#print(vectorizer_1_test.get_feature_names())
categories_state_1_test = categories_state_1_train.transform(X_test['school_state'].values)
print("Shape of matrix after one hot encodig ",categories_state_1_test.shape)



#Project grade category for X_train
#vectorizer_2_train = CountVectorizer(vocabulary=list(word_dict_134.keys()), lowercase=False, bina
ry=True)
vectorizer_2_train_tr = CountVectorizer(lowercase=False, binary=True)
vectorizer_2_train_tr.fit(X_train['project_grade_category'].values)
#print(vectorizer_2_train_tr.get_feature_names())
categories_grade_train = vectorizer_2_train_tr.fit(X_train['project_grade_category'].values)
#print("Shape of matrix after one hot encodig ",categories_grade_train_tr.shape)

categories_grade_train_tr = categories_grade_train.transform(X_train['project_grade_category'].val
ues)
print("Shape of matrix after one hot encodig ",categories_grade_train_tr.shape)

#Project grade category for X_test
#vectorizer_2_test = CountVectorizer(vocabulary=list(word_dict_134.keys()), lowercase=False, binar
y=True)
#vectorizer_2_test = CountVectorizer(lowercase=False, binary=True)
#vectorizer_2_test.fit(X_test['project_grade_category'].values)
#print(vectorizer_2_test.get_feature_names())
categories_grade_test = categories_grade_train.transform(X_test['project_grade_category'].values)
print("Shape of matrix after one hot encodig ",categories_grade_test.shape)



from string import punctuation

#https://medium.com/@chaimgluck1/have-messy-text-data-clean-it-with-simple-lambda-functions-645918
fcc2fc
#project_data.teacher_prefix = project_data.teacher_prefix.apply(lambda x:
x.translate(string.punctuation))

#https://stackoverflow.com/questions/50443494/error-in-removing-punctuation-float-object-has-no-at
tribute-translate
#project_data['teacher_prefix'] = project_data.fillna({'teacher_prefix':''})

project_data['teacher_prefix'] = project_data['teacher_prefix'].replace(np.nan, 'teacher')

#teacher_prefix for X_train
vectorizer_3_train_tr = CountVectorizer(lowercase=False, binary=True)
vectorizer_3_train_tr.fit(X_train['teacher_prefix'].values)
#print(vectorizer_3_train_tr.get_feature_names())
categories_teacher_prefix_train = vectorizer_3_train_tr.fit(X_train['teacher_prefix'].values)
#print("Shape of matrix after one hot encodig ",categories_teacher_prefix_train_tr.shape)

categories_teacher_prefix_train_tr =
categories_teacher_prefix_train.transform(X_train['teacher_prefix'].values)
print("Shape of matrix after one hot encodig ",categories_teacher_prefix_train_tr.shape)

#teacher_prefix for X_test
#vectorizer_3_test = CountVectorizer(lowercase=False, binary=True)
#vectorizer_3_test.fit(X_test['teacher_prefix'].values)
```

```
#vectorizer_3_test.fit(X_test['teacher_prefix'].values)
#print(vectorizer_3_test.get_feature_names())
categories_teacher_prefix_test = categories_teacher_prefix_train.transform(X_test['teacher_prefix'
].values)
print("Shape of matrix after one hot encodig ",categories_teacher_prefix_test.shape)
```

```
Shape of matrix after one hot encodig  (76473, 9)
Shape of matrix after one hot encodig  (32775, 9)
Shape of matrix after one hot encodig  (76473, 30)
Shape of matrix after one hot encodig  (32775, 30)
Shape of matrix after one hot encodig  (76473, 51)
Shape of matrix after one hot encodig  (32775, 51)
Shape of matrix after one hot encodig  (76473, 4)
Shape of matrix after one hot encodig  (32775, 4)
Shape of matrix after one hot encodig  (76473, 6)
Shape of matrix after one hot encodig  (32775, 6)
```

2.3 Make Data Model Ready: encoding eassay, and project_title

In [44]:

```
# 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

#Vectorizing essays for X_train.

#First fitting the vector
vectorizer_essays_train_tr = CountVectorizer(min_df=10, ngram_range=(2,2), max_features=5000)
text_bow_train = vectorizer_essays_train_tr.fit(X_train['preprocessed_essays'])
#print("Shape of matrix after one hot encodig ",text_bow_train_tr.shape)

#transforming train data
text_bow_train_tr = text_bow_train.transform(X_train['preprocessed_essays'])
print("Shape of matrix after one hot encodig ",text_bow_train_tr.shape)

#transforming text data.
vectorizer_essays_test = CountVectorizer(min_df=10, ngram_range=2)
text_bow_test = text_bow_train.transform(X_test['preprocessed_essays'])
print("Shape of matrix after one hot encodig ",text_bow_test.shape)



# Vectorizing Title for X_train
vectorizer_title_tr_tr = CountVectorizer(min_df=10)
title_bow_tr = vectorizer_title_tr_tr.fit(X_train['preprocessed_titles'])
#print("Shape of matrix after one hot encodig ",title_bow_tr_tr.shape)

title_bow_tr_tr = title_bow_tr.transform(X_train['preprocessed_titles'])
print("Shape of matrix after one hot encodig ",title_bow_tr_tr.shape)

# Vectorizing Title for X_test
vectorizer_title_test = CountVectorizer(min_df=10)
title_bow_test = title_bow_tr.transform(X_test['preprocessed_titles'])
print("Shape of matrix after one hot encodig ",title_bow_test.shape)



#Vectorizing using tfidf for essays for x_train
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf_essays_train_tr = TfidfVectorizer(min_df=10, ngram_range=(2,2), max_features=5000)
text_tfidf_train = vectorizer_tfidf_essays_train_tr.fit(X_train['preprocessed_essays'])
#print("Shape of matrix after one hot encodig ",text_tfidf_train_tr.shape)

text_tfidf_train_tr = text_tfidf_train.transform(X_train['preprocessed_essays'])
print("Shape of matrix after one hot encodig ",text_tfidf_train_tr.shape)
print("Shape of matrix after one hot encodig ",text_tfidf_train_tr.shape)

#Vectorizing using tfidf for essays for x_test
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf_essays_test = TfidfVectorizer(min_df=10)
text_tfidf_test = text_tfidf_train.transform(X_test['preprocessed_essays'])
print("Shape of matrix after one hot encodig ",text_tfidf_test.shape)




#Vectorizing using tfidf for titles using X_train
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf_tr = TfidfVectorizer(min_df=10)
title_tfidf_train = vectorizer_tfidf_tr.fit(X_train['preprocessed_titles'])
#print("Shape of matrix after one hot encodig ",title_tfidf_train_tr.shape)

title_tfidf_train_tr = title_tfidf_train.transform(X_train['preprocessed_titles'])
print("Shape of matrix after one hot encodig ",title_tfidf_train_tr.shape)

#Vectorizing using tfidf for titles using X_test
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
title_tfidf_test = title_tfidf_train.transform(X_test['preprocessed_titles'])
print("Shape of matrix after one hot encodig ",title_tfidf_test.shape)
```

```
Shape of matrix after one hot encodig  (76473, 5000)
Shape of matrix after one hot encodig  (32775, 5000)
Shape of matrix after one hot encodig  (76473, 2684)
Shape of matrix after one hot encodig  (32775, 2684)
Shape of matrix after one hot encodig  (76473, 5000)
Shape of matrix after one hot encodig  (32775, 5000)
Shape of matrix after one hot encodig  (76473, 2684)
Shape of matrix after one hot encodig  (32775, 2684)
```

In [45]:

```
avg_w2v_vectors_1_train_tr = []
for sentence in tqdm(X_train['preprocessed_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_1_train_tr.append(vector)

print(len(avg_w2v_vectors_1_train_tr))
print(len(avg_w2v_vectors_1_train_tr[0]))



avg_w2v_vectors_1_test = []
for sentence in tqdm(X_test['preprocessed_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_1_test.append(vector)

print(len(avg_w2v_vectors_1_test))
print(len(avg_w2v_vectors_1_test[0]))
```

```
100%|██████████████████████████████████| 76473/76473 [00:02<00:00, 25702.65it/s]
```

```
76473
300
```

```
100%|██████████████████████████████████| 32775/32775 [00:01<00:00, 27650.72it/s]
```

```
32775
300
```

In [46]:

```
# Similarly you can vectorize for title also, first for train
tfidf_model_1_train_tr = TfidfVectorizer()
tfidf_model_1_train_tr.fit(X_train['preprocessed_titles'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary_1 = dict(zip(tfidf_model_1_train_tr.get_feature_names(), list(tfidf_model_1_train_tr.id
f_)))
tfidf_words_1_train_tr = set(tfidf_model_1_train_tr.get_feature_names())
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_1_train_tr = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['preprocessed_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    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_1_train_tr):
            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_1[word]*(sentence.count(word)/len(sentence.split())) # getting the
tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_1_train_tr.append(vector)

print(len(tfidf_w2v_vectors_1_train_tr))
print(len(tfidf_w2v_vectors_1_train_tr[0]))


# Similarly you can vectorize for title also
tfidf_model_1_test = TfidfVectorizer()
tfidf_model_1_test.fit(X_test['preprocessed_titles'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary_2 = dict(zip(tfidf_model_1_test.get_feature_names(), list(tfidf_model_1_test.idf_)))
tfidf_words_1_test = set(tfidf_model_1_test.get_feature_names())
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_1_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['preprocessed_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    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_1_test):
            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_2[word]*(sentence.count(word)/len(sentence.split())) # getting the
tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_1_test.append(vector)

print(len(tfidf_w2v_vectors_1_test))
print(len(tfidf_w2v_vectors_1_test[0]))
```

```
100%|██████████████████████████████████| 76473/76473 [00:06<00:00, 12285.93it/s]
```

```
76473
300
```

```
100%|██████████████████████████████████| 32775/32775 [00:02<00:00, 11246.15it/s]
```

```
32775
300
```

```
300
```

In [47]:

```
avg_w2v_vectors_1_train_tr_essay = []
for sentence in tqdm(X_train['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_1_train_tr_essay.append(vector)

print(len(avg_w2v_vectors_1_train_tr_essay))
print(len(avg_w2v_vectors_1_train_tr_essay[0]))



avg_w2v_vectors_1_test_essay = []
for sentence in tqdm(X_test['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_1_test_essay.append(vector)

print(len(avg_w2v_vectors_1_test_essay))
print(len(avg_w2v_vectors_1_test_essay[0]))
```

```
100%|██████████████████████████████████| 76473/76473 [00:50<00:00, 1506.41it/s]
```

```
76473
300
```

```
100%|██████████████████████████████████| 32775/32775 [00:24<00:00, 1350.67it/s]
```

```
32775
300
```

In [48]:

```
# Similarly you can vectorize for title also, first for train
tfidf_model_1_train_tr_essay = TfidfVectorizer()
tfidf_model_1_train_tr_essay.fit(X_train['preprocessed_essays'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary_1 = dict(zip(tfidf_model_1_train_tr_essay.get_feature_names(),
list(tfidf_model_1_train_tr_essay.idf_)))
tfidf_words_1_train_tr_essay = set(tfidf_model_1_train_tr_essay.get_feature_names())
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_1_train_tr_essay = []; # the avg-w2v for each sentence/review is stored in this
list
for sentence in tqdm(X_train['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    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_1_train_tr_essay):
            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_1_train_tr_essay.append(vector)

print(len(tfidf_w2v_vectors_1_train_tr_essay))
print(len(tfidf_w2v_vectors_1_train_tr_essay[0]))


# Similarly you can vectorize for title also
tfidf_model_1_test_essay = TfidfVectorizer()
tfidf_model_1_test_essay.fit(X_test['preprocessed_essays'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary_2 = dict(zip(tfidf_model_1_test_essay.get_feature_names(),
list(tfidf_model_1_test_essay.idf_)))
tfidf_words_1_test_essay = set(tfidf_model_1_test_essay.get_feature_names())
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_1_test_essay = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    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_1_test_essay):
            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_1_test_essay.append(vector)

print(len(tfidf_w2v_vectors_1_test_essay))
print(len(tfidf_w2v_vectors_1_test_essay[0]))
```

```
100%|███████████████████████████████████| 76473/76473 [06:06<00:00, 208.80it/s]
```

```
76473
300
```

```
100%|███████████████████████████████████| 32775/32775 [02:45<00:00, 198.07it/s]
```

```
32775
300
```

In [49]:

```
from sklearn.preprocessing import StandardScaler

#price scalar and standardized for train
price_scalar_train_tr = StandardScaler()
price_scalar_train_tr.fit(X_train['price'].values.reshape(-1,1)) # finding the mean and standard
deviation of this data
#print(f"Mean : {price_scalar_train_tr.mean_[0]}, Standard deviation :
{np.sqrt(price_scalar_train_tr.var_[0])}")
price_standardized_train_tr = price_scalar_train_tr.transform(X_train['price'].values.reshape(-1, 1
))

#price scalar and standardized for test
price_scalar_test = StandardScaler()
price_scalar_test.fit(X_test['price'].values.reshape(-1,1)) # finding the mean and standard
deviation of this data
#print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")
price_standardized_test = price_scalar_test.transform(X_test['price'].values.reshape(-1, 1))

teacher_number_of_previously_posted_projects_scalar_train_tr = StandardScaler()
teacher_number_of_previously_posted_projects_scalar_train_tr.fit(X_train['teacher_number_of_previou
sly_posted_projects'].values.reshape(-1,1)) # finding the mean and standard deviation of this data
#print(f"Mean : {teacher_number_of_previously_posted_projects_scalar_train_tr.mean_[0]}, Standard
deviation : {np.sqrt(teacher_number_of_previously_posted_projects_scalar_train_tr.var_[0])}")
# Now standardize the data with above maen and variance.
teacher_number_of_previously_posted_projects_standardized_train_tr =
teacher_number_of_previously_posted_projects_scalar_train_tr.transform(X_train['teacher_number_of_p
reviously_posted_projects'].values.reshape(-1, 1))
```

```
teacher_number_of_previously_posted_projects_scalar_test = StandardScaler()
teacher_number_of_previously_posted_projects_scalar_test.fit(X_test['teacher_number_of_previously_p
sted_projects'].values.reshape(-1,1)) # finding the mean and standard deviation of this data
#print(f"Mean : {teacher_number_of_previously_posted_projects_scalar.mean_[0]}, Standard deviation
: {np.sqrt(teacher_number_of_previously_posted_projects_scalar.var_[0])}")
# Now standardize the data with above maen and variance.
teacher_number_of_previously_posted_projects_standardized_test =
teacher_number_of_previously_posted_projects_scalar_test.transform(X_test['teacher_number_of_previo
usly_posted_projects'].values.reshape(-1, 1))


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

#categorical, numerical features + project_title(BOW)
X1_train_tr = hstack((categories_one_hot_train_tr, sub_categories_one_hot_train_tr,
categories_state_1_train_tr, categories_grade_train_tr, categories_teacher_prefix_train_tr,
price_standardized_train_tr, teacher_number_of_previously_posted_projects_standardized_train_tr, t
itle_bow_tr_tr, text_bow_train_tr))


X1_test = hstack((categories_one_hot_test, sub_categories_one_hot_test, categories_state_1_test, c
ategories_grade_test, categories_teacher_prefix_test, price_standardized_test,
teacher_number_of_previously_posted_projects_standardized_test, title_bow_test, text_bow_test))

#categorical, numerical features + project_title(TFIDF)
X2_train_tr = hstack((categories_one_hot_train_tr, sub_categories_one_hot_train_tr,
categories_state_1_train_tr, categories_grade_train_tr, categories_teacher_prefix_train_tr,
price_standardized_train_tr, teacher_number_of_previously_posted_projects_standardized_train_tr, t
itle_tfidf_train_tr, text_tfidf_train_tr))


X2_test = hstack((categories_one_hot_test, sub_categories_one_hot_test, categories_state_1_test, c
ategories_grade_test, categories_teacher_prefix_test, price_standardized_test,
teacher_number_of_previously_posted_projects_standardized_test, title_tfidf_test, text_tfidf_test)
)

#categorical, numerical features + project_title(AVG W2V)
X3_train_tr = hstack((categories_one_hot_train_tr, sub_categories_one_hot_train_tr,
categories_state_1_train_tr, categories_grade_train_tr, categories_teacher_prefix_train_tr,
price_standardized_train_tr, teacher_number_of_previously_posted_projects_standardized_train_tr, a
vg_w2v_vectors_1_train_tr_essay, avg_w2v_vectors_1_train_tr))


X3_test = hstack((categories_one_hot_test, sub_categories_one_hot_test, categories_state_1_test, c
ategories_grade_test, categories_teacher_prefix_test, price_standardized_test,
teacher_number_of_previously_posted_projects_standardized_test, avg_w2v_vectors_1_test_essay, avg_
w2v_vectors_1_test))

#categorical, numerical features + project_title(TFIDF W2V)
X4_train_tr = hstack((categories_one_hot_train_tr, sub_categories_one_hot_train_tr,
categories_state_1_train_tr, categories_grade_train_tr, categories_teacher_prefix_train_tr,
price_standardized_train_tr, teacher_number_of_previously_posted_projects_standardized_train_tr, t
fidf_w2v_vectors_1_train_tr_essay, tfidf_w2v_vectors_1_train_tr))


X4_test = hstack((categories_one_hot_test, sub_categories_one_hot_test, categories_state_1_test, c
ategories_grade_test, categories_teacher_prefix_test, price_standardized_test,
teacher_number_of_previously_posted_projects_standardized_test, tfidf_w2v_vectors_1_test_essay, tf
idf_w2v_vectors_1_test))
```

2.4 Appling 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 instrucations

In [50]:

```
# 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


import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets, neighbors
from matplotlib.colors import ListedColormap

#https://www.ritchieng.com/machine-learning-efficiently-search-tuning-param/

# imports
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import GridSearchCV
# define the parameter values that should be searched
# for python 2, k_range = range(1, 31)

from sklearn.linear_model import LogisticRegression
from sklearn import metrics

logreg = LogisticRegression(class_weight='balanced')

k_range = [0.0001, 0.001, 0.01, 0.1, 10, 100, 1000]
#print(k_range)

# create a parameter grid: map the parameter names to the values that should be searched
# simply a python dictionary
# key: parameter name
# value: list of values that should be searched for that parameter
# single key-value pair for param_grid
param_grid = [{'C': [0.0001, 0.001, 0.01, 0.1, 10, 100, 1000]}]


# instantiate model

#clf = MultinomialNB(class_prior=[0.5, 0.5])
#knn

#instantiate the grid
grid = GridSearchCV(logreg, param_grid, cv=3, scoring='roc_auc', return_train_score=True)
```

In [51]:

```
grid.fit(X1_train_tr, y_train)
```

Out[51]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=LogisticRegression(C=1.0, class_weight='balanced',
                                          dual=False, fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='warn',
                                          n_jobs=None, penalty='l2',
                                          random_state=None, solver='warn',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'C': [0.0001, 0.001, 0.01, 0.1, 10, 100, 1000]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [52]:

```
grid.cv_results_
```

Out[52]:

```
{'mean_fit_time': array([ 4.09275397,  1.10848125,  2.10460409,  4.94527777, 18.81157764,
        37.80075399, 59.80311569]),
 'std_fit_time': array([4.86863146, 0.0675513 , 0.21958588, 0.34459843, 0.76157566,
        4.03963404, 1.15384125]),
 'mean_score_time': array([0.17590404, 0.04450329, 0.08259241, 0.02598484, 0.10071484,
        0.04564126, 0.03731259]),
 'std_score_time': array([0.19359052, 0.0343256 , 0.08429443, 0.00141321, 0.06175841,
        0.02998693, 0.01227666]),
 'param_C': masked_array(data=[0.0001, 0.001, 0.01, 0.1, 10, 100, 1000],
              mask=[False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'C': 0.0001},
  {'C': 0.001},
  {'C': 0.01},
  {'C': 0.1},
  {'C': 10},
  {'C': 100},
  {'C': 1000}],
 'split0_test_score': array([0.663842  , 0.68728863, 0.68644114, 0.65839287, 0.6314427 ,
        0.6268488 , 0.62463969]),
 'split1_test_score': array([0.65634538, 0.68259324, 0.68326131, 0.65479666, 0.62606215,
        0.6217089 , 0.61950633]),
 'split2_test_score': array([0.66426113, 0.68881498, 0.69071935, 0.66551629, 0.63794517,
        0.63353385, 0.63146948]),
 'mean_test_score': array([0.66148283, 0.68623226, 0.68680721, 0.65956851, 0.63181659,
        0.62736376, 0.62520508]),
 'std_test_score': array([0.00363675, 0.00264755, 0.00305569, 0.00445449, 0.00485838,
        0.00484118, 0.00490023]),
 'rank_test_score': array([3, 2, 1, 4, 5, 6, 7]),
 'split0_train_score': array([0.67828161, 0.73373588, 0.80653225, 0.85178448, 0.8776722 ,
        0.87826718, 0.87828928]),
 'split1_train_score': array([0.68240301, 0.73660135, 0.80918505, 0.85498122, 0.88189804,
        0.88238093, 0.88238198]),
 'split2_train_score': array([0.68116396, 0.73526084, 0.80569524, 0.85043244, 0.87665843,
        0.87718509, 0.87720382]),
 'mean_train_score': array([0.68061619, 0.73519936, 0.80713751, 0.85239938, 0.87874289,
        0.87927773, 0.87929169]),
 'std_train_score': array([0.00172656, 0.00117063, 0.0014876 , 0.00190725, 0.00226909,
        0.00223832, 0.00222964])}
```

In [53]:

```
pd.DataFrame(grid.cv_results_)
```

Out[53]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_score	split1_test_score	s
0	4.092754	4.868631	0.175904	0.193591	0.0001	{'C': 0.0001}	0.663842	0.656345	0
1	1.108481	0.067551	0.044503	0.034326	0.001	{'C': 0.001}	0.687289	0.682593	0
2	2.104604	0.219586	0.082592	0.084294	0.01	{'C': 0.01}	0.686441	0.683261	0
3	4.945278	0.344598	0.025985	0.001413	0.1	{'C': 0.1}	0.658393	0.654797	0
4	18.811578	0.761576	0.100715	0.061758	10	{'C': 10}	0.631443	0.626062	0
5	37.800754	4.039634	0.045641	0.029987	100	{'C': 100}	0.626849	0.621709	0
6	59.803116	1.153841	0.037313	0.012277	1000	{'C': 1000}	0.624640	0.619506	0

In [54]:

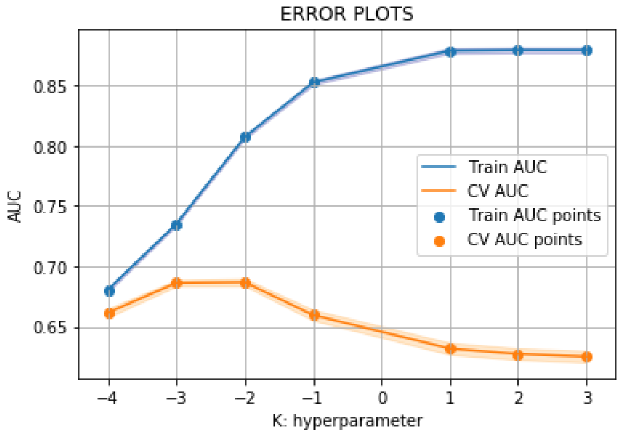
```
# examine the best model

# Single best score achieved across all params (k)
print(grid.best_score_)

# Dictionary containing the parameters (k) used to generate that score
print(grid.best_params_)

# Actual model object fit with those best parameters
# Shows default parameters that we did not specify

print(grid.best_estimator_)
```

```
0.6868072095970453
{'C': 0.01}
LogisticRegression(C=0.01, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [55]:

```
train_auc1 = grid.cv_results_['mean_train_score']
train_auc_std1 = grid.cv_results_['std_train_score']
cv_auc1 = grid.cv_results_['mean_test_score']
cv_auc_std1= grid.cv_results_['std_test_score']

log_k_range =[]
for a in tqdm(k_range):

    b = np.log10(a)
    log_k_range.append(b)

plt.plot(log_k_range, train_auc1, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_k_range, train_auc1 - train_auc_std1,train_auc1 + train_auc_std1,alpha=0
.2,color='darkblue')

plt.plot(log_k_range, cv_auc1, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_k_range, cv_auc1 - cv_auc_std1,cv_auc1 + cv_auc_std1,alpha=0.2,color='da
rkorange')

plt.scatter(log_k_range, train_auc1, label='Train AUC points')
plt.scatter(log_k_range, cv_auc1, label='CV AUC points')


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

```
100%|██████████████████████████████████████████| 7/7 [00:00<00:00, 184.31it/s]
```

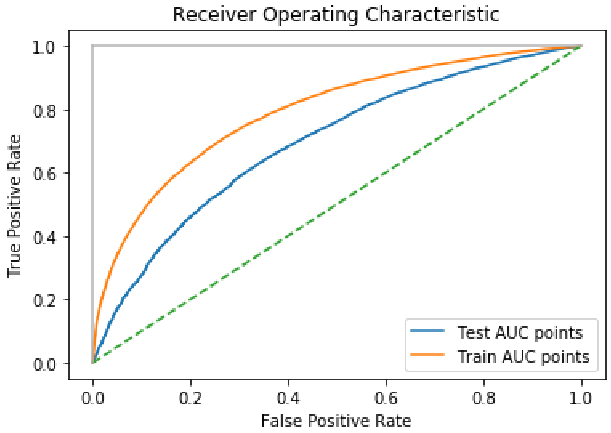


In [56]:

```
Trained_model_BOW = LogisticRegression(C=0.01, class_weight='balanced').fit(X1_train_tr, y_train)
```

In [57]:

```
# Get predicted probabilities
y_score_test = Trained_model_BOW.predict_proba(X1_test)[:,1]
y_score_train = Trained_model_BOW.predict_proba(X1_train_tr)[:,1]

# Create true and false positive rates
false_positive_rate, true_positive_rate, threshold = roc_curve(y_test, y_score_test)
false_positive_rate1, true_positive_rate1, threshold1 = roc_curve(y_train, y_score_train)

# Plot ROC curve
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate, true_positive_rate, label='Test AUC points')
plt.plot(false_positive_rate1, true_positive_rate1, label='Train AUC points')
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend()
plt.show()
```



In [58]:

```
import numpy as np
from sklearn.metrics import roc_auc_score

roc_auc_score(y_test, y_score_test)
```

Out[58]:

```
0.6934695793891335
```

In [59]:

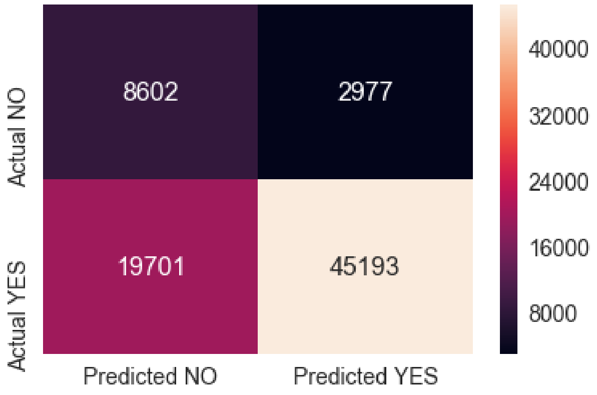
```
#https://chrisalbon.com/machine_learning/model_evaluation/generate_text_reports_on_performance/
from sklearn.metrics import classification_report

# Create list of target class names
#class_names = project_data['project_is_approved'].target_names

# Train model and make predictions
y_hat = Trained_model_BOW.predict(X1_test)

print(classification_report(y_test, y_hat))
```

```
              precision    recall  f1-score   support

           0       0.25      0.61      0.35      4963
           1       0.91      0.67      0.77     27812
```

```
    accuracy                           0.66     32775
   macro avg       0.58      0.64      0.56     32775
weighted avg       0.81      0.66      0.71     32775
```

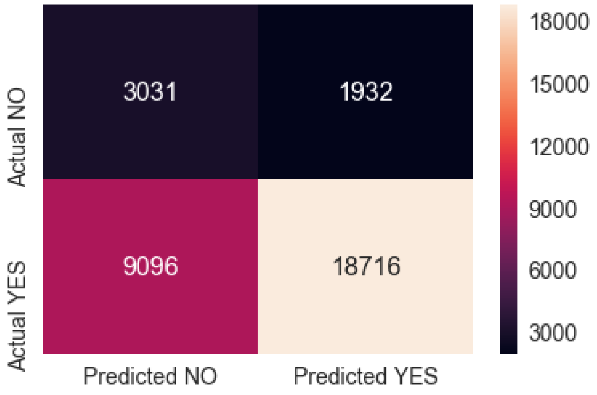
In [60]:

```
#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')


get_confusion_matrix(Trained_model_BOW, X1_train_tr, y_train)
```



In [61]:

```
get_confusion_matrix(Trained_model_BOW, X1_test, y_test)
```



In [62]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets, neighbors
from matplotlib.colors import ListedColormap

#https://www.ritchieng.com/machine-learning-efficiently-search-tuning-param/

# imports
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import GridSearchCV
# define the parameter values that should be searched
# for python 2, k_range = range(1, 31)

from sklearn.linear_model import LogisticRegression
from sklearn import metrics

logreg = LogisticRegression(class_weight='balanced')

k_range = [0.0001, 0.001, 0.01, 0.1, 10, 100, 1000]
#print(k_range)

# create a parameter grid: map the parameter names to the values that should be searched
# simply a python dictionary
# key: parameter name
# value: list of values that should be searched for that parameter
# single key-value pair for param_grid
param_grid = [{'C': [0.0001, 0.001, 0.01, 0.1, 10, 100, 1000]}]


# instantiate model

#clf = MultinomialNB(class_prior=[0.5, 0.5])
#knn

#instantiate the grid
grid2 = GridSearchCV(logreg, param_grid, cv=3, scoring='roc_auc', return_train_score=True)
```

In [63]:

```
grid2.fit(X2_train_tr, y_train)
```

Out[63]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=LogisticRegression(C=1.0, class_weight='balanced',
                                          dual=False, fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='warn',
                                          n_jobs=None, penalty='l2',
                                          random_state=None, solver='warn',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'C': [0.0001, 0.001, 0.01, 0.1, 10, 100, 1000]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [64]:

```
# view the complete results (list of named tuples)
grid2.cv_results_
```

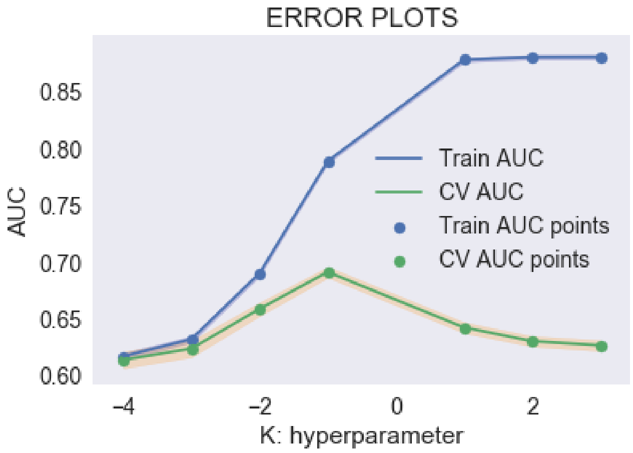
Out[64]:

```
{'mean_fit_time': array([ 0.53019031,  1.17778111,  1.89467072,  3.79866584, 15.99199677,
        35.41416152, 54.98649756]),
 'std_fit_time': array([0.0089808 , 0.20220755, 0.11118002, 0.11463677, 0.23690525,
        3.50078018, 5.44184791]),
 'mean_score_time': array([0.021655  , 0.02431973, 0.02132265, 0.0223213 , 0.02232083,
        0.02232154, 0.02865036]),
 'std_score_time': array([0.00169888, 0.00286528, 0.00094268, 0.00047131, 0.00047182,
        0.00188441, 0.00817411]),
 'param_C': masked_array(data=[0.0001, 0.001, 0.01, 0.1, 10, 100, 1000],
              mask=[False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'C': 0.0001},
  {'C': 0.001},
  {'C': 0.01},
  {'C': 0.1},
  {'C': 10},
  {'C': 100},
  {'C': 1000}],
 'split0_test_score': array([0.6201511 , 0.62934267, 0.66484525, 0.69622014, 0.6423129 ,
        0.63009526, 0.62621802]),
 'split1_test_score': array([0.60313677, 0.61286892, 0.6503419 , 0.68452589, 0.63573589
 'split1_test_score': array([0.60313677, 0.61286892, 0.6503419 , 0.68452589, 0.63573589,
        0.62394578, 0.62012393]),
 'split2_test_score': array([0.61657359, 0.62725012, 0.65930615, 0.69103601, 0.64734963,
        0.63515976, 0.6317598 ]),
 'mean_test_score': array([0.6132872 , 0.62315393, 0.65816451, 0.69059408, 0.64179941,
        0.62973353, 0.62603385]),
 'std_test_score': array([0.00732453, 0.0073226 , 0.0059758 , 0.00478442, 0.00475513,
        0.00458518, 0.00475206]),
 'rank_test_score': array([7, 6, 2, 1, 3, 4, 5]),
 'split0_train_score': array([0.61274906, 0.6281444 , 0.68629148, 0.78649287, 0.87672537,
        0.87918067, 0.87927595]),
 'split1_train_score': array([0.62070117, 0.63476305, 0.69164141, 0.79050039, 0.88021763,
        0.88257861, 0.88266271]),
 'split2_train_score': array([0.61570575, 0.63157664, 0.69081674, 0.78754859, 0.8754416 ,
        0.87769864, 0.87776591]),
 'mean_train_score': array([0.61638532, 0.6314947 , 0.68958321, 0.78818062, 0.87746154,
        0.87981931, 0.87990152]),
 'std_train_score': array([0.0032818 , 0.00270267, 0.00235182, 0.00169601, 0.0020181 ,
        0.00204278, 0.00204746])}
```

In [65]:

```
train_auc2 = grid2.cv_results_['mean_train_score']
train_auc_std2 = grid2.cv_results_['std_train_score']
cv_auc2 = grid2.cv_results_['mean_test_score']
cv_auc_std2 = grid2.cv_results_['std_test_score']

log_k_range =[]
for a in tqdm(k_range):

    b = np.log10(a)
    log_k_range.append(b)

plt.plot(log_k_range, train_auc2, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_k_range, train_auc2 - train_auc_std2, train_auc2 + train_auc_std2,alpha=
0.2,color='darkblue')

plt.plot(log_k_range, cv_auc2, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_k_range, cv_auc2 - cv_auc_std2, cv_auc2 + cv_auc_std2,alpha=0.2,color='d
arkorange')

plt.scatter(log_k_range, train_auc2, label='Train AUC points')
plt.scatter(log_k_range, cv_auc2, label='CV AUC points')


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

```
100%|██████████████████████████████████████████████████████| 7/7 [00:00<?, ?it/s]
```



In [66]:

```
# examine the best model

# Single best score achieved across all params (k)
print(grid2.best_score_)

# Dictionary containing the parameters (k) used to generate that score
print(grid2.best_params_)

# Actual model object fit with those best parameters
# Shows default parameters that we did not specify
print(grid2.best_estimator_)
```

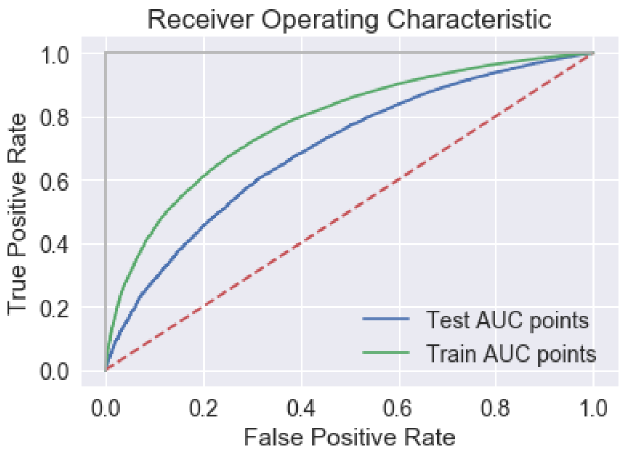
```
0.6905940814600453
{'C': 0.1}
LogisticRegression(C=0.1, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [67]:

```
Trained_model_TFIDF = LogisticRegression(C=0.1, class_weight='balanced').fit(X2_train_tr, y_train)
```

In [68]:

```
# Get predicted probabilities
y_score_test2 = Trained_model_TFIDF.predict_proba(X2_test)[:,1]
y_score_train2 = Trained_model_TFIDF.predict_proba(X2_train_tr)[:,1]

# Create true and false positive rates
false_positive_rate2, true_positive_rate2, threshold2 = roc_curve(y_test, y_score_test2)
false_positive_rate21, true_positive_rate21, threshold21 = roc_curve(y_train, y_score_train2)

# Plot ROC curve
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate2, true_positive_rate2, label='Test AUC points')
plt.plot(false_positive_rate21, true_positive_rate21, label='Train AUC points')
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend()
plt.show()
```



In [69]:

```
import numpy as np
from sklearn.metrics import roc_auc_score

roc_auc_score(y_test, y_score_test2)
```

Out[69]:

```
0.6950772912128494
```

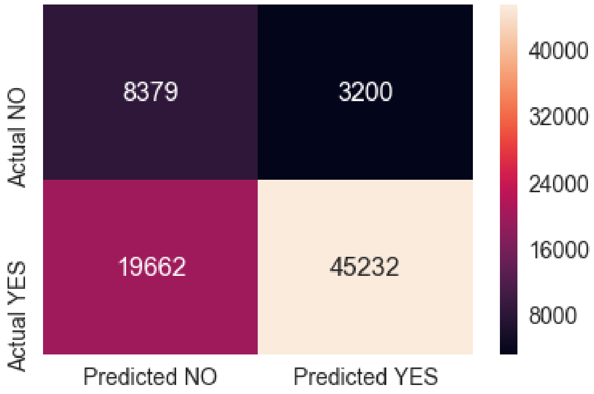
In [70]:

```
#https://chrisalbon.com/machine_learning/model_evaluation/generate_text_reports_on_performance/
from sklearn.metrics import classification_report

# Create list of target class names
#class_names = project_data['project_is_approved'].target_names

# Train model and make predictions
y_hat_2 = Trained_model_TFIDF.predict(X2_test)

print(classification_report(y_test, y_hat_2))
```

```
              precision    recall  f1-score   support

           0       0.25      0.61      0.36      4963
           1       0.91      0.68      0.77     27812

    accuracy                           0.67     32775
   macro avg       0.58      0.64      0.57     32775
weighted avg       0.81      0.67      0.71     32775
```

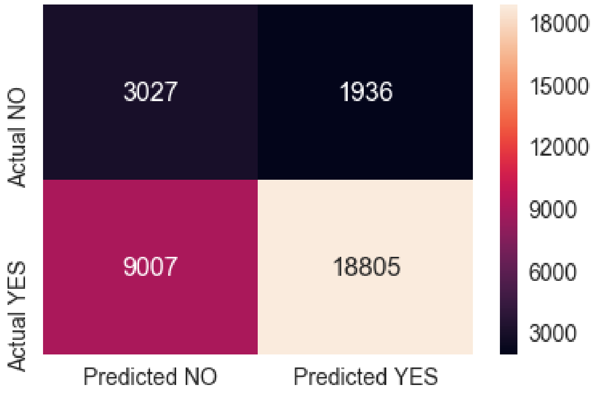
In [71]:

```
get_confusion_matrix(Trained_model_TFIDF, X2_train_tr, y_train)
```



In [72]:

```
get_confusion_matrix(Trained_model_TFIDF, X2_test, y_test)
```



In [73]:

```
grid3 = GridSearchCV(logreg, param_grid, cv=3, scoring='roc_auc', return_train_score=True)
```

In [74]:

In [74]:

```
grid3.fit(X3_train_tr, y_train)
```

Out[74]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=LogisticRegression(C=1.0, class_weight='balanced',
                                          dual=False, fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='warn',
                                          n_jobs=None, penalty='l2',
                                          random_state=None, solver='warn',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'C': [0.0001, 0.001, 0.01, 0.1, 10, 100, 1000]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [75]:

```
# view the complete results (list of named tuples)
grid3.cv_results_
```

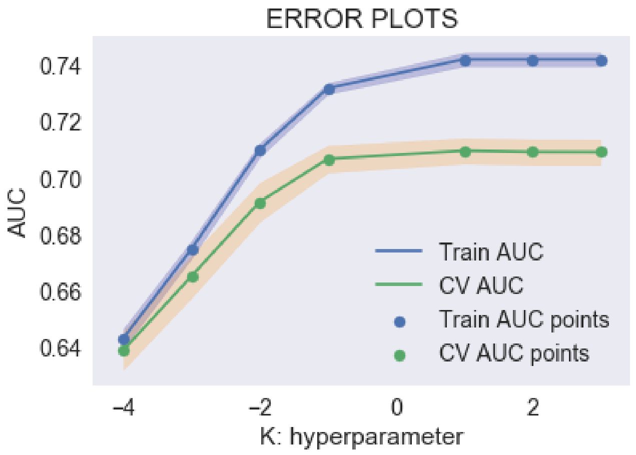
Out[75]:

```
{'mean_fit_time': array([  5.04438321,   5.71241514,  10.79595566,  26.24340796,
         85.20826705, 127.49601046, 177.36562761]),
 'std_fit_time': array([1.03741952, 0.26086077, 0.22070014, 2.30702216, 6.49306214,
        2.55229036, 3.84250307]),
 'mean_score_time': array([0.07429202, 0.06962832, 0.07828975, 0.07162372, 0.07562407,
        0.08028833, 0.07662336]),
 'std_score_time': array([0.00555571, 0.00703584, 0.01712365, 0.00833509, 0.00046974,
        0.00654421, 0.00249237]),
 'param_C': masked_array(data=[0.0001, 0.001, 0.01, 0.1, 10, 100, 1000],
              mask=[False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'C': 0.0001},
  {'C': 0.001},
  {'C': 0.01},
  {'C': 0.1},
  {'C': 10},
  {'C': 100},
  {'C': 1000}],
 'split0_test_score': array([0.64634434, 0.67362283, 0.69911237, 0.71196741, 0.71508593,
        0.71470585, 0.71465796]),
 'split1_test_score': array([0.62990923, 0.65541754, 0.68239002, 0.70036741, 0.704166  ,
        0.70374005, 0.70368993]),
 'split2_test_score': array([0.63911577, 0.66535323, 0.69223565, 0.70726839, 0.70907747,
        0.70862287, 0.70856872]),
 'mean_test_score': array([0.63845654, 0.66479798, 0.6912461 , 0.70653446, 0.70944321,
        0.709023  , 0.70897228]),
 'std_test_score': array([0.00672584, 0.00744271, 0.0068627 , 0.00476408, 0.00446558,
        0.00448574, 0.00448681]),
 'rank_test_score': array([7, 6, 5, 4, 1, 2, 3]),
 'split0_train_score': array([0.63847128, 0.67051238, 0.70686145, 0.72931792, 0.73898551,
        0.73900175, 0.73900031]),
 'split1_train_score': array([0.64716163, 0.67821027, 0.71289217, 0.73409352, 0.7449301 ,
        0.7449654 , 0.74496418]),
 'split2_train_score': array([0.64265677, 0.67471035, 0.7095361 , 0.73160996, 0.74203202,
        0.74202244, 0.74201887]),
 'mean_train_score': array([0.64276323, 0.67447767, 0.70976324, 0.7316738 , 0.74198255,
        0.74199653, 0.74199446]),
 'std_train_score': array([0.00354862, 0.00314696, 0.00246727, 0.00195015, 0.00242712,
        0.00243472, 0.0024348 ])}
```

In [76]:

```
train_auc3 = grid3.cv_results_['mean_train_score']
train_auc_std3 = grid3.cv_results_['std_train_score']
cv_auc3 = grid3.cv_results_['mean_test_score']
cv_auc_std3 = grid3.cv_results_['std_test_score']

log_k_range =[]
log_k_range = []
for a in tqdm(k_range):

    b = np.log10(a)
    log_k_range.append(b)

plt.plot(log_k_range, train_auc3, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_k_range, train_auc3 - train_auc_std3, train_auc3 + train_auc_std3, alpha
=0.2,color='darkblue')

plt.plot(log_k_range, cv_auc3, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_k_range, cv_auc3 - cv_auc_std3, cv_auc3 + cv_auc_std3, alpha=0.2,color='
darkorange')

plt.scatter(log_k_range, train_auc3, label='Train AUC points')
plt.scatter(log_k_range, cv_auc3, label='CV AUC points')


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

```
100%|██████████████████████████████████████████| 7/7 [00:00<00:00, 6998.84it/s]
```



In [77]:

```
# examine the best model

# Single best score achieved across all params (k)
print(grid3.best_score_)

# Dictionary containing the parameters (k) used to generate that score
print(grid3.best_params_)

# Actual model object fit with those best parameters
# Shows default parameters that we did not specify
print(grid3.best_estimator_)
```

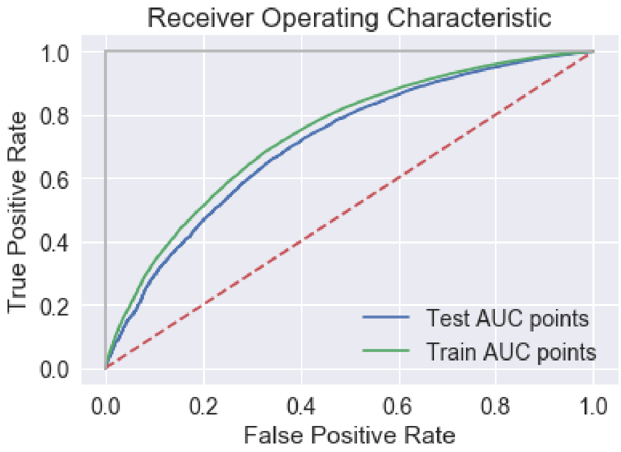
```
0.7094432113943934
{'C': 10}
LogisticRegression(C=10, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [78]:

```
Trained_model_AVGW2V = LogisticRegression(C=10, class_weight='balanced').fit(X3_train_tr, y_train)
```

In [79]:

```
# Get predicted probabilities
y_score_test3 = Trained_model_AVGW2V.predict_proba(X3_test)[:,1]
y_score_train3 = Trained_model_AVGW2V.predict_proba(X3_train_tr)[:,1]

# Create true and false positive rates
false_positive_rate3, true_positive_rate3, threshold3 = roc_curve(y_test, y_score_test3)
false_positive_rate31, true_positive_rate31, threshold31 = roc_curve(y_train, y_score_train3)

# Plot ROC curve
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate3, true_positive_rate3, label='Test AUC points')
plt.plot(false_positive_rate31, true_positive_rate31, label='Train AUC points')
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend()
plt.show()
```



In [80]:

```
import numpy as np
from sklearn.metrics import roc_auc_score

roc_auc_score(y_test, y_score_test3)
```

Out[80]:

```
0.7112609652576773
```

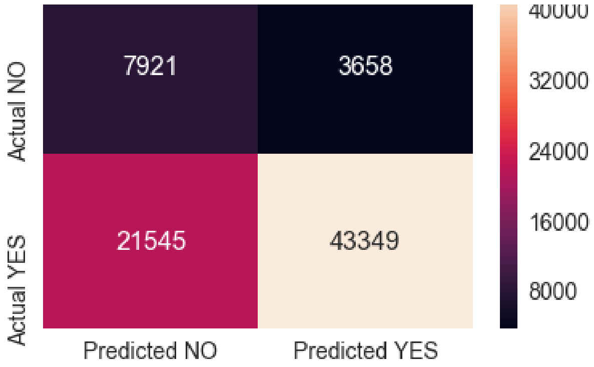
In [81]:

```
# Train model and make predictions
y_hat_3 = Trained_model_AVGW2V.predict(X3_test)

print(classification_report(y_test, y_hat_3))
```

```
              precision    recall  f1-score   support

           0       0.26      0.65      0.37      4963
           1       0.91      0.66      0.77     27812

    accuracy                           0.66     32775
   macro avg       0.59      0.66      0.57     32775
weighted avg       0.82      0.66      0.71     32775
```

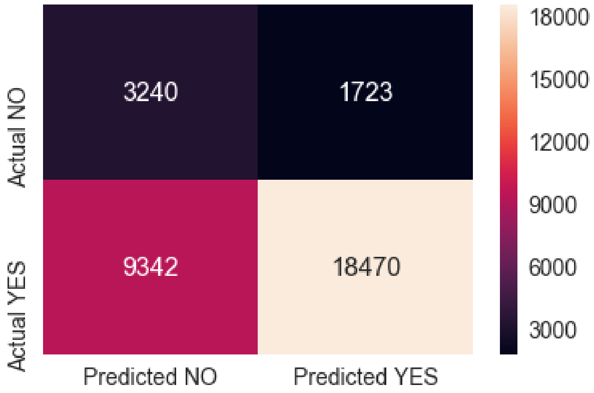
In [82]:

```
get_confusion_matrix(Trained_model_AVGW2V, X3_train_tr, y_train)
```



In [83]:

```
get_confusion_matrix(Trained_model_AVGW2V, X3_test, y_test)
```



In [84]:

```
grid4 = GridSearchCV(logreg, param_grid, cv=3, scoring='roc_auc', return_train_score=True)
```

In [85]:

```
grid4.fit(X4_train_tr, y_train)
```

Out[85]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=LogisticRegression(C=1.0, class_weight='balanced',
                                          dual=False, fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='warn',
                                          n_jobs=None, penalty='l2',
                                          random_state=None, solver='warn',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'C': [0.0001, 0.001, 0.01, 0.1, 10, 100, 1000]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [86]:

```
# view the complete results (list of named tuples)
grid4.cv_results_
```

Out[86]:

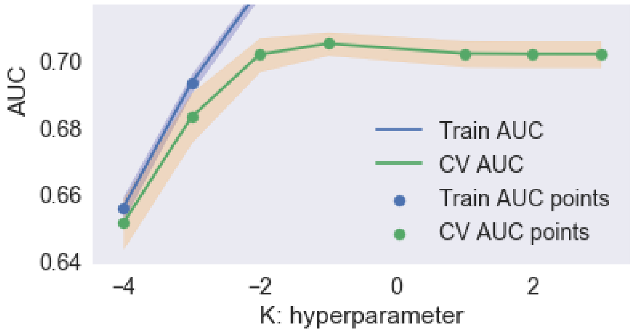
```
{'mean_fit_time': array([  3.73508445,   5.99941595,  11.29485226,  27.67486413,
         82.04694446, 104.57397405, 164.7500627 ]),
 'std_fit_time': array([ 0.75113749,  0.21983782,  0.87812639,  0.71977376,  4.53080225,
         5.1183976 , 34.93392509]),
 'mean_score_time': array([0.07229209, 0.07862298, 0.0746882 , 0.09594607, 0.09394813,
        0.07129486, 0.09262737]),
 'std_score_time': array([0.0119474 , 0.01160741, 0.00776072, 0.03341597, 0.01282427,
        0.00385633, 0.01746666]),
 'param_C': masked_array(data=[0.0001, 0.001, 0.01, 0.1, 10, 100, 1000],
              mask=[False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'C': 0.0001},
  {'C': 0.001},
  {'C': 0.01},
  {'C': 0.1},
  {'C': 10},
  {'C': 100},
  {'C': 1000}],
 'split0_test_score': array([0.65914568, 0.69048323, 0.70641052, 0.70848816, 0.70655812,
        0.70646131, 0.7064507 ]),
 'split1_test_score': array([0.64134566, 0.67293523, 0.69474816, 0.70038519, 0.69716006,
        0.69699584, 0.69697838]),
 'split2_test_score': array([0.65330883, 0.68510696, 0.70347056, 0.70552449, 0.70186606,
        0.70170587, 0.70168959]),
 'mean_test_score': array([0.6512668 , 0.68284188, 0.70154312, 0.70479932, 0.70186147,
        0.70172107, 0.70170629]),
 'std_test_score': array([0.00740896, 0.00734086, 0.0049524 , 0.00334756, 0.00383678,
        0.00386431, 0.00386711]),
 'rank_test_score': array([7, 6, 5, 1, 2, 3, 4]),
 'split0_train_score': array([0.65225163, 0.69030624, 0.71879725, 0.73023024, 0.73245335,
        0.73244333, 0.73244225]),
 'split1_train_score': array([0.6603268 , 0.69662067, 0.72374351, 0.73590339, 0.73899016,
        0.73896652, 0.73896333]),
 'split2_train_score': array([0.65526799, 0.69239736, 0.72048164, 0.73275764, 0.73561826,
        0.73561095, 0.73560885]),
 'mean_train_score': array([0.65594881, 0.69310809, 0.72100747, 0.73296376, 0.73568726,
        0.7356736 , 0.73567148]),
 'std_train_score': array([0.00333164, 0.00262639, 0.00205325, 0.00232063, 0.00266909,
        0.00266345, 0.00266259])}
```

In [87]:

```
train_auc4 = grid4.cv_results_['mean_train_score']
train_auc_std4 = grid4.cv_results_['std_train_score']
cv_auc4 = grid4.cv_results_['mean_test_score']
cv_auc_std4 = grid4.cv_results_['std_test_score']

log_k_range =[]
for a in tqdm(k_range):

    b = np.log10(a)
    log_k_range.append(b)

plt.plot(log_k_range, train_auc4, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_k_range, train_auc4 - train_auc_std4, train_auc4 + train_auc_std4, alpha
=0.2,color='darkblue')

plt.plot(log_k_range, cv_auc4, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_k_range, cv_auc4 - cv_auc_std4, cv_auc4 + cv_auc_std4, alpha=0.2,color='
darkorange')

plt.scatter(log_k_range, train_auc4, label='Train AUC points')
plt.scatter(log_k_range, cv_auc4, label='CV AUC points')


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

```
100%|██████████████████████████████████████████████| 7/7 [00:00<00:00, 31.44it/s]
```

ERROR PLOTS
0.74
0.72



In [88]:

```
# examine the best model

# Single best score achieved across all params (k)
print(grid4.best_score_)

# Dictionary containing the parameters (k) used to generate that score
print(grid4.best_params_)

# Actual model object fit with those best parameters
# Shows default parameters that we did not specify
print(grid4.best_estimator_)
```

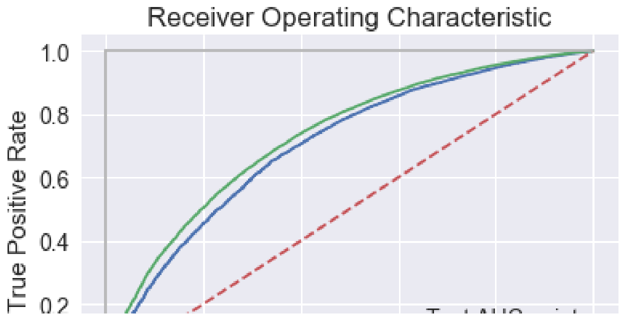
```
0.7047993218788849
{'C': 0.1}
LogisticRegression(C=0.1, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [89]:

```
Trained_model_TFIDFW2V = LogisticRegression(C=0.1, class_weight='balanced').fit(X4_train_tr, y_trai
n)
```

In [90]:

```
# Get predicted probabilities
y_score_test4 = Trained_model_TFIDFW2V.predict_proba(X4_test)[:,1]
y_score_train4 = Trained_model_TFIDFW2V.predict_proba(X4_train_tr)[:,1]

# Create true and false positive rates
false_positive_rate4, true_positive_rate4, threshold4 = roc_curve(y_test, y_score_test4)
false_positive_rate41, true_positive_rate41, threshold41 = roc_curve(y_train, y_score_train4)

# Plot ROC curve
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate4, true_positive_rate4, label='Test AUC points')
plt.plot(false_positive_rate41, true_positive_rate41, label='Train AUC points')
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend()
plt.show()
```



In [91]:

```
roc_auc_score(y_test, y_score_test4)
```

Out[91]:

```
0.7038977401562008
```

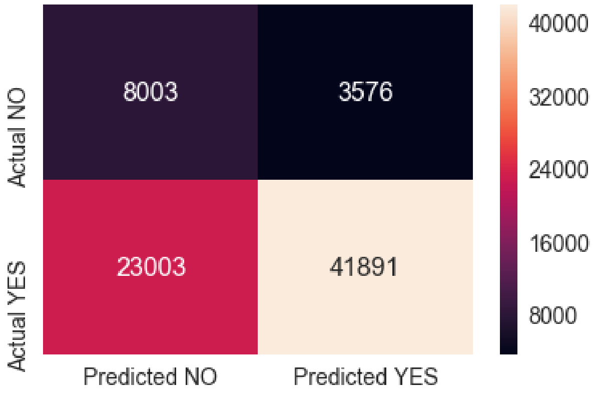
In [92]:

```
y_hat_4 = Trained_model_AVGW2V.predict(X4_test)

print(classification_report(y_test, y_hat_4))
```

```
              precision    recall  f1-score   support

           0       0.28      0.54      0.37      4963
           1       0.90      0.75      0.82     27812

    accuracy                           0.72     32775
   macro avg       0.59      0.65      0.60     32775
weighted avg       0.81      0.72      0.75     32775
```

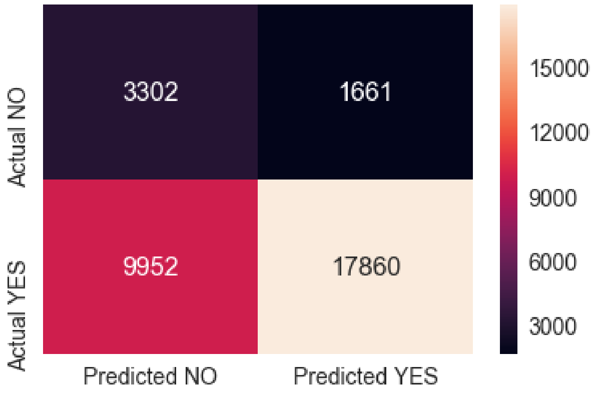
In [93]:

```
get_confusion_matrix(Trained_model_TFIDFW2V, X4_train_tr, y_train)
```



In [94]:

```
get_confusion_matrix(Trained_model_TFIDFW2V, X4_test, y_test)
```



2.5 Logistic Regression with added Features `Set 5`

In [95]:

```
# 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


#price scalar and standardized for test
price_scalar_test = StandardScaler()
price_scalar_test.fit(X_test['price'].values.reshape(-1,1)) # finding the mean and standard
deviation of this data
#print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")
price_standardized_test = price_scalar_test.transform(X_test['price'].values.reshape(-1, 1))

essay_sentiment_score = StandardScaler()
essay_sentiment_score.fit(X_train['essay_sentiment'].values.reshape(-1,1)) # finding the mean and
standard deviation of this data
#print(f"Mean : {teacher_number_of_previously_posted_projects_scalar_train_tr.mean_[0]}, Standard
deviation : {np.sqrt(teacher_number_of_previously_posted_projects_scalar_train_tr.var_[0])}")
# Now standardize the data with above maen and variance.
essay_sentiment_score_train = essay_sentiment_score.transform(X_train['essay_sentiment'].values.re
shape(-1, 1))
essay_sentiment_score_test =
essay_sentiment_score.transform(X_test['essay_sentiment'].values.reshape(-1, 1))
print(essay_sentiment_score_train.shape)
print(essay_sentiment_score_test.shape)

title_count = StandardScaler()
title_count.fit(X_train['title_count'].values.reshape(-1,1)) # finding the mean and standard
deviation of this data
#print(f"Mean : {teacher_number_of_previously_posted_projects_scalar_train_tr.mean_[0]}, Standard
deviation : {np.sqrt(teacher_number_of_previously_posted_projects_scalar_train_tr.var_[0])}")
# Now standardize the data with above maen and variance.
title_count_train = title_count.transform(X_train['title_count'].values.reshape(-1, 1))
title_count_test = title_count.transform(X_test['title_count'].values.reshape(-1, 1))
print(title_count_train.shape)
print(title_count_test.shape)

essay_count = StandardScaler()
essay_count.fit(X_train['essay_count'].values.reshape(-1,1)) # finding the mean and standard
deviation of this data
#print(f"Mean : {teacher_number_of_previously_posted_projects_scalar_train_tr.mean_[0]}, Standard
deviation : {np.sqrt(teacher_number_of_previously_posted_projects_scalar_train_tr.var_[0])}")
# Now standardize the data with above maen and variance.
essay_count_train = essay_count.transform(X_train['essay_count'].values.reshape(-1, 1))
essay_count_test = essay_count.transform(X_test['essay_count'].values.reshape(-1, 1))
print(essay_count_train.shape)
print(essay_count_test.shape)
```

```
(76473, 1)
(32775, 1)
(76473, 1)
(32775, 1)
(76473, 1)
(32775, 1)
```

In [96]:

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

#categorical, numerical features + project_title(BOW)
X5_train_tr = hstack((categories_one_hot_train_tr, sub_categories_one_hot_train_tr,
categories_state_1_train_tr, categories_grade_train_tr, categories_teacher_prefix_train_tr,
price_standardized_train_tr, teacher_number_of_previously_posted_projects_standardized_train_tr, e
ssay_sentiment_score_train, title_count_train, essay_count_train))


X5_test = hstack((categories_one_hot_test, sub_categories_one_hot_test, categories_state_1_test, c
ategories_grade_test, categories_teacher_prefix_test, price_standardized_test,
teacher_number_of_previously_posted_projects_standardized_test, essay_sentiment_score_test,
title_count_test, essay_count_test))
```

In [97]:

```
grid5 = GridSearchCV(logreg, param_grid, cv=3, scoring='roc_auc', return_train_score=True)
```

In [98]:

```
grid5.fit(X5_train_tr, y_train)
```

Out[98]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=LogisticRegression(C=1.0, class_weight='balanced',
                                          dual=False, fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='warn',
                                          n_jobs=None, penalty='l2',
                                          random_state=None, solver='warn',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'C': [0.0001, 0.001, 0.01, 0.1, 10, 100, 1000]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [99]:

```
grid5.cv_results_
```

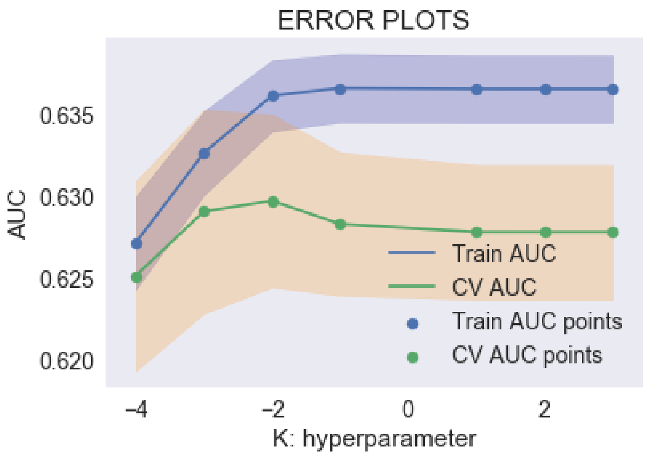
Out[99]:

```
{'mean_fit_time': array([0.38933929, 0.51454504, 0.71359046, 1.08904346, 3.17318249,
        3.71168439, 4.33871539]),
 'std_fit_time': array([0.21247205, 0.08904045, 0.08788508, 0.1513387 , 0.21904973,
        0.42108736, 0.18077436]),
 'mean_score_time': array([0.05262868, 0.02598508, 0.02132169, 0.0199887 , 0.02498603,
        0.02198847, 0.02099045]),
 'std_score_time': array([0.04689969, 0.00565386, 0.0012467 , 0.00294257, 0.00652819,
        0.00778457, 0.00706477]),
 'param_C': masked_array(data=[0.0001, 0.001, 0.01, 0.1, 10, 100, 1000],
              mask=[False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'C': 0.0001},
  {'C': 0.001},
  {'C': 0.01},
  {'C': 0.1},
  {'C': 10},
  {'C': 100},
  {'C': 1000}],
 'split0_test_score': array([0.62460317, 0.63058042, 0.63179823, 0.63032644, 0.6297326 ,
        0.62972461, 0.62972404]),
 'split1_test_score': array([0.61825391, 0.62076435, 0.62243027, 0.62219105, 0.62206554,
        0.6220686 , 0.6220674 ]),
 'split2_test_score': array([0.63247308, 0.63578907, 0.63487639, 0.63233081, 0.6315734 ,
        0.63157158, 0.63157245]),
 'mean_test_score': array([0.62510995, 0.62904454, 0.62970159, 0.62828274, 0.62779049,
        0.62778824, 0.62778794]),
 'std_test_score': array([0.00581595, 0.00622916, 0.00529294, 0.00438451, 0.00411731,
        0.00411408, 0.00411482]),
 'rank_test_score': array([7, 2, 1, 3, 4, 5, 6]),
 'split0_train_score': array([0.62515801, 0.63084536, 0.63438944, 0.63488114, 0.63485276,
        0.63485208, 0.63485179]),
 'split1_train_score': array([0.63116175, 0.63625408, 0.63920246, 0.63952699, 0.63943828,
        0.63943621, 0.63943577]),
 'split2_train_score': array([0.62504781, 0.63075123, 0.63477544, 0.63534213, 0.63531785,
        0.63531641, 0.63531602]),
 'mean_train_score': array([0.62712252, 0.63261689, 0.63612245, 0.63658342, 0.6365363 ,
        0.6365349 , 0.63653453]),
 'std_train_score': array([0.00285652, 0.00257217, 0.00218359, 0.00208991, 0.00206078,
        0.00206028, 0.00206022])}
```

In [100]:

```
train_auc5 = grid5.cv_results_['mean_train_score']
train_auc_std5 = grid5.cv_results_['std_train_score']
cv_auc5 = grid5.cv_results_['mean_test_score']
cv_auc_std5 = grid5.cv_results_['std_test_score']

log_k_range =[]
for a in tqdm(k_range):

    b = np.log10(a)
    log_k_range.append(b)

plt.plot(log_k_range, train_auc5, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_k_range, train_auc5 - train_auc_std5, train_auc5 + train_auc_std5, alpha
=0.2,color='darkblue')

plt.plot(log_k_range, cv_auc5, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_k_range, cv_auc5 - cv_auc_std5, cv_auc5 + cv_auc_std5, alpha=0.2,color='
darkorange')

plt.scatter(log_k_range, train_auc5, label='Train AUC points')
plt.scatter(log_k_range, cv_auc5, label='CV AUC points')


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

```
100%|████████████████████████████████████████████████████████████| 7/7 [00:00<?, ?it/s]
```



In [101]:

```
# examine the best model

# Single best score achieved across all params (k)
print(grid5.best_score_)

# Dictionary containing the parameters (k) used to generate that score
print(grid5.best_params_)

# Actual model object fit with those best parameters
# Shows default parameters that we did not specify
```

```
print(grid5.best_estimator_)
```

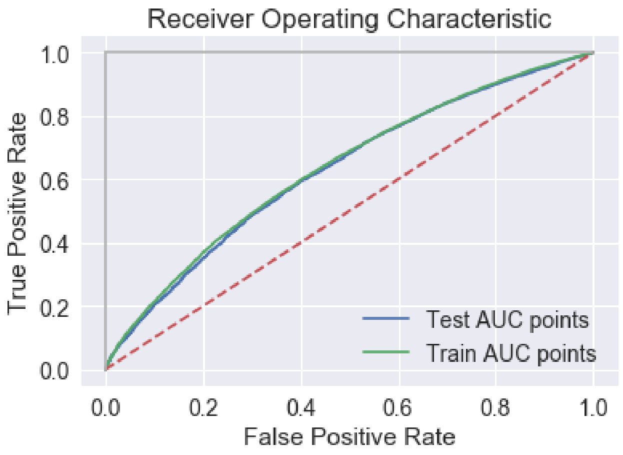
```
0.6297015884317912
{'C': 0.01}
LogisticRegression(C=0.01, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [102]:

```
Trained_model_Final = LogisticRegression(C=0.01, class_weight='balanced').fit(X5_train_tr, y_train)
```

In [103]:

```
# Get predicted probabilities
y_score_test5 = Trained_model_Final.predict_proba(X5_test)[:,1]
y_score_train5 = Trained_model_Final.predict_proba(X5_train_tr)[:,1]

# Create true and false positive rates
false_positive_rate5, true_positive_rate5, threshold5 = roc_curve(y_test, y_score_test5)
false_positive_rate51, true_positive_rate51, threshold51 = roc_curve(y_train, y_score_train5)

# Plot ROC curve
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate5, true_positive_rate5, label='Test AUC points')
plt.plot(false_positive_rate51, true_positive_rate51, label='Train AUC points')
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend()
plt.show()
```



In [104]:

```
roc_auc_score(y_test, y_score_test5)
```

Out[104]:

```
0.6280994315507022
```

In [105]:

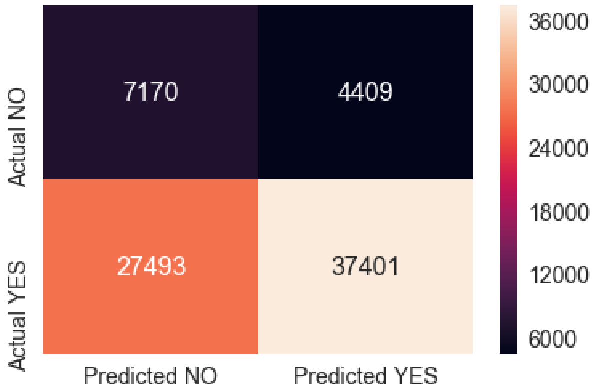
```
y_hat_5 = Trained_model_Final.predict(X5_test)

print(classification_report(y_test, y_hat_5))
```

```
              precision    recall  f1-score   support

           0       0.21      0.61      0.31      4963
           1       0.89      0.58      0.70     27812
```

```
    accuracy                           0.58     32775
   macro avg       0.55      0.60      0.51     32775
weighted avg       0.79      0.58      0.64     32775
```

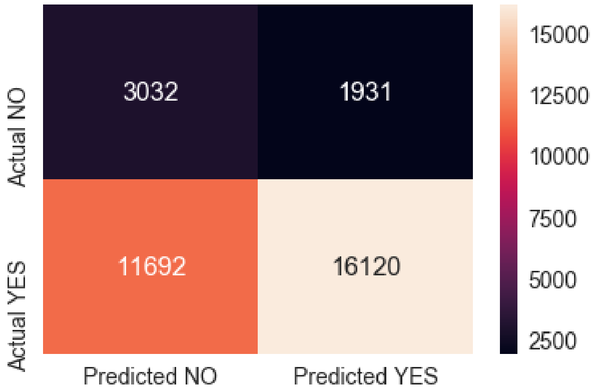
In [106]:

```
get_confusion_matrix(Trained_model_Final, X5_train_tr, y_train)
```



In [107]:

```
get_confusion_matrix(Trained_model_Final, X5_test, y_test)
```



3. Conclusion

In [108]:

```
# Please compare all your models using Prettytable library

from prettytable import PrettyTable

DBZ = PrettyTable()
DBZ.field_names = ["Vectorizer", "Model", "Hyperparameter", "Train-AUC", "Test-AUC"]

#DBZ.add_row(["BOW", "Multinomial Naive Bayes", "0.1", "0.7040514309423979",
"0.7122564520961516"])
#DBZ.add_row(["TF-IDF", "Multinomial Naive Bayes", "0.1", "0.6705654387437061",
"0.6781435281807364"])

DBZ.add_row(["BOW", "Logistic Regression", grid.best_params_ , grid.best_score_,
roc_auc_score(y_test, y_score_test)])
DBZ.add_row(["TF-IDF", "Logistic Regression", grid2.best_params_ , grid2.best_score_,
roc_auc_score(y_test, y_score_test2)])
DBZ.add_row(["AVGW2V", "Logistic Regression", grid3.best_params_ , grid3.best_score_,
roc_auc_score(y_test, y_score_test3)])
DBZ.add_row(["TFIDFW2V", "Logistic Regression", grid4.best_params_ , grid4.best_score_,
roc_auc_score(y_test, y_score_test4)])
DBZ.add_row(["No text data", "Logistic Regression", grid5.best_params_ , grid5.best_score_,
roc_auc_score(y_test, y_score_test5)])
roc_auc_score(y_test, y_score_test5)])

print(DBZ)
```

```
+--------------+---------------------+----------------+--------------------+--------------------+
|  Vectorizer  |        Model        | Hyperparameter |     Train-AUC      |      Test-AUC      |
+--------------+---------------------+----------------+--------------------+--------------------+
|     BOW      | Logistic Regression |  {'C': 0.01}   | 0.6868072095970453 | 0.6934695793891335 |
|    TF-IDF    | Logistic Regression |   {'C': 0.1}   | 0.6905940814600453 | 0.6950772912128494 |
|    AVGW2V    | Logistic Regression |   {'C': 10}    | 0.7094432113943934 | 0.7112609652576773 |
|   TFIDFW2V   | Logistic Regression |   {'C': 0.1}   | 0.7047993218788849 | 0.7038977401562008 |
| No text data | Logistic Regression |  {'C': 0.01}   | 0.6297015884317912 | 0.6280994315507022 |
+--------------+---------------------+----------------+--------------------+--------------------+
```