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]:

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


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


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


project_data.head(2)
```

Out[4]:

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

In [5]:

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

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

Out[5]:

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

1.2 preprocessing of `project_subject_categories`

In [6]:

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

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

```
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [7]:

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

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

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

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

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

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

1.3 Text preprocessing

In [8]:

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

In [9]:

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

Out[9]:

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

In [10]:

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

In [11]:

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

```
I have been fortunate enough to use the Fairy Tale STEM kits in my classroom as well as the STEM j
ournals, which my students really enjoyed.  I would love to implement more of the Lakeshore STEM k
its in my classroom for the next school year as they provide excellent and engaging STEM
lessons.My students come from a variety of backgrounds, including language and socioeconomic statu
s.  Many of them don't have a lot of experience in science and engineering and these kits give me
the materials to provide these exciting opportunities for my students.Each month I try to do
several science or STEM/STEAM projects.  I would use the kits and robot to help guide my science i
nstruction in engaging and meaningful ways.  I can adapt the kits to my current language arts paci
ng guide where we already teach some of the material in the kits like tall tales (Paul Bunyan) or
Johnny Appleseed.  The following units will be taught in the next school year where I will
implement these kits: magnets, motion, sink vs. float, robots.  I often get to these units and don
't know If I am teaching the right way or using the right materials.    The kits will give me
additional ideas, strategies, and lessons to prepare my students in science.It is challenging to d
evelop high quality science activities.  These kits give me the materials I need to provide my
students with science activities that will go along with the curriculum in my classroom.  Although
I have some things (like magnets) in my classroom, I don't know how to use them effectively.  The
kits will provide me with the right amount of materials and show me how to use them in an
appropriate way.
==================================================
I teach high school English to students with learning and behavioral disabilities. My students all
vary in their ability level. However, the ultimate goal is to increase all students literacy level
s. This includes their reading, writing, and communication levels.I teach a really dynamic group o
f students. However, my students face a lot of challenges. My students all live in poverty and in
a dangerous neighborhood. Despite these challenges, I have students who have the the desire to def
eat these challenges. My students all have learning disabilities and currently all are performing
below grade level. My students are visual learners and will benefit from a classroom that fulfills
their preferred learning style.The materials I am requesting will allow my students to be prepared
for the classroom with the necessary supplies.  Too often I am challenged with students who come t
o school unprepared for class due to economic challenges.  I want my students to be able to focus
on learning and not how they will be able to get school supplies.  The supplies will last all year
.  Students will be able to complete written assignments and maintain a classroom journal.  The ch
art paper will be used to make learning more visual in class and to create posters to aid students
```

in their learning. The students have access to a classroom printer. The toner will be used to print student work that is completed on the classroom Chromebooks.I want to try and remove all barriers for the students learning and create opportunities for learning. One of the biggest barriers is the students not having the resources to get pens, paper, and folders. My students will be able to increase their literacy skills because of this project.

==

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

==

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

==

My classroom consists of twenty-two amazing sixth graders from different cultures and backgrounds. They are a social bunch who enjoy working in partners and working with groups. They are hard-working and eager to head to middle school next year. My job is to get them ready to make this transition and make it as smooth as possible. In order to do this, my students need to come to school every day and feel safe and ready to learn. Because they are getting ready to head to middle school, I give them lots of choice- choice on where to sit and work, the order to complete assignments, choice of projects, etc. Part of the students feeling safe is the ability for them to come into a welcoming, encouraging environment. My room is colorful and the atmosphere is casual. I want them to take ownership of the classroom because we ALL share it together. Because my time with them is limited, I want to ensure they get the most of this time and enjoy it to the best of their abilities.Currently, we have twenty-two desks of differing sizes, yet the desks are similar to the ones the students will use in middle school. We also have a kidney table with crates for seating. I allow my students to choose their own spots while they are working independently or in groups. More often than not, most of them move out of their desks and onto the crates. Believe it or not, this has proven to be more successful than making them stay at their desks! It is because of this that I am looking toward the “Flexible Seating” option for my classroom.\r\n The students look forward to their work time so they can move around the room. I would like to get rid of the constricting desks and move toward more “fun” seating options. I am requesting various seating so my students have more options to sit. Currently, I have a stool and a papasan chair I inherited from the previous sixth-grade teacher as well as five milk crate seats I made, but I would like to give them more options and reduce the competition for the “good seats”. I am also requesting two rugs as not only more seating options but to make the classroom more welcoming and appealing. In order for my students to be able to write and complete work without desks, I am requesting a class set of clipboards. Finally, due to curriculum that requires groups to work together, I am requesting tables that we can fold up when we are not using them to leave more room for our flexible seating o

```
ables that we can fold up when we are not using them to leave more room for our flexible seating o
ptions.\r\nI know that with more seating options, they will be that much more excited about coming
to school! Thank you for your support in making my classroom one students will remember
forever!nannan
==================================================
```

In [12]:

```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 [13]:

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

```
\"A person is a person, no matter how small.\" (Dr.Seuss) I teach the smallest students with the b
iggest enthusiasm for learning. My students learn in many different ways using all of our senses a
nd multiple intelligences. I use a wide range of techniques to help all my students succeed. \r\nS
tudents in my class come from a variety of different backgrounds which makes for wonderful sharing
of experiences and cultures, including Native Americans.\r\nOur school is a caring community of su
ccessful learners which can be seen through collaborative student project based learning in and ou
t of the classroom. Kindergarteners in my class love to work with hands-on materials and have many
different opportunities to practice a skill before it is mastered. Having the social skills to wor
k cooperatively with friends is a crucial aspect of the kindergarten curriculum.Montana is the
perfect place to learn about agriculture and nutrition. My students love to role play in our
pretend kitchen in the early childhood classroom. I have had several kids ask me, \"Can we try coo
king with REAL food?\" I will take their idea and create \"Common Core Cooking Lessons\" where we
learn important math and writing concepts while cooking delicious healthy food for snack time. My
students will have a grounded appreciation for the work that went into making the food and knowled
ge of where the ingredients came from as well as how it is healthy for their bodies. This project
would expand our learning of nutrition and agricultural cooking recipes by having us peel our own
apples to make homemade applesauce, make our own bread, and mix up healthy plants from our classro
om garden in the spring. We will also create our own cookbooks to be printed and shared with famil
ies. \r\nStudents will gain math and literature skills as well as a life long enjoyment for health
y cooking.nannan
==================================================
```

In [14]:

```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)
```

```
 A person is a person, no matter how small.  (Dr.Seuss) I teach the smallest students with the big
gest enthusiasm for learning. My students learn in many different ways using all of our senses and
multiple intelligences. I use a wide range of techniques to help all my students succeed.
Students in my class come from a variety of different backgrounds which makes for wonderful
sharing of experiences and cultures, including Native Americans.  Our school is a caring community
of successful learners which can be seen through collaborative student project based learning in a
nd out of the classroom. Kindergarteners in my class love to work with hands-on materials and have
many different opportunities to practice a skill before it is mastered. Having the social skills t
o work cooperatively with friends is a crucial aspect of the kindergarten curriculum.Montana is
the perfect place to learn about agriculture and nutrition. My students love to role play in our p
retend kitchen in the early childhood classroom. I have had several kids ask me,  Can we try cooki
retend kitchen in the early childhood classroom. I have had several kids ask me, "Can we try cooki
ng with REAL food?  I will take their idea and create  Common Core Cooking Lessons  where we learn
important math and writing concepts while cooking delicious healthy food for snack time. My
students will have a grounded appreciation for the work that went into making the food and knowled
ge of where the ingredients came from as well as how it is healthy for their bodies. This project
would expand our learning of nutrition and agricultural cooking recipes by having us peel our own
apples to make homemade applesauce, make our own bread, and mix up healthy plants from our classro
om garden in the spring. We will also create our own cookbooks to be printed and shared with famil
ies.   Students will gain math and literature skills as well as a life long enjoyment for healthy
cooking.nannan
```

In [15]:

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

```
 A person is a person no matter how small Dr Seuss I teach the smallest students with the biggest
enthusiasm for learning My students learn in many different ways using all of our senses and multi
ple intelligences I use a wide range of techniques to help all my students succeed Students in my
class come from a variety of different backgrounds which makes for wonderful sharing of
experiences and cultures including Native Americans Our school is a caring community of successful
learners which can be seen through collaborative student project based learning in and out of the
classroom Kindergarteners in my class love to work with hands on materials and have many different
opportunities to practice a skill before it is mastered Having the social skills to work
cooperatively with friends is a crucial aspect of the kindergarten curriculum Montana is the
perfect place to learn about agriculture and nutrition My students love to role play in our
pretend kitchen in the early childhood classroom I have had several kids ask me Can we try cooking
with REAL food I will take their idea and create Common Core Cooking Lessons where we learn
important math and writing concepts while cooking delicious healthy food for snack time My
students will have a grounded appreciation for the work that went into making the food and knowled
ge of where the ingredients came from as well as how it is healthy for their bodies This project w
ould expand our learning of nutrition and agricultural cooking recipes by having us peel our own a
pples to make homemade applesauce make our own bread and mix up healthy plants from our classroom
garden in the spring We will also create our own cookbooks to be printed and shared with families
Students will gain math and literature skills as well as a life long enjoyment for healthy cooking
nannan
```

In [16]:

```python
# 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 [17]:

```python
# Combining all the above stundents
```

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

```
100%|████████████████████████████████| 109248/109248 [02:27<00:00, 738.25it/s]
```

In [18]:

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

Out[18]:

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

1.4 Preprocessing of `project_title`

In [19]:

```
# 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)
```

```
Engineering STEAM into the Primary Classroom
==================================================
Building Blocks for Learning
==================================================
Empowering Students Through Art:Learning About Then and Now
==================================================
Health Nutritional Cooking in Kindergarten
==================================================
Turning to Flexible Seating: One Sixth-Grade Class's Journey to Freedom
==================================================
```

In [20]:

```
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('\\"', ' ')
    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())
```

```
Special Needs Students Need Additional Access to Technology
==================================================
```

```
100%|███████████████████████████████| 109248/109248 [00:07<00:00, 15160.99it/s]
```

In [21]:

```
preprocessed_titles[13143]
```

Out[21]:

```
'chromebooks coding'
```

1.5 Preparing data for models

In [22]:

```
project_data.columns
```

Out[22]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', 'project_title', 'project_essay_1',
       'project_essay_2', 'project_essay_3', 'project_essay_4',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', '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 [23]:

```
# 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 [24]:

```
# 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 [25]:

```
# you can do the similar thing with state, teacher_prefix and project_grade_category also
vectorizer_1 = CountVectorizer(lowercase=False, binary=True)
vectorizer_1.fit(project_data['school_state'].values)
print(vectorizer_1.get_feature_names())


categories_state_1 = vectorizer_1.transform(project_data['school_state'].values)
print("Shape of matrix after one hot encodig ",categories_state_1.shape)
```

```
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'IA', 'ID', 'IL', 'IN', 'K
S', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM',
'NV', 'NY', 'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA', 'WI', 'WV
', 'WY']
Shape of matrix after one hot encodig  (109248, 51)
```

In [26]:

```
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("-", "_
")
#vectorizer_2 = CountVectorizer(vocabulary=list(word_dict_134.keys()), lowercase=False,
binary=True)
vectorizer_2 = CountVectorizer(lowercase=False, binary=True)
vectorizer_2.fit(project_data['project_grade_category'].values)
print(vectorizer_2.get_feature_names())


categories_grade = vectorizer_2.transform(project_data['project_grade_category'].values)
print("Shape of matrix after one hot encodig ",categories_grade.shape)
```

```
['Grades3_5', 'Grades6_8', 'Grades9_12', 'GradesPreK_2']
Shape of matrix after one hot encodig  (109248, 4)
```

In [27]:

```
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
```

```
#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')


vectorizer_3 = CountVectorizer(lowercase=False, binary=True)
vectorizer_3.fit(project_data['teacher_prefix'].values)
print(vectorizer_3.get_feature_names())


categories_teacher_prefix = vectorizer_3.transform(project_data['teacher_prefix'].values)
print("Shape of matrix after one hot encodig ",categories_teacher_prefix.shape)
```

```
['Dr', 'Mr', 'Mrs', 'Ms', 'Teacher', 'teacher']
Shape of matrix after one hot encodig  (109248, 6)
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [28]:

```
# 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, 16512)
```

In [29]:

```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
vectorizer = CountVectorizer(min_df=10)
title_bow = vectorizer.fit_transform(preprocessed_titles)
print("Shape of matrix after one hot encodig ",title_bow.shape)
```

```
Shape of matrix after one hot encodig  (109248, 3329)
```

1.5.2.2 TFIDF vectorizer

In [30]:

```
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, 16512)
```

In [31]:

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

```
Shape of matrix after one hot encodig  (109248, 3329)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [32]:

```
'''
# 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[32]:

```
'\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 [33]:

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

In [34]:

```
# 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 [01:13<00:00, 1496.32it/s]
```

```
109248
300
```

In [35]:

```
avg_w2v_vectors_1 = []
for sentence in tqdm(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.append(vector)

print(len(avg_w2v_vectors_1))
print(len(avg_w2v_vectors_1[0]))
```

```
100%|█████████████████████████████████| 109248/109248 [00:04<00:00, 26384.43it/s]
```

```
109248
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [36]:

```
# 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 [37]:

```
# average Word2Vec
# compute average word2vec for each review.
# 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 [08:25<00:00, 216.18it/s]
```

```
109248
300
```

In [38]:

```
# Similarly you can vectorize for title also
tfidf_model_1 = TfidfVectorizer()
tfidf_model_1.fit(preprocessed_titles)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_1.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words_1 = set(tfidf_model_1.get_feature_names())
```

In [39]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_1 = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(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):
            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.append(vector)

print(len(tfidf_w2v_vectors_1))
print(len(tfidf_w2v_vectors_1[0]))
```

```
100%|███████████████████████████████████| 109248/109248 [00:09<00:00, 12090.56it/s]
```

```
109248
300
```

1.5.3 Vectorizing Numerical features

In [40]:

```
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 [41]:

```
# 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
```

price_standardized

In [42]:

```
teacher_number_of_previously_posted_projects_scalar = StandardScaler()
teacher_number_of_previously_posted_projects_scalar.fit(project_data['price'].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 =
teacher_number_of_previously_posted_projects_scalar.transform(project_data['teacher_number_of_previ
ously_posted_projects'].values.reshape(-1, 1))
```

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

1.5.4 Merging all the above features

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

In [43]:

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

In [44]:

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

Out[44]:

```
(109248, 16552)
```

In [45]:

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

```
X = hstack((categories_one_hot, sub_categories_one_hot, text_bow, price_standardized))
X.shape

#categorical, numerical features + project_title(BOW)
X1 = hstack((categories_state_1, categories_one_hot, sub_categories_one_hot,
categories_teacher_prefix, categories_grade, price_standardized,
teacher_number_of_previously_posted_projects_standardized, title_bow))

#categorical, numerical features + project_title(TFIDF)
X2 = hstack((categories_state_1, categories_one_hot, sub_categories_one_hot,
categories_teacher_prefix, categories_grade, price_standardized,
teacher_number_of_previously_posted_projects_standardized, title_tfidf))

#categorical, numerical features + project_title(AVG W2V)
X3 = hstack((categories_state_1, categories_one_hot, sub_categories_one_hot,
categories_teacher_prefix, categories_grade, price_standardized,
teacher_number_of_previously_posted_projects_standardized, avg_w2v_vectors_1))

#categorical, numerical features + project_title(TFIDF W2V)
X4 = hstack((categories_state_1, categories_one_hot, sub_categories_one_hot,
categories_teacher_prefix, categories_grade, price_standardized,
teacher_number_of_previously_posted_projects_standardized, tfidf_w2v_vectors_1))
```

In [46]:

```
Y = hstack((categories_state_1, categories_grade.shape, categories_teacher_prefix, title_tfidf, tit
le_bow, text_tfidf, avg_w2v_vectors, avg_w2v_vectors_1, tfidf_w2v_vectors, tfidf_w2v_vectors_1, X,
teacher_number_of_previously_posted_projects_standardized))
Y.shape

ase = project_data['project_is_approved']

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 [47]:

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

Out[47]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_ca
0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	GradesPreK_2
1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades3_5
2	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	2016-04-27 00:46:53	GradesPreK_2
3	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	GradesPreK_2
4	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades3_5

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_ca
5	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-27 01:10:09	Grades3_5
6	95963	p155767	e50367a62524e11fbd2dc79651b6df21	Mrs.	CA	2016-04-27 01:29:58	Grades3_5
7	139722	p182545	22460c54072bd0cf958cc8349fac8b8f	Ms.	CA	2016-04-27 02:02:27	Grades3_5
8	72317	p087808	598621c141cda5fb184ee7e8ccdd3fcc	Ms.	CA	2016-04-27 02:04:15	GradesPreK_2
9	114684	p049177	679f50f18ce50aabcc602d17f7627206	Mrs.	HI	2016-04-27 02:18:58	Grades3_5
10	57854	p099430	4000cfe0c8b2df75a218347c1765e283	Ms.	IL	2016-04-27 07:19:44	GradesPreK_2
11	166022	p120079	8e22592f19b346df505bbdf6144c28d5	Mr.	OH	2016-04-27 07:24:47	Grades9_12
12	79341	p091436	bb2599c4a114d211b3381abe9f899bf8	Mrs.	OH	2016-04-27 07:24:47	GradesPreK_2
13	128817	p239087	11a60ddd63717c59fdd5a13ea92d34aa	Mrs.	KY	2016-04-27 08:02:22	Grades3_5
14	127145	p203619	85c61480f0eaea60734523665a3838b4	Mrs.	SC	2016-04-27 08:06:29	Grades3_5
15	104404	p258140	341dc52d3229176eda913da90b6c19c7	Mrs.	SC	2016-04-27 08:23:26	GradesPreK_2
16	149397	p131036	bf5bf59287e7c676a634a00284596b64	Mrs.	FL	2016-04-27 08:42:52	Grades3_5
17	179302	p199881	82ae813a6e2dc0da592de93861a69561	Mrs.	OH	2016-04-27 08:43:52	Grades3_5
18	50256	p203475	63e9a9f2c9811a247f1aa32ee6f92644	Mrs.	CA	2016-04-27 08:45:34	Grades3_5

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_ca
19	139237	p147271	7f2072d18c67087af27066f60b2c9d85	Mrs.	MO	2016-04-27 08:51:43	Grades9_12
20	146737	p224791	ff5d658932d9ad0d9ebedabea582648e	Mrs.	MI	2016-04-27 08:51:57	GradesPreK_2
21	14427	p058390	578585b8ab7349189837e9618ca0f7f4	Mrs.	NY	2016-04-27 09:02:04	Grades3_5
22	59671	p061990	03093ad866c578b107d5be6957837c5f	Mr.	CA	2016-04-27 09:03:05	Grades9_12
23	148085	p196567	171f782b55614c56213131bcb8d44e06	Mrs.	VA	2016-04-27 09:08:08	Grades3_5
24	12619	p023504	18c82623ff01c59e593f7d81ab11e62c	Ms.	NY	2016-04-27 09:08:34	Grades3_5
25	121622	p138958	57626865698278199f753dc0f8e3ed00	Ms.	GA	2016-04-27 09:13:12	GradesPreK_2
26	135897	p092089	44ab4df75ae4e8b9bb23b818a7a1b1a4	Mrs.	MD	2016-04-27 09:15:30	GradesPreK_2
27	123970	p136762	362f046c8551fa0b2515f99d6e3ce6ea	Mr.	TX	2016-04-27 09:17:34	GradesPreK_2
28	165036	p042345	3c2efbcac105fc8a55df610ed03f4e77	Mrs.	MS	2016-04-27 09:18:49	Grades3_5
29	164738	p248458	40da977f63fb3d85589a063471304b11	Ms.	NJ	2016-04-27 09:33:03	GradesPreK_2
...	...	...	...	...	...	...	...
109218	73173	p011863	e9a57ff541d9965373d9f05baec6dbb9	Mrs.	MD	2017-04-30 20:21:54	GradesPreK_2
109219	63258	p185518	783c9da904d2902781a4205a8a6f2cf2	Ms.	MO	2017-04-30 20:33:50	GradesPreK_2
109220	110157	p093760	7c0bb16f949a62e91151789662b27675	Mrs.	MI	2017-04-30 20:50:01	Grades3_5
109221	180842	p113135	ea758136dee04fab896aac935276161d	Mrs.	NC	2017-04-30	GradesPreK_2

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	20:55:19 Date	project_grade_ca
109222	94252	p095511	d0b4f2709a391b3953bca7e4d0655992	Ms.	OK	2017- 04-30 20:56:44	GradesPreK_2
109223	67820	p117003	a22232bad54f69e12f379fe86f3f8828	Ms.	CA	2017- 04-30 21:03:56	Grades9_12
109224	84481	p140704	e36637824051b8a2edc16c6ec0eb4832	Mr.	CA	2017- 04-30 21:25:13	GradesPreK_2
109225	28930	p079867	6240693c06f02e3bb63e89afa413f379	Mr.	NE	2017- 04-30 21:26:54	Grades3_5
109226	180481	p036737	b2f85df8fe445189b1e56d7b6561adbe	Mrs.	WI	2017- 04-30 21:30:24	GradesPreK_2
109227	169090	p162286	68c376fb8289fafb9831d0c886669fd1	Ms.	OH	2017- 04-30 21:39:35	Grades3_5
109228	28565	p215499	194004c4aee808bcd24deff39b3acdb8	Ms.	IN	2017- 04-30 21:42:14	GradesPreK_2
109229	71940	p038577	e8e0311f1765ef3a427a0c4da811a5fe	Ms.	IL	2017- 04-30 21:42:43	GradesPreK_2
109230	150872	p149431	f308097ab4af3a20ad3d96b13083b9c4	Ms.	LA	2017- 04-30 21:53:11	Grades3_5
109231	20564	p021779	504e698d91890380ff7e278e3918bb2f	Mr.	CA	2017- 04-30 21:53:50	Grades6_8
109232	180953	p075974	3654cb255584baee31fded55e9fa593b	Mrs.	CA	2017- 04-30 22:01:52	Grades3_5
109233	61360	p007550	05677e17e14429f6942245da50bd3da4	Mrs.	CA	2017- 04-30 22:02:26	GradesPreK_2
109234	60690	p243246	1c6ad7948ab442bad6f72fd8ad64dd7f	Mrs.	FL	2017- 04-30 22:07:22	GradesPreK_2
109235	3550	p215525	f09efb73f135c77ed938ca4df6a33ff5	Ms.	GA	2017- 04-30 22:09:11	Grades3_5

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_ca
109236	171039	p173915	be550b77ce85b91080a76df5d3d9bf17	Mr.	NY	2017-04-30 22:18:10	Grades9_12
109237	167772	p198397	9a934dc531c21fd9392e6b70bd1c38ea	Mrs.	IN	2017-04-30 22:24:34	GradesPreK_2
109238	41021	p149303	dc07f461cc1b8767846023734c44cc43	Mrs.	FL	2017-04-30 22:35:13	GradesPreK_2
109239	103254	p173126	05c9cc90376e1d7fde1b7138f2bc4d8d	Ms.	IN	2017-04-30 22:36:34	Grades6_8
109240	19563	p257657	b55850e67a9e5d917958c43082be9e9b	Ms.	CA	2017-04-30 22:42:19	Grades9_12
109241	34853	p067693	63ab3770bef577efb8b738303ef19e54	Ms.	CA	2017-04-30 23:06:36	Grades3_5
109242	175286	p222440	51ffd84df3423f8d5e38943e54b8388e	Mrs.	CT	2017-04-30 23:10:29	GradesPreK_2
109243	45036	p194916	29cf137e5a40b0f141d9fd7898303a5c	Mrs.	HI	2017-04-30 23:11:45	Grades9_12
109244	12610	p162971	22fee80f2078c694c2d244d3ecb1c390	Ms.	NM	2017-04-30 23:23:24	GradesPreK_2
109245	179833	p096829	c8c81a73e29ae3bdd4140be8ad0bea00	Mrs.	IL	2017-04-30 23:25:42	Grades3_5
109246	13791	p184393	65545a295267ad9df99f26f25c978fd0	Mrs.	HI	2017-04-30 23:27:07	Grades9_12
109247	124250	p028318	1fff5a88945be8b2c728c6a85c31930f	Mrs.	CA	2017-04-30 23:45:08	GradesPreK_2

109248 rows × 22 columns

Assignment 3: Apply KNN

```
<li><strong>[Task-1] Apply KNN(brute force version) on these feature sets</strong>
    <ul>
        <li><font color='red'>Set 1</font>: categorical, numerical features + project_title(
BOW) + preprocessed_essay (BOW)</li>
        <li><font color='red'>Set 2</font>: categorical, numerical features + project_title(
        <li><font color='red'>Set 2</font>: categorical, numerical features + project_title(
TFIDF)+  preprocessed_essay (TFIDF)</li>
        <li><font color='red'>Set 3</font>: categorical, numerical features + project_title(
AVG W2V)+  preprocessed_essay (AVG W2V)</li>
        <li><font color='red'>Set 4</font>: categorical, numerical features + project_title(
TFIDF W2V)+  preprocessed_essay (TFIDF W2V)</li>
    </ul>
</li>
<br>
<li><strong>Hyper paramter tuning to find best K</strong>
    <ul>
<li>Find the best hyper parameter which results in the maximum <a
href='https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-
operating-characteristic-curve-roc-curve-and-auc-1/'>AUC</a> value</li>
<li>Find the best hyper paramter using k-fold cross validation (or) simple cross validation
data</li>
<li>Use gridsearch-cv or randomsearch-cv or  write your own for loops to do this task</li>
    </ul>
</li>
<br>
<li>
<strong>Representation of results</strong>
    <ul>
<li>You need to plot the performance of model both on train data and cross validation data
for each hyper parameter, as shown in the figure
<img src='train_cv_auc.JPG' width=300px></li>
<li>Once you find the best hyper parameter, you need to train your model-M using the best h
yper-param. Now, find the AUC on test data and plot the ROC curve on both train and test us
ing model-M.
<img src='train_test_auc.JPG' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a
href='https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-mat
rix-tpr-fpr-fnr-tnr-1/'>confusion matrix</a> with predicted and original labels of test dat
a points
<img src='confusion_matrix.png' width=300px></li>
    </ul>
</li>
<li><strong> [Task-2] </strong>
    <ul>
        <li>Select top 2000 features from feature <font color='red'>Set 2</font> using <a hr
ef='https://scikit-
learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html'>`SelectKBest`
```

and then apply KNN on top of these features</li>

-
  ```
                from sklearn.datasets import load_digits
                from sklearn.feature_selection import SelectKBest, chi2
                X, y = load_digits(return_X_y=True)
                X.shape
                X_new = SelectKBest(chi2, k=20).fit_transform(X, y)



                X_new.shape
                ========
                output:
                (1797, 64)
                (1797, 20)
                </pre>
            </li>
            <li>Repeat the steps 2 and 3 on the data matrix after feature selection</li>
        </ul>
    </li>
    <br>
    <li><strong>Conclusion</strong>
  ```

```
        <ul>
    <li>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<a hre
    f='http://zetcode.com/python/prettytable/'> link</a>
        <img src='summary.JPG' width=400px>
    </li>
        </ul>



</ol>
```

Note: Data Leakage

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

2. K Nearest Neighbor

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

In [48]:

```python
# 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)

# split the train data set into cross validation train and cross validation test
#X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(X_train, y_train, test_size=0.3)

#X_cv
#project_data.shape
```

In the Above cell i have split the data into train, cross validation and test for project_data and i created a separate matrix Y for the project is approved column, and split that as well into train, cv and test. The new categories preprocessed essays and preprocessed titles were added directly to project_data before splitting.

2.2 Make Data Model Ready: encoding numerical, categorical features

In [49]:

```
# 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

#feature names encoding for X_train
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
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)
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, binary=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'].values)
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, binary=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-645918fcc2fc
#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-attribute-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)
#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)
```

In the Above cell, i have performed one hot encoding for clean categories, clean subcategories, school state, project grade category and teacher prefix, for train, cv and test data. All categories were encoded separately. ie school state was encoded separately for train, cv, test.

2.3 Make Data Model Ready: encoding eassay, and project_title

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
# 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)
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)
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)
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)

#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, 14377)
Shape of matrix after one hot encodig  (32775, 14377)
Shape of matrix after one hot encodig  (76473, 2718)
Shape of matrix after one hot encodig  (32775, 2718)
Shape of matrix after one hot encodig  (76473, 14377)
Shape of matrix after one hot encodig  (32775, 14377)
Shape of matrix after one hot encodig  (76473, 2718)
Shape of matrix after one hot encodig  (32775, 2718)
```

In the above block of code, i have done vectorization for preprocessed essays and preprocessed titles using BOW and TFIDF respectively for train, cv and test data separately.

In [51]:

```
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 [00:45<00:00, 1676.68it/s]
```

```
76473
300
```

```
100%|█████████████████████████████████| 32775/32775 [00:02<00:00, 14778.65it/s]
```

```
32775
300
```

In the above cell, the vectorization of preprocessed titles was done using Avg W2V for train, cv and test data.

In [52]:

```
# 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[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.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[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.append(vector)

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

```
100%|██████████████████████████████████| 76473/76473 [00:06<00:00, 11188.32it/s]
```

```
76473
300
```

```
100%|██████████████████████████████████| 32775/32775 [00:02<00:00, 11041.68it/s]
```

```
32775
300
```

In the above cell, the vectorization of preprocessed titles was done using TFIDF W2V for train, cv and test data.

In [53]:

```
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))
#X1_train = hstack((categories_one_hot_train, sub_categories_one_hot_train,
categories_state_1_train, categories_grade_train, categories_teacher_prefix_train,
price_standardized_train, teacher_number_of_previously_posted_projects_standardized_train, title_b
ow_tr))
#X1_cv = hstack((categories_one_hot_cv, sub_categories_one_hot_cv, categories_state_1_cv, categori
es_grade_cv, categories_teacher_prefix_cv, price_standardized_cv,
teacher_number_of_previously_posted_projects_standardized_cv, title_bow_cv))
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))

#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))
#X2_train = hstack((categories_one_hot_train, sub_categories_one_hot_train,
categories_state_1_train, categories_grade_train, categories_teacher_prefix_train,
price_standardized_train, teacher_number_of_previously_posted_projects_standardized_train, title_t
fidf_train))
#X2_cv = hstack((categories_one_hot_cv, sub_categories_one_hot_cv, categories_state_1_cv, categori
es_grade_cv, categories_teacher_prefix_cv, price_standardized_cv,
teacher_number_of_previously_posted_projects_standardized_cv, title_tfidf_cv))
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))

#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))
#X3_train = hstack((categories_one_hot_train, sub_categories_one_hot_train,
categories_state_1_train, categories_grade_train, categories_teacher_prefix_train,
price_standardized_train, teacher_number_of_previously_posted_projects_standardized_train, avg_w2v
_vectors_1_train))
#X3_cv = hstack((categories_one_hot_cv, sub_categories_one_hot_cv, categories_state_1_cv, categori
es_grade_cv, categories_teacher_prefix_cv, price_standardized_cv,
teacher_number_of_previously_posted_projects_standardized_cv, avg_w2v_vectors_1_cv))
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))

#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))
#X4_train = hstack((categories_one_hot_train, sub_categories_one_hot_train,
categories_state_1_train, categories_grade_train, categories_teacher_prefix_train,
price_standardized_train, teacher_number_of_previously_posted_projects_standardized_train, tfidf_w
2v_vectors_1_train))
#X4_cv = hstack((categories_one_hot_cv, sub_categories_one_hot_cv, categories_state_1_cv, categori
es_grade_cv, categories_teacher_prefix_cv, price_standardized_cv,
teacher_number_of_previously_posted_projects_standardized_cv, tfidf_w2v_vectors_1_cv))
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))
```

In [54]:

```
X1_train_tr_1 = X1_train_tr
X2_train_tr_2 = X2_train_tr
X3_train_tr_3 = X3_train_tr
X4_train_tr_4 = X4_train_tr

X1_test_1 = X1_test
X2_test_2 = X2_test
X3_test_3 = X3_test
X4_test_4 = X4_test

y_train_1 = y_train
y_test_1 = y_test
```

In [55]:

```
X1_train_tr_1
```

Out[55]:

```
<76473x2820 sparse matrix of type '<class 'numpy.float64'>'
 with 904064 stored elements in COOrdinate format>
```

In [56]:

```
X1_test_1
```

Out[56]:

```
<32775x2820 sparse matrix of type '<class 'numpy.float64'>'
 with 386078 stored elements in COOrdinate format>
```

In [57]:

```
y_train_1
```

Out[57]:

```
65565     0
38689     1
40405     1
83350     1
16217     1
```

```
29992      0
43764      1
102669     1
21159      1
50961      1
101911     1
54529      0
92766      1
48184      1
4027       1
19248      1
51672      0
103687     1
1118       0
40015      1
27918      1
51671      1
77009      0
35912      1
44162      1
54951      1
64801      1
76376      1
88027      1
7321       1
          ..
11911      1
95123      1
107220     1
90009      1
23695      1
25692      1
14986      1
86619      1
85107      1
83127      1
61142      1
68236      1
79639      0
100121     1
45742      1
30847      1
2514       0
27519      1
56510      1
64104      1
11841      0
62418      1
41989      1
19461      1
35020      1
3358       1
39338      1
98928      1
51988      1
36038      0
Name: project_is_approved, Length: 76473, dtype: int64
```

In the Above few cells, i have vectorized the numerical features namely price and teacher_number_of_previously_posted_projects and then standardized them as well.

After doing that, i compiled all the previous vectorized features into sets of sparse matrix, classified on basis of whether they belong to train, CV and test and also on basis of what kind of vectorization we perform on preprocessed_titles. I have not created any vectors that use preprocessed_essays because the files will take too long to run on my laptop

In [58]:

```
#https://chrisalbon.com/machine_learning/feature_engineering/dimensionality_reduction_on_sparse_fea
_matrix/
# Load libraries
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import TruncatedSVD
from scipy.sparse import csr_matrix
from sklearn import datasets
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt
import numpy as np

# Create a TSVD
tsvd = TruncatedSVD(n_components=10)


# Conduct TSVD on sparse matrix

#for BOW
X1_train_tr_tsvd = tsvd.fit(X1_train_tr).transform(X1_train_tr)
#X1_train_tsvd = tsvd.fit(X1_train).transform(X1_train)
#X1_cv_tsvd = tsvd.fit(X1_cv).transform(X1_cv)
X1_test_tsvd = tsvd.fit(X1_test).transform(X1_test)

#for TFIDF
X2_train_tr_tsvd = tsvd.fit(X2_train_tr).transform(X2_train_tr)
#X2_train_tsvd = tsvd.fit(X2_train).transform(X2_train)
#X2_cv_tsvd = tsvd.fit(X2_cv).transform(X2_cv)
X2_test_tsvd = tsvd.fit(X2_test).transform(X2_test)

#for AVG W2V
X3_train_tr_tsvd = tsvd.fit(X3_train_tr).transform(X3_train_tr)
#X3_train_tsvd = tsvd.fit(X3_train).transform(X3_train)
#X3_cv_tsvd = tsvd.fit(X3_cv).transform(X3_cv)
X3_test_tsvd = tsvd.fit(X3_test).transform(X3_test)

#For TFIDF W2V
X4_train_tr_tsvd = tsvd.fit(X4_train_tr).transform(X4_train_tr)
#X4_train_tsvd = tsvd.fit(X4_train).transform(X4_train)
#X4_cv_tsvd = tsvd.fit(X4_cv).transform(X4_cv)
X4_test_tsvd = tsvd.fit(X4_test).transform(X4_test)
```

In [59]:

```
X1_train_tr_tsvd.shape
```

Out[59]:

```
(76473, 10)
```

In [60]:

```
X1_test_tsvd.shape
```

Out[60]:

```
(32775, 10)
```

Using Truncated SVD, i have reduced the dimensions of the sparse matrixes i got previously to 10 for all categories.

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

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

In [61]:

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

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

2.4.1 Applying KNN brute force on BOW, SET 1

In [62]:

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

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

In [63]:

```
#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)
k_range = list(range(1, 106, 8))
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 = dict(n_neighbors=k_range)
print(param_grid)

# instantiate model
knn = KNeighborsClassifier(n_neighbors=5)
```

```
[1, 9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89, 97, 105]
{'n_neighbors': [1, 9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89, 97, 105]}
```

In [64]:

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

In [65]:

```
X1_train_tr_tsvd
```

Out[65]:

```
array([[ 0.9424472 , -0.2286163 , -0.57473837, ...,  0.02670447,
         0.524051  , -0.40770261],
       [ 0.95785125,  0.03011646, -0.3759    , ..., -0.57443066,
         0.4990248 , -0.18490555],
       [ 1.51095302,  0.71237601,  0.16559955, ..., -0.16056395,
        -0.25080033, -0.47329513],
       ...,
       [ 1.92087824,  0.21278575, -0.3664222 , ..., -0.30198953,
        -0.53139956, -0.31338835],
       [ 1.29903157, -0.17583013, -0.75843452, ...,  0.23495876,
         0.6652416 , -0.8480538 ],
       [ 1.12010315,  1.35249828,  1.07643886, ..., -0.01816888,
         0.1658779 , -0.37216289]])
```

In [66]:

```
# fit the grid with data
grid.fit(X1_train_tr_tsvd, y_train)
```

Out[66]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='warn', n_jobs=None,
             param_grid={'n_neighbors': [1, 9, 17, 25, 33, 41, 49, 57, 65, 73,
                                         81, 89, 97, 105]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [67]:

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

Out[67]:

```
{'mean_fit_time': array([0.15857466, 0.12126414, 0.12326288, 0.11626641, 0.10960348,
        0.11926524, 0.10460647, 0.1069382 , 0.10527301, 0.10793845,
        0.10494002, 0.11693231, 0.11893185, 0.10860348]),
 'std_fit_time': array([0.07606686, 0.01676999, 0.01437567, 0.00623288, 0.00376956,
        0.01007208, 0.00956295, 0.00786916, 0.00543371, 0.00431897,
        0.00535047, 0.01156916, 0.02145086, 0.00524641]),
 'mean_score_time': array([ 1.1030345 ,  2.26336892,  3.19250075,  3.9427379 ,  4.73394942,
         5.28896395,  6.04852772,  6.63952295,  7.51735234,  7.75088342,
         8.50478681,  9.53819235, 10.14617674, 10.35538999]),
 'std_score_time': array([0.21451592, 0.21297624, 0.24505528, 0.46639329, 0.54991825,
        0.64306872, 0.71447512, 0.94233851, 1.27444262, 0.95594749,
        1.06307873, 1.03080481, 1.20522427, 1.46315537]),
 'param_n_neighbors': masked_array(data=[1, 9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89, 97, 105],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'n_neighbors': 1},
  {'n_neighbors': 9},
  {'n_neighbors': 17},
  {'n_neighbors': 25},
  {'n_neighbors': 33},
  {'n_neighbors': 41},
  {'n_neighbors': 49},
  {'n_neighbors': 57},
  {'n_neighbors': 65},
  {'n_neighbors': 73},
  {'n_neighbors': 81},
  {'n_neighbors': 89},
  {'n_neighbors': 97},
  {'n_neighbors': 105}],
 'split0_test_score': array([0.51804197, 0.55984121, 0.57220785, 0.57604374, 0.5824806 ,
        0.58686829, 0.58975249, 0.59062968, 0.59036207, 0.59327119,
        0.59424375, 0.59306668, 0.59530809, 0.59489712]),
 'split1_test_score': array([0.52147394, 0.55550141, 0.5702766 , 0.58156758, 0.58705105,
        0.58789344, 0.58927211, 0.59294805, 0.59445815, 0.59562611,
        0.59597571, 0.59803217, 0.59869786, 0.60008575]),
 'split2_test_score': array([0.5235585 , 0.55714437, 0.5697334 , 0.57778798, 0.58342393,
        0.58496268, 0.58597642, 0.5883207 , 0.58969898, 0.59062446,
        0.59203817, 0.59321152, 0.59489841, 0.59471424]),
 'mean_test_score': array([0.52102473, 0.5574957 , 0.57073932, 0.57846641, 0.58431851,
        0.58657483, 0.58833373, 0.59063284, 0.59150641, 0.59317396,
        0.59408591, 0.59477012, 0.59630146, 0.59656571]),
 'std_test_score': array([0.0022744 , 0.00178906, 0.00106186, 0.00230558, 0.0019702 ,
        0.00121433, 0.00167831, 0.00188909, 0.00210468, 0.00204305,
        0.00161135, 0.00230737, 0.00170274, 0.00249017]),
 'rank_test_score': array([14, 13, 12, 11, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1]),
 'split0_train_score': array([1.        , 0.77791638, 0.72565553, 0.69975173, 0.68312331,
        0.67223899, 0.66508397, 0.65825392, 0.65367076, 0.65052633,
        0.64691847, 0.64444529, 0.64153855, 0.6387798 ]),
 'split1_train_score': array([0.99998844, 0.77824099, 0.72490919, 0.69684393, 0.68370335,
        0.67292623, 0.66495832, 0.65845318, 0.65318308, 0.64860992,
        0.64497475, 0.64192913, 0.63974797, 0.63614248]),
 'split2_train_score': array([0.99998844, 0.77935709, 0.72479282, 0.69877383, 0.68282269,
        0.67315682, 0.66631225, 0.65973073, 0.65629468, 0.65280486,
        0.6478319 , 0.64565837, 0.64282283, 0.6400678 ]),
 'mean_train_score': array([0.9999923 , 0.77850482, 0.72511918, 0.69845649, 0.68321645,
        0.67277401, 0.66545151, 0.65881261, 0.65438284, 0.65064704,
        0.64657504, 0.64401093, 0.64136978, 0.63833003]),
 'std_train_score': array([5.44812566e-06, 6.17042716e-04, 3.82220570e-04, 1.20812542e-03,
        3.65511689e-04, 3.89854352e-04, 6.10789953e-04, 6.54286285e-04,
        1.36645419e-03, 1.71470302e-03, 1.19143857e-03, 1.55312713e-03,
        1.26096606e-03, 1.63376026e-03])}
```

In [68]:

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

Out[68]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_neighbors	params	split0_test_score	s
0	0.158575	0.076067	1.103034	0.214516	1	{'n_neighbors': 1}	0.518042	0
1	0.121264	0.016770	2.263369	0.212976	9	{'n_neighbors': 9}	0.559841	0
2	0.123263	0.014376	3.192501	0.245055	17	{'n_neighbors': 17}	0.572208	0
3	0.116266	0.006233	3.942738	0.466393	25	{'n_neighbors': 25}	0.576044	0
4	0.109603	0.003770	4.733949	0.549918	33	{'n_neighbors': 33}	0.582481	0
5	0.119265	0.010072	5.288964	0.643069	41	{'n_neighbors': 41}	0.586868	0
6	0.104606	0.009563	6.048528	0.714475	49	{'n_neighbors': 49}	0.589752	0
7	0.106938	0.007869	6.639523	0.942339	57	{'n_neighbors': 57}	0.590630	0
8	0.105273	0.005434	7.517352	1.274443	65	{'n_neighbors': 65}	0.590362	0
9	0.107938	0.004319	7.750883	0.955947	73	{'n_neighbors': 73}	0.593271	0
10	0.104940	0.005350	8.504787	1.063079	81	{'n_neighbors': 81}	0.594244	0
11	0.116932	0.011569	9.538192	1.030805	89	{'n_neighbors': 89}	0.593067	0
12	0.118932	0.021451	10.146177	1.205224	97	{'n_neighbors': 97}	0.595308	0
13	0.108603	0.005246	10.355390	1.463155	105	{'n_neighbors': 105}	0.594897	0

In [69]:

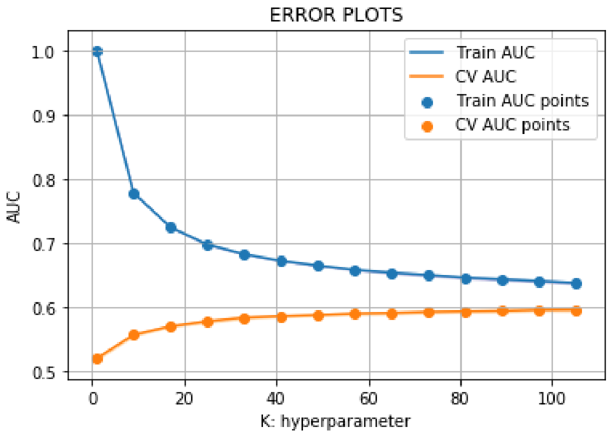
```
# examine the best model

# 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_)
```

```
0.5965657079658473
{'n_neighbors': 105}
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=105, p=2,
                     weights='uniform')
```

In [70]:

```
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']

plt.plot(k_range, train_auc1, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(k_range, train_auc1 - train_auc_std1,train_auc1 + train_auc_std1,alpha=0.2,c
olor='darkblue')

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

plt.scatter(k_range, train_auc1, label='Train AUC points')
plt.scatter(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()
```

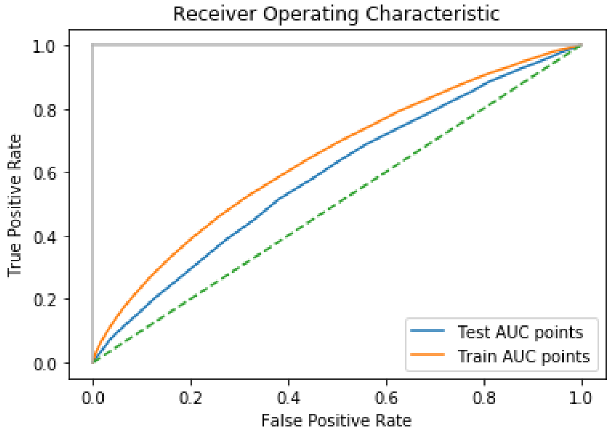


In [71]:

```
Trained_model_BOW = KNeighborsClassifier(n_neighbors=105, metric='minkowski', n_jobs=-1).fit(X1_tra
in_tr_tsvd, y_train)
```

In [72]:

```
# Get predicted probabilities
y_score_test = Trained_model_BOW.predict_proba(X1_test_tsvd)[:,1]
y_score_train = Trained_model_BOW.predict_proba(X1_train_tr_tsvd)[:,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 [73]:

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

roc_auc_score(y_test, y_score_test)
```

Out[73]:

```
0.5883562524916512
```

In [74]:

```
#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_tsvd)

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

```
C:\Users\utsav94\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1437:
UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.

C:\Users\utsav94\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1437:
UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.

C:\Users\utsav94\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1437:
UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      4963
           1       0.85      1.00      0.92     27812

    accuracy                           0.85     32775
   macro avg       0.42      0.50      0.46     32775
weighted avg       0.72      0.85      0.78     32775
```

In [75]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

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

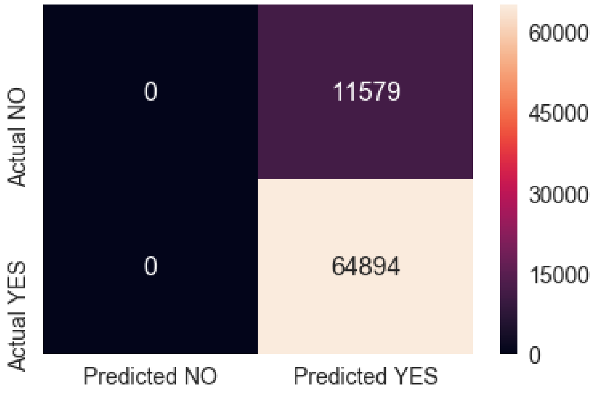
In [76]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(threshold1, false_positive_rate1, true_positive_rate1)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test, best_t)))
```

```
the maximum value of tpr*(1-fpr) 0.36211023495787625 for threshold 0.848
Train confusion matrix
[[11579     0]
 [    0 64894]]
Test confusion matrix
[[ 4963     0]
 [    0 27812]]
```

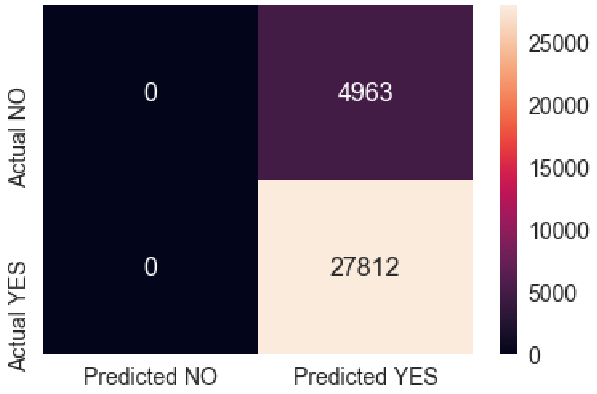
In [77]:

```
#https://www.kaggle.com/shashank49/donors-choose-knn

#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_tsvd, y_train)
```



In [78]:

```
get_confusion_matrix(Trained_model_BOW, X1_test_tsvd, y_test)
```



2.4.2 Applying KNN brute force on TFIDF, SET 2

In [79]:

```
#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.metrics import roc_curve, auc
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import GridSearchCV
# define the parameter values that should be searched
# for python 2, k_range = range(1, 31)
k_range = list(range(1, 106, 8))
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 = dict(n_neighbors=k_range)
print(param_grid)

# instantiate model
knn = KNeighborsClassifier(n_neighbors=5)
```

```
[1, 9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89, 97, 105]
{'n_neighbors': [1, 9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89, 97, 105]}
```

In [80]:

```
#instantiate the grid
grid2 = GridSearchCV(knn, param_grid, cv=3, scoring='roc_auc', n_jobs=-1, return_train_score=True)
```

In [81]:

```
# fit the grid with data
grid2.fit(X2_train_tr_tsvd, y_train)
```

Out[81]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='warn', n_jobs=-1,
             param_grid={'n_neighbors': [1, 9, 17, 25, 33, 41, 49, 57, 65, 73,
                                       81, 89, 97, 105]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [82]:

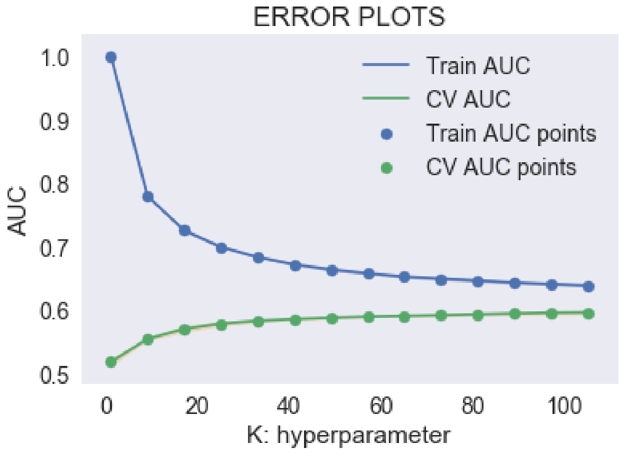
```
# view the complete results (list of named tuples)
grid2.cv_results_
```

Out[82]:

```
{'mean_fit_time': array([0.26917823, 0.28517   , 0.2718447 , 0.26518099, 0.29882733,
        0.27650857, 0.27717384, 0.28783377, 0.26451532, 0.28983331,
        0.26018413, 0.27051083, 0.28683575, 0.278174  ]),
 'std_fit_time': array([0.04503997, 0.0212374 , 0.03622762, 0.02902735, 0.0288623 ,
        0.02780916, 0.04133225, 0.06187403, 0.04814863, 0.06843401,
        0.04616387, 0.02922142, 0.0481665 , 0.0356287 ]),
 'mean_score_time': array([ 1.78497624,  4.28254128,  6.00955192,  8.01806466,  9.44424685,
        10.61957232, 12.11737855, 13.64150445, 14.48568583, 15.75029453,
        16.98958333, 17.95003152, 19.89424833, 19.72967712]),
 'std_score_time': array([0.04174506, 0.44630101, 0.61355569, 1.12010078, 1.25910155,
        1.08097244, 1.55402123, 1.64825978, 1.93500375, 1.92981542,
        2.251677  , 2.26104994, 2.7315702 , 3.77746836]),
 'param_n_neighbors': masked_array(data=[1, 9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89, 97, 105],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'n_neighbors': 1},
  {'n_neighbors': 9},
  {'n_neighbors': 17},
  {'n_neighbors': 25},
  {'n_neighbors': 33},
  {'n_neighbors': 41},
  {'n_neighbors': 49},
  {'n_neighbors': 57},
  {'n_neighbors': 65},
  {'n_neighbors': 73},
  {'n_neighbors': 81},
  {'n_neighbors': 89},
  {'n_neighbors': 97},
  {'n_neighbors': 105}],
 'split0_test_score': array([0.51061666, 0.55683171, 0.57561391, 0.58069685, 0.58272157,
        0.58590987, 0.58872913, 0.5896115 , 0.59043633, 0.5923742 ,
        0.59418421, 0.59230174, 0.59279253, 0.5927001 ]),
 'split1_test_score': array([0.521262  , 0.55367542, 0.57187684, 0.58120928, 0.58561362,
        0.5869446 , 0.58907536, 0.58999426, 0.59226279, 0.59374109,
        0.59468477, 0.59813589, 0.59928677, 0.60010836]),
 'split2_test_score': array([0.52267526, 0.55295861, 0.56346138, 0.57377758, 0.58030321,
        0.58405085, 0.58545542, 0.58882373, 0.58936106, 0.58932429,
        0.59075664, 0.59298103, 0.59500265, 0.59567134]),
 'mean_test_score': array([0.51818448, 0.55448863, 0.57031754, 0.57856133, 0.5828795 ,
        0.58563513, 0.58775334, 0.58947651, 0.59068674, 0.59181324,
        0.59320858, 0.59447288, 0.59569395, 0.59615989]),
 'std_test_score': array([0.00538243, 0.0016825 , 0.0050823 , 0.00338899, 0.00217082,
        0.00119722, 0.00163097, 0.0004873 , 0.00119777, 0.00184625,
        0.00174574, 0.00260494, 0.00269597, 0.0030441 ]),
 'rank_test_score': array([14, 13, 12, 11, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1]),
 'split0_train_score': array([1.        , 0.77840463, 0.72341684, 0.69971389, 0.68418501,
        0.67235972, 0.66418632, 0.65787437, 0.65314747, 0.64851442,
        0.64612622, 0.64370492, 0.64089763, 0.63870591]),
 'split1_train_score': array([0.99998844, 0.77910145, 0.72389583, 0.69787001, 0.68216955,
        0.67042017, 0.66114243, 0.65595198, 0.6497826 , 0.64771267,
        0.64403479, 0.64056509, 0.63869814, 0.63588114]),
 'split2_train_score': array([0.99998844, 0.77975883, 0.72865353, 0.70012906, 0.68345817,
        0.67266107, 0.66577422, 0.65947101, 0.65424507, 0.65170487,
        0.64890599, 0.64521891, 0.64271953, 0.64108256]),
 'mean_train_score': array([0.9999923 , 0.7790883 , 0.72532207, 0.69923765, 0.68327091,
        0.67181366, 0.66370099, 0.65776578, 0.65239171, 0.64931065,
        0.64635567, 0.64316297, 0.64077177, 0.63855654]),
 'std_train_score': array([5.44812566e-06, 5.52930551e-04, 2.36379997e-03, 9.81811403e-04,
        8.33394493e-04, 9.92991060e-04, 1.92180870e-03, 1.43868819e-03,
        1.89855926e-03, 1.72431847e-03, 1.99526723e-03, 1.93817646e-03,
        1.64413512e-03, 2.12609587e-03])}
```

In [83]:

```
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']


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

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

plt.scatter(k_range, train_auc2, label='Train AUC points')
plt.scatter(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()
```



In [84]:

```
# 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_)
```

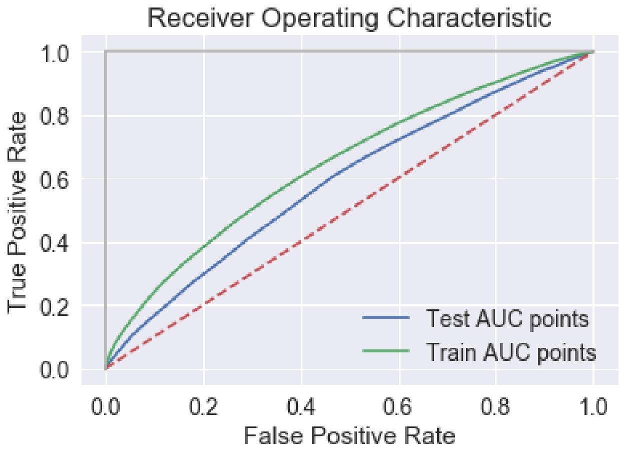
```
0.5961598938676921
{'n_neighbors': 105}
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=105, p=2,
                     weights='uniform')
```

In [85]:

```
Trained_model_TFIDF = KNeighborsClassifier(n_neighbors=97, metric='minkowski', n_jobs=-1).fit(X2_tr
ain_tr_tsvd, y_train)
```

In [86]:

```
# Get predicted probabilities
y_score_test2 = Trained_model_TFIDF.predict_proba(X2_test_tsvd)[:,1]
y_score_train2 = Trained_model_TFIDF.predict_proba(X2_train_tr_tsvd)[:,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 [87]:

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

roc_auc_score(y_test, y_score_test2)
```

Out[87]:

```
0.5892053808567406
```

In [88]:

```
#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_tsvd)

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

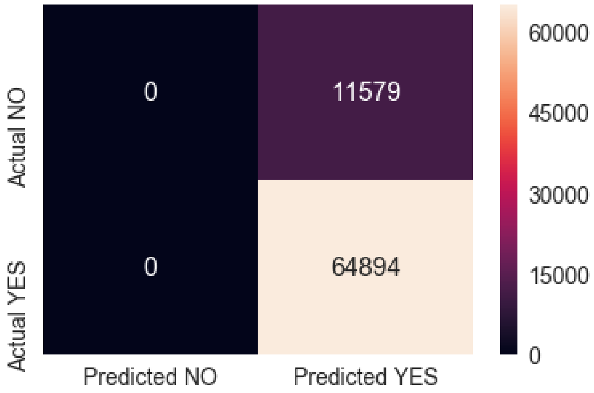
```
C:\Users\utsav94\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1437:
UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.

C:\Users\utsav94\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1437:
UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.

C:\Users\utsav94\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1437:
UndefinedMetricWarning:
```

```
UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      4963
           1       0.85      1.00      0.92     27812

    accuracy                           0.85     32775
   macro avg       0.42      0.50      0.46     32775
weighted avg       0.72      0.85      0.78     32775
```

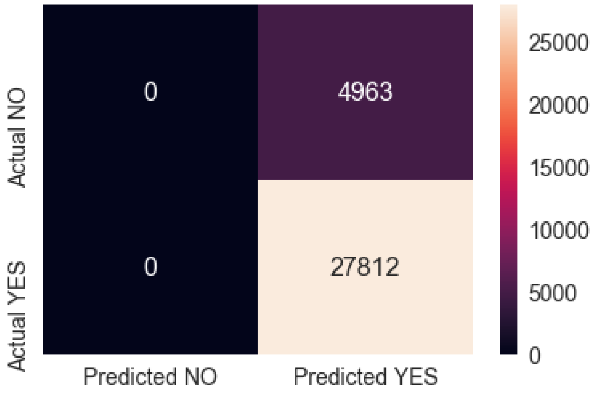
In [89]:

```
get_confusion_matrix(Trained_model_TFIDF, X2_train_tr_tsvd, y_train)
```



In [90]:

```
get_confusion_matrix(Trained_model_TFIDF, X2_test_tsvd, y_test)
```



2.4.3 Applying KNN brute force on AVG W2V, SET 3

In [91]:

```
#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

# define the parameter values that should be searched
# for python 2, k_range = range(1, 31)
k_range = list(range(1, 106, 8))
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 = dict(n_neighbors=k_range)
print(param_grid)

# instantiate model
knn = KNeighborsClassifier(n_neighbors=5)
```

```
[1, 9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89, 97, 105]
{'n_neighbors': [1, 9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89, 97, 105]}
```

In [92]:

```
#instantiate the grid
grid3 = GridSearchCV(knn, param_grid, cv=3, scoring='roc_auc', return_train_score=True)
```

In [93]:

```
# fit the grid with data
grid3.fit(X3_train_tr_tsvd, y_train)
```

Out[93]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='warn', n_jobs=None,
             param_grid={'n_neighbors': [1, 9, 17, 25, 33, 41, 49, 57, 65, 73,
                                         81, 89, 97, 105]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [94]:

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

Out[94]:

```
{'mean_fit_time': array([0.10827001, 0.11693279, 0.12026437, 0.11293578, 0.11160247,
        0.10893742, 0.11393531, 0.10660553, 0.10293976, 0.10627206,
        0.09827701, 0.10394073, 0.10194016, 0.10893766]),
 'std_fit_time': array([0.00329869, 0.01218586, 0.00188509, 0.00355674, 0.00047081,
        0.0053511 , 0.01348259, 0.00703593, 0.0063731 , 0.00188391,
        0.00094235, 0.00859741, 0.00081653, 0.01563227]),
 'mean_score_time': array([ 7.16688697, 15.19294802, 19.34389782, 21.86811455, 24.65618237,
        26.34488026, 26.40174643, 27.70576429, 29.50273387, 29.73093589,
        30.53813966, 31.66682474, 33.76162314, 35.02123364]),
 'std_score_time': array([0.30972234, 0.44915917, 0.63259988, 0.41471046, 0.44013088,
        0.42793398, 0.46238575, 0.66251034, 0.24266688, 0.84883527,
        0.90785472, 0.82614175, 0.81491524, 0.50116596]),
 'param_n_neighbors': masked_array(data=[1, 9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89, 97, 105],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'n_neighbors': 1},
  {'n_neighbors': 9},
  {'n_neighbors': 17},
  {'n_neighbors': 25},
  {'n_neighbors': 33},
  {'n_neighbors': 41},
  {'n_neighbors': 49},
  {'n_neighbors': 57},
  {'n_neighbors': 65},
  {'n_neighbors': 73},
  {'n_neighbors': 81},
  {'n_neighbors': 89},
  {'n_neighbors': 97},
  {'n_neighbors': 105}],
 'split0_test_score': array([0.51037493, 0.55226862, 0.5695589 , 0.58013667, 0.58520157,
        0.58925704, 0.59124768, 0.59336651, 0.59378507, 0.59537932,
        0.59776174, 0.60111717, 0.60183742, 0.60439851]),
 'split1_test_score': array([0.515303  , 0.55755415, 0.57062862, 0.58263282, 0.59241263,
        0.59289106, 0.59608829, 0.60005213, 0.60065853, 0.60217784,
        0.60352179, 0.60436825, 0.60543153, 0.60608063]),
 'split2_test_score': array([0.51222956, 0.55415713, 0.57001074, 0.57959731, 0.58407478,
        0.59018364, 0.594175  , 0.59596811, 0.59878764, 0.59979381,
        0.60015034, 0.60085379, 0.60369028, 0.60454979]),
 'mean_test_score': array([0.51263581, 0.55465994, 0.57006608, 0.58078894, 0.58722968,
        0.59077723, 0.59383695, 0.59646222, 0.59774368, 0.59911693,
        0.60047793, 0.60211307, 0.60365305, 0.60500964]),
 'std_test_score': array([0.0020323 , 0.00218693, 0.00043847, 0.00132228, 0.00369366,
        0.00154182, 0.00199059, 0.00275169, 0.00290157, 0.00281647,
        0.00236293, 0.00159827, 0.00146754, 0.00075982]),
 'rank_test_score': array([14, 13, 12, 11, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1]),
 'split0_train_score': array([1.        , 0.77780327, 0.72612112, 0.700447  , 0.68470848,
        0.67508859, 0.66940346, 0.66260104, 0.65787006, 0.65550852,
        0.65386194, 0.65065793, 0.64956102, 0.64668065]),
 'split1_train_score': array([1.        , 0.77357298, 0.72583403, 0.70061555, 0.68494444,
        0.67579176, 0.66684723, 0.66206279, 0.65805978, 0.65459036,
        0.65142503, 0.64902516, 0.64674028, 0.64538106]),
 'split2_train_score': array([0.99993523, 0.77323868, 0.72276526, 0.69892828, 0.68451025,
        0.67338093, 0.66673255, 0.66300679, 0.6588397 , 0.65453704,
        0.65257696, 0.64980159, 0.64859647, 0.6477069 ]),
 'mean_train_score': array([0.99997841, 0.77487164, 0.7249068 , 0.69999694, 0.68472106,
        0.67475376, 0.66766108, 0.66255688, 0.65825651, 0.65487864,
        0.65262131, 0.64982823, 0.64829925, 0.64658953]),
 'std_train_score': array([3.05313809e-05, 2.07746111e-03, 1.51882879e-03, 7.58786124e-04,
        1.77481574e-04, 1.01229307e-03, 1.23293812e-03, 3.86651976e-04,
        4.19585026e-04, 4.45925704e-04, 9.95359934e-04, 6.66841825e-04,
        1.17058182e-03, 9.51703032e-04])}
```

In [95]:

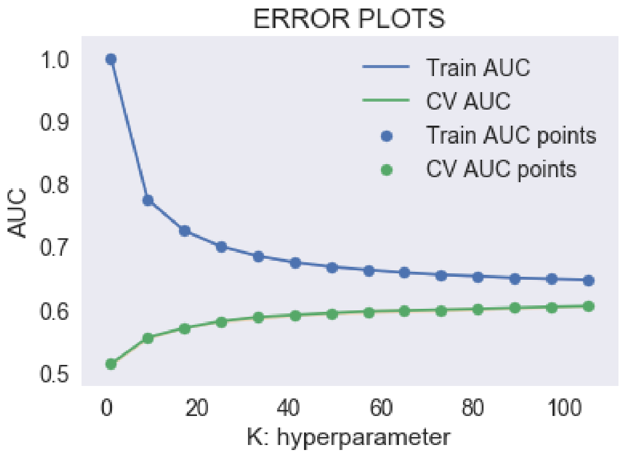
```
# examine the best model

# 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_)
```

```
0.6050096437859052
{'n_neighbors': 105}
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=105, p=2,
                     weights='uniform')
```

In [96]:

```
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']


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

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

```
plt.scatter(k_range, train_auc3, label='Train AUC points')
plt.scatter(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()
```

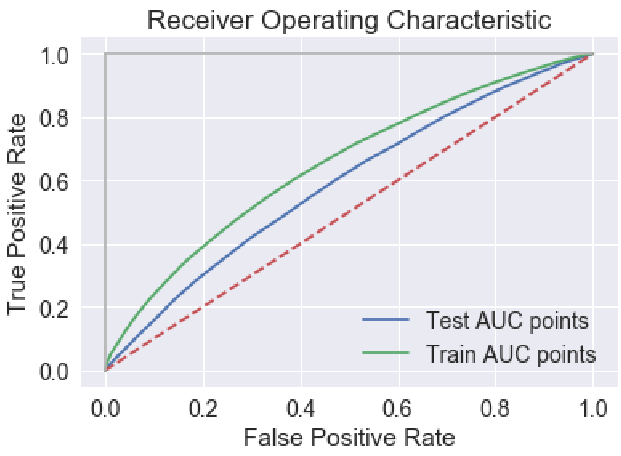


In [97]:

```
Trained_model_AVGW2V = KNeighborsClassifier(n_neighbors=105, metric='minkowski', n_jobs=-1).fit(X3_
train_tr_tsvd, y_train)
```

In [98]:

```
# Get predicted probabilities
y_score_test3 = Trained_model_AVGW2V.predict_proba(X3_test_tsvd)[:,1]
y_score_train3 = Trained_model_AVGW2V.predict_proba(X3_train_tr_tsvd)[:,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 [99]:

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

roc_auc_score(y_test, y_score_test3)
```

Out[99]:

```
0.5890108085609433
```

In [100]:

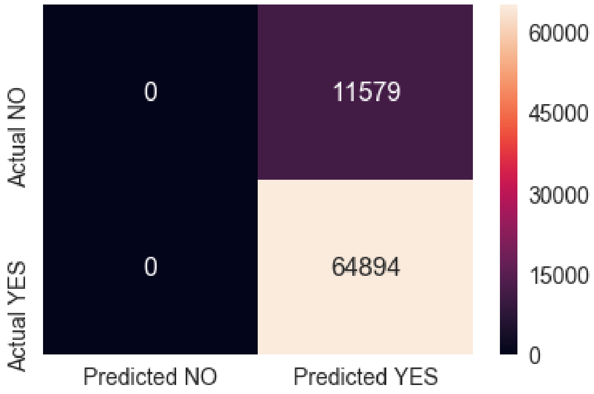
```
#https://chrisalbon.com/machine_learning/model_evaluation/generate_text_reports_on_performance/
from sklearn.metrics import classification_report

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

# Train model and make predictions
y_hat_3 = Trained_model_AVGW2V.predict(X3_test_tsvd)

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

```
C:\Users\utsav94\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1437:
UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.

C:\Users\utsav94\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1437:
UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.

C:\Users\utsav94\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1437:
UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      4963
           1       0.85      1.00      0.92     27812

    accuracy                           0.85     32775
   macro avg       0.42      0.50      0.46     32775
weighted avg       0.72      0.85      0.78     32775
```

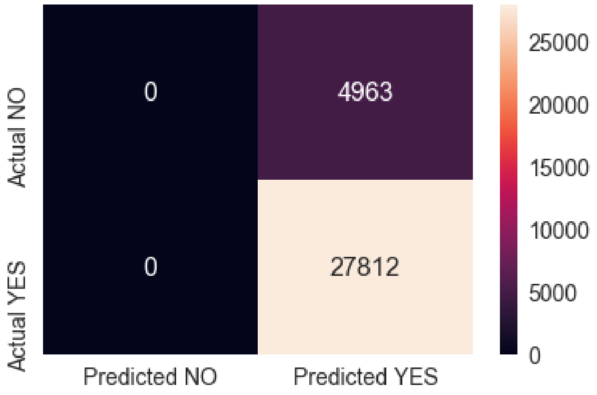
In [101]:

```
get_confusion_matrix(Trained_model_AVGW2V, X3_train_tr_tsvd, y_train)
```



In [102]:

```
get_confusion_matrix(Trained_model_AVGW2V, X3_test_tsvd, y_test)
```



2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

In [103]:

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

# imports
import matplotlib.pyplot as plt
%matplotlib inline


# define the parameter values that should be searched
# for python 2, k_range = range(1, 31)
k_range = list(range(1, 106, 8))
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 = dict(n_neighbors=k_range)
print(param_grid)

# instantiate model
knn = KNeighborsClassifier(n_neighbors=5)
```

```
[1, 9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89, 97, 105]
{'n_neighbors': [1, 9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89, 97, 105]}
```

In [104]:

```
#instantiate the grid
grid4 = GridSearchCV(knn, param_grid, cv=3, scoring='roc_auc', return_train_score=True)
```

In [105]:

```
# fit the grid with data
grid4.fit(X4_train_tr_tsvd, y_train)
```

Out[105]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='warn', n_jobs=None,
             param_grid={'n_neighbors': [1, 9, 17, 25, 33, 41, 49, 57, 65, 73,
                                         81, 89, 97, 105]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [106]:

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

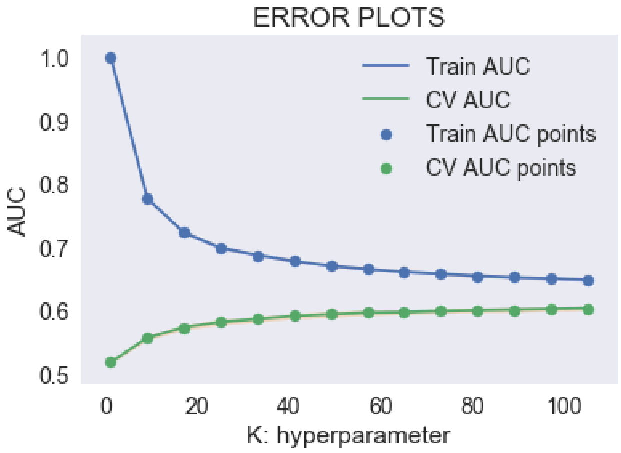
Out[106]:

```
{'mean_fit_time': array([0.14758182, 0.10727096, 0.10194174, 0.10427396, 0.12292854,
        0.09128173, 0.10094086, 0.09128126, 0.09527771, 0.09361251,
        0.0869573 , 0.09827669, 0.09228031, 0.09594488]),
 'std_fit_time': array([0.08939276, 0.00339751, 0.00326429, 0.00659552, 0.02705252,
        0.00612472, 0.01282529, 0.00758228, 0.00530913, 0.00308911,
        0.00827427, 0.00792551, 0.00124658, 0.00294166]),
 'mean_score_time': array([ 6.22743352, 13.90368716, 17.99367253, 19.67437434, 25.9433798 ,
        23.90527932, 26.09868757, 26.63304647, 27.62048101, 29.61900123,
        30.13136609, 31.71579663, 32.64726226, 33.08601062]),
 'std_score_time': array([0.26849415, 0.27357067, 1.53671551, 0.28200789, 1.05792624,
        0.45743953, 1.05860133, 0.16061613, 0.06917666, 0.92154079,
        0.11963569, 0.2476437 , 0.15172763, 0.25767793]),
 'param_n_neighbors': masked_array(data=[1, 9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89, 97, 105],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'n_neighbors': 1},
  {'n_neighbors': 9},
  {'n_neighbors': 17},
  {'n_neighbors': 25},
  {'n_neighbors': 33},
  {'n_neighbors': 41},
  {'n_neighbors': 49},
  {'n_neighbors': 57},
  {'n_neighbors': 65},
  {'n_neighbors': 73},
  {'n_neighbors': 81},
  {'n_neighbors': 89},
  {'n_neighbors': 97},
  {'n_neighbors': 105}],
 'split0_test_score': array([0.51643171, 0.55696564, 0.56934302, 0.57649789, 0.58185908,
        0.58648962, 0.58946835, 0.59178393, 0.59298009, 0.59569484,
        0.59696437, 0.59752632, 0.59861987, 0.60042527]),
 'split1_test_score': array([0.51921213, 0.55957385, 0.57778585, 0.58683711, 0.59116057,
        0.59618908, 0.59925439, 0.60088009, 0.59935873, 0.60244401,
        0.6041282 , 0.60498386, 0.60584475, 0.60764558]),
 'split2_test_score': array([0.51727232, 0.55322198, 0.57218458, 0.58072913, 0.58535721,
        0.59009818, 0.5923573 , 0.59584673, 0.59830534, 0.59817133,
        0.59891494, 0.60032847, 0.6016856 , 0.60158426]),
 'mean_test_score': array([0.51763871, 0.55658721, 0.57310444, 0.58135465, 0.58612558,
        0.59092558, 0.59369331, 0.5961702 , 0.59688132, 0.59877003,
        0.60000248, 0.60094618, 0.60205004, 0.60321836]),
 'std_test_score': array([0.0011643 , 0.00260688, 0.00350764, 0.00424412, 0.00383602,
        0.00400281, 0.00410534, 0.00372056, 0.00279198, 0.00278769,
        0.00302405, 0.00307573, 0.00296081, 0.00316608]),
 'rank_test_score': array([14, 13, 12, 11, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1]),
 'split0_train_score': array([1.        , 0.77828798, 0.7243943 , 0.70127116, 0.68825158,
        0.67711555, 0.67062119, 0.66590567, 0.66083   , 0.65713427,
        0.65398098, 0.65182191, 0.65096352, 0.64822221]),
 'split1_train_score': array([1.        , 0.77395898, 0.72234574, 0.69591109, 0.68600848,
        0.67556196, 0.6692461 , 0.66294797, 0.65945183, 0.65590733,
        0.65238036, 0.65048384, 0.64716997, 0.64540125]),
 'split2_train_score': array([0.99993523, 0.77671441, 0.72023927, 0.69778602, 0.68643041,
        0.67918793, 0.66941185, 0.66515981, 0.66094172, 0.65824094,
        0.65575743, 0.65245005, 0.65174416, 0.64985461]),
 'mean_train_score': array([0.99997841, 0.77632046, 0.72232643, 0.69832276, 0.68689682,
        0.67728848, 0.66975971, 0.66467115, 0.66040785, 0.65709418,
        0.65403959, 0.65158527, 0.64995922, 0.64782602]),
 'std_train_score': array([3.05313809e-05, 1.78912663e-03, 1.69634016e-03, 2.22090914e-03,
        9.73323010e-04, 1.48533716e-03, 6.12905638e-04, 1.25594382e-03,
        6.77546420e-04, 9.53114813e-04, 1.37930791e-03, 8.19956290e-04,
        1.99787641e-03, 1.83953634e-03])}
```

In [107]:

```
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_auc4 = grid4.cv_results_['mean_test_score']
cv_auc_std4 = grid4.cv_results_['std_test_score']


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

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

plt.scatter(k_range, train_auc4, label='Train AUC points')
plt.scatter(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()
```



In [108]:

```
# 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_)
```

```
0.6032183566240675
{'n_neighbors': 105}
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=105, p=2,
                     weights='uniform')
```

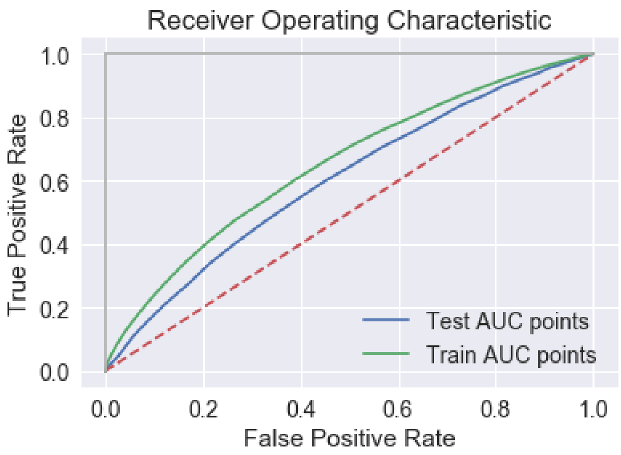
In [109]:

```
trained_model_TFIDFW2V = KNeighborsClassifier(n_neighbors=105, metric='minkowski', n_jobs=1).fit(X4
_train_tr_tsvd, y_train)
```

In [110]:

```
# Get predicted probabilities
y_score_test4 = trained_model_TFIDFW2V.predict_proba(X4_test_tsvd)[:,1]
y_score_train4 = trained_model_TFIDFW2V.predict_proba(X4_train_tr_tsvd)[:,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 [111]:

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

roc_auc_score(y_test, y_score_test4)
```

Out[111]:

```
0.6021356760000995
```

In [112]:

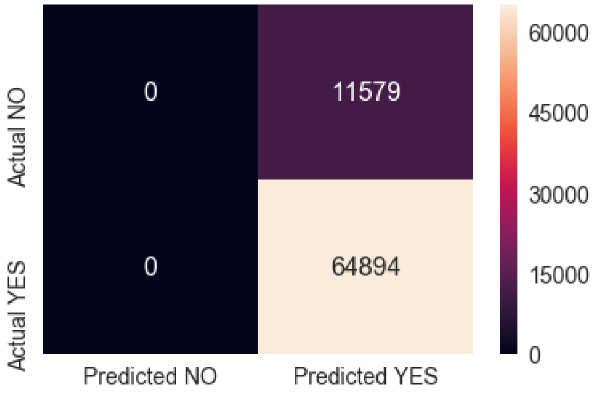
```
#https://chrisalbon.com/machine_learning/model_evaluation/generate_text_reports_on_performance/
from sklearn.metrics import classification_report

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

# Train model and make predictions
y_hat_4 = trained_model_TFIDFW2V.predict(X4_test_tsvd)

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

```
C:\Users\utsav94\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1437:
UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.

C:\Users\utsav94\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1437:
UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.

C:\Users\utsav94\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1437:
UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      4963
           1       0.85      1.00      0.92     27812

    accuracy                           0.85     32775
   macro avg       0.42      0.50      0.46     32775
weighted avg       0.72      0.85      0.78     32775
```

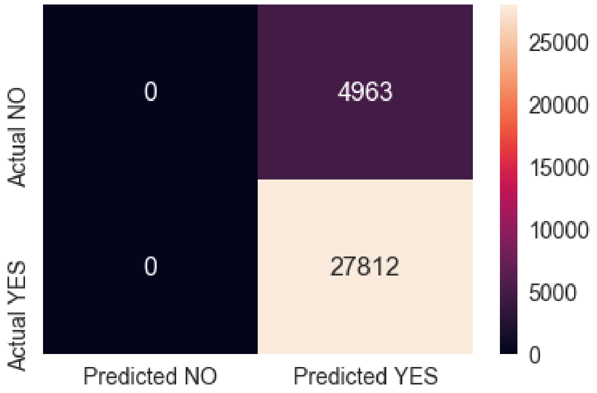
In [113]:

```
get_confusion_matrix(trained_model_TFIDFW2V, X4_train_tr_tsvd, y_train)
```



In [114]:

```
get_confusion_matrix(trained_model_TFIDFW2V, X4_test_tsvd, y_test)
```



2.5 Feature selection with `SelectKBest`

In [115]:

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

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

In [116]:

```
X1_train_tr_1_tr = X1_train_tr_1
X2_train_tr_2_tr = X2_train_tr_2
X3_train_tr_3_tr = X3_train_tr_3
X4_train_tr_4_tr = X4_train_tr_4

X1_test_1_tr = X1_test_1
X2_test_2_tr = X2_test_2
X3_test_3_tr = X3_test_3
X4_test_4_tr = X4_test_4

y_train_1_tr = y_train_1
y_test_1_tr = y_test_1

y_train_2_tr = y_train_1
y_test_2_tr = y_test_1

y_train_3_tr = y_train_1
y_test_3_tr = y_test_1

y_train_4_tr = y_train_1
y_test_4_tr = y_test_1
```

In [117]:

```
X1_train_tr_1
```

Out[117]:

```
<76473x2820 sparse matrix of type '<class 'numpy.float64'>'
 with 904064 stored elements in COOrdinate format>
```

In [118]:

```
from sklearn.datasets import load_digits
from sklearn.feature_selection import SelectKBest, mutual_info_classif
#X1_train_tr, y_train = load_digits(return_X_y=True)

X1_new = SelectKBest(mutual_info_classif, k=20).fit(X1_train_tr_1, y_train_1_tr)
#print(X1_new.shape)

#X2_train_tr, y_train = load_digits(return_X_y=True)
#X2.shape
X2_new = SelectKBest(mutual_info_classif, k=20).fit(X2_train_tr_2, y_train_2_tr)
#print(X2_new.shape)

#X3_train_tr, y_train = load_digits(return_X_y=True)
#ape
X3_new = SelectKBest(mutual_info_classif, k=20).fit(X3_train_tr_3, y_train_3_tr)
#print(X3_new.shape)

#X4_train_tr, y_train = load_digits(return_X_y=True)
#4.shape
X4_new = SelectKBest(mutual_info_classif, k=20).fit(X4_train_tr_4, y_train_4_tr)
#print(X4_new.shape)

X1_new_final = X1_new.transform(X1_train_tr_1)
print(X1_new_final.shape)

#X2_train_tr, y_train = load_digits(return_X_y=True)
#X2.shape
X2_new_final = X2_new.transform(X2_train_tr_2)
print(X2_new_final.shape)

#X3_train_tr, y_train = load_digits(return_X_y=True)
#ape
X3_new_final = X3_new.transform(X3_train_tr_3)
print(X3_new_final.shape)

#X4_train_tr, y_train = load_digits(return_X_y=True)
#4.shape
X4_new_final = X4_new.transform(X4_train_tr_4)
print(X4_new_final.shape)

#X1_test, y_test = load_digits(return_X_y=True)
#1.shape
```

```
X1_new_test = X1_new.transform(X1_test_1)
print(X1_new_test.shape)

#X2_test, y_test = load_digits(return_X_y=True)
#2.shape
X2_new_test = X2_new.transform(X2_test_2)
print(X2_new_test.shape)

#X3_test, y_test = load_digits(return_X_y=True)
#X3.shape
X3_new_test = X3_new.transform(X3_test_3)
print(X3_new_test.shape)

#X4_test, y_test = load_digits(return_X_y=True)
#X4.shape
X4_new_test = X4_new.transform(X4_test_4)
print(X4_new_test.shape)

from sklearn.preprocessing import label_binarize
y_train_new = label_binarize(y_train, classes=[0, 1])
y_test_new = label_binarize(y_test, classes=[0, 1])
```

```
(76473, 20)
(76473, 20)
(76473, 20)
(76473, 20)
(32775, 20)
(32775, 20)
(32775, 20)
(32775, 20)
```

In [119]:

```
#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)
k_range = list(range(1, 106, 8))
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 = dict(n_neighbors=k_range)
print(param_grid)

# instantiate model
knn = KNeighborsClassifier(n_neighbors=5)
```

```
[1, 9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89, 97, 105]
{'n_neighbors': [1, 9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89, 97, 105]}
```

In [120]:

```
#instantiate the grid
grid1k = GridSearchCV(knn, param_grid, cv=3, scoring='roc_auc', return_train_score=True)
```

In [121]:

```
# fit the grid with data
grid1k.fit(X1_new_final, y_train)
```

Out[121]:

Out[121]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='warn', n_jobs=None,
             param_grid={'n_neighbors': [1, 9, 17, 25, 33, 41, 49, 57, 65, 73,
                                         81, 89, 97, 105]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [122]:

```
# view the complete results (list of named tuples)
grid1k.cv_results_
```

Out[122]:

```
{'mean_fit_time': array([0.13692188, 0.02165484, 0.01665807, 0.02065531, 0.01432641,
        0.0199887 , 0.02465328, 0.01865443, 0.01767095, 0.01799051,
        0.01732461, 0.01631761, 0.01599105, 0.01332664]),
 'std_fit_time': array([0.17173186, 0.0059027 , 0.00308965, 0.005246  , 0.00205286,
        0.00407956, 0.013115  , 0.00730908, 0.0059076 , 0.00244814,
        0.00530946, 0.00618475, 0.00647692, 0.00124777]),
 'mean_score_time': array([65.52739088, 80.55776334, 80.39552355, 79.34445985, 79.33080093,
        80.21629254, 80.23228359, 75.73619564, 70.06577063, 77.92727367,
        70.72868427, 68.01196345, 68.45704222, 70.98063199]),
 'std_score_time': array([3.06785226, 0.6799438 , 0.59575872, 0.97302714, 0.90554619,
        0.9291042 , 0.1409737 , 3.46743712, 2.42939581, 3.45677023,
        3.4861569 , 0.22151011, 0.71510159, 4.46058537]),
 'param_n_neighbors': masked_array(data=[1, 9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89, 97, 105],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'n_neighbors': 1},
  {'n_neighbors': 9},
  {'n_neighbors': 17},
  {'n_neighbors': 25},
  {'n_neighbors': 33},
  {'n_neighbors': 41},
  {'n_neighbors': 49},
  {'n_neighbors': 57},
  {'n_neighbors': 65},
  {'n_neighbors': 73},
  {'n_neighbors': 81},
  {'n_neighbors': 89},
  {'n_neighbors': 97},
  {'n_neighbors': 105}],
 'split0_test_score': array([0.51940569, 0.5721985 , 0.58749095, 0.59350012, 0.5986891 ,
        0.60351991, 0.60719734, 0.61085895, 0.61473965, 0.61720593,
        0.61728992, 0.61911743, 0.61883008, 0.61988024]),
 'split1_test_score': array([0.52051963, 0.57347079, 0.59070402, 0.59638776, 0.60416931,
        0.60673481, 0.61119703, 0.61607392, 0.61814896, 0.61830739,
        0.62100182, 0.62331337, 0.62405989, 0.62640725]),
 'split2_test_score': array([0.51743536, 0.57479852, 0.58718286, 0.59666143, 0.60360492,
        0.60637027, 0.609795  , 0.61159458, 0.61329778, 0.61495263,
        0.61787349, 0.62048305, 0.62156721, 0.62070654]),
 'mean_test_score': array([0.51912025, 0.57348923, 0.58845928, 0.5955164 , 0.60215438,
        0.60554162, 0.60939642, 0.61284248, 0.61539548, 0.61682201,
        0.61872173, 0.62097126, 0.62148569, 0.62233133]),
 'std_test_score': array([0.00127521, 0.00106153, 0.00159225, 0.00143014, 0.0024612 ,
        0.00143734, 0.00165702, 0.00230463, 0.00203404, 0.00139621,
        0.00162977, 0.00174744, 0.00213586, 0.00290178]),
 'rank_test_score': array([14, 13, 12, 11, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1]),
 'split0_train_score': array([0.97842606, 0.78507281, 0.73287116, 0.70997598, 0.6965761 ,
        0.68695082, 0.68048705, 0.67577078, 0.67077416, 0.66708577,
        0.66591474, 0.66419278, 0.66213599, 0.66101426]),
 'split1_train_score': array([0.97662857, 0.78259909, 0.73223229, 0.70804308, 0.69269417,
        0.68415244, 0.67730381, 0.67330203, 0.66863935, 0.66690133,
        0.66319747, 0.66216038, 0.66038511, 0.65771162]),
 'split2_train_score': array([0.97559507, 0.78289885, 0.73170409, 0.70699011, 0.69256911,
        0.68335734, 0.67752483, 0.67362541, 0.66945011, 0.66720564,
        0.66483134, 0.66276603, 0.66044816, 0.65778332]),
 'mean_train_score': array([0.97688323, 0.78352358, 0.73226918, 0.70833639, 0.69394646,
        0.6848202 , 0.67843856, 0.67423274, 0.6696212 , 0.66706425,
        0.66464785, 0.66303973, 0.66098975, 0.6588364 ]),
 'std_train_score': array([0.00116969, 0.00110228, 0.00047717, 0.00123649, 0.00186014,
        0.00154115, 0.00145131, 0.00109554, 0.00087989, 0.00012517,
        0.00111688, 0.000852  , 0.00081092, 0.00154026])}
```

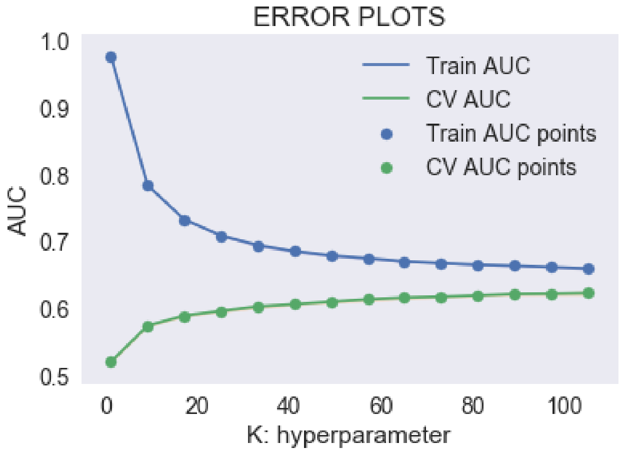
In [123]:

```
# examine the best model

# Single best score achieved across all params (k)
print(grid1k.best_score_)

# Dictionary containing the parameters (k) used to generate that score
print(grid1k.best_params_)

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

print(grid1k.best_estimator_)
```

```
0.6223313345825788
{'n_neighbors': 105}
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=105, p=2,
                     weights='uniform')
```

In [124]:

```
train_auc1k = grid1k.cv_results_['mean_train_score']
train_auc_std1k = grid1k.cv_results_['std_train_score']
cv_auc1k = grid1k.cv_results_['mean_test_score']
cv_auc_std1k = grid1k.cv_results_['std_test_score']

plt.plot(k_range, train_auc1k, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(k_range, train_auc1k - train_auc_std1k, train_auc1k + train_auc_std1k, alpha
=0.2,color='darkblue')

plt.plot(k_range, cv_auc1k, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(k_range, cv_auc1k - cv_auc_std1k, cv_auc1k + cv_auc_std1k, alpha=0.2,color='
darkorange')

plt.scatter(k_range, train_auc1k, label='Train AUC points')
plt.scatter(k_range, cv_auc1k, label='CV AUC points')


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

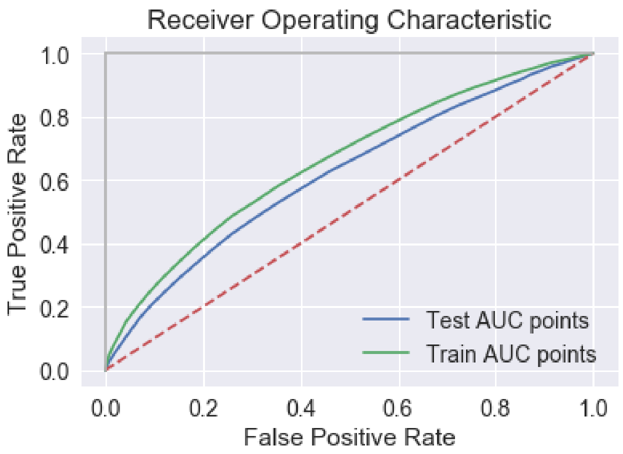


In [125]:

```
Trained_model_BOW_k = KNeighborsClassifier(algorithm = 'brute', n_neighbors=105, metric='minkowski'
, n_jobs=-1).fit(X1_new_final, y_train)
```

In [126]:

```
import sklearn.metrics as metrics

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels
# Get predicted probabilities
y_score_test = Trained_model_BOW_k.predict_proba(X1_new_test)[:,1]
y_score_train = Trained_model_BOW_k.predict_proba(X1_new_final)[:,1]

#y_score_test_tr = confusion_matrix(y_test.argmax(axis=1), y_score_test.argmax(axis=1))
#y_score_train_tr = confusion_matrix(y_train.argmax(axis=1), y_score_train.argmax(axis=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 [127]:

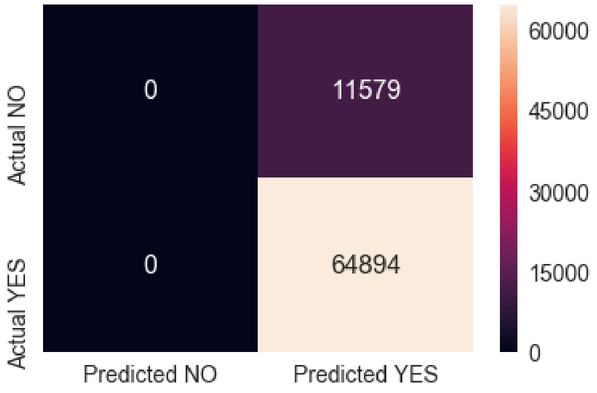
```
import numpy as np
from sklearn.metrics import roc_auc_score

roc_auc_score(y_test, y_score_test)
```

Out[127]:

```
0.6187239766708563
```

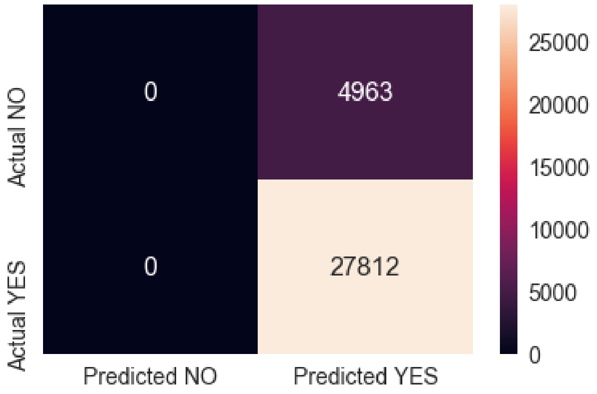
In [128]:

```
get_confusion_matrix(Trained_model_BOW_k, X1_new_final, y_train)
```



In [129]:

```
get_confusion_matrix(Trained_model_BOW_k, X1_new_test, y_test)
```



3. Conclusions

In [130]:

```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable

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

DBZ.add_row(["Brute", "kNN", "105", "0.6006272716100003"])
DBZ.add_row(["Brute", "kNN", "97", "0.5972478666307289"])
DBZ.add_row(["Brute", "kNN", "97", "0.4963273492070866"])
DBZ.add_row(["Brute", "kNN", "105", "0.5946352389242309"])
```

In [131]:

```
print(DBZ)
```

```
+------------+-------+----------------+--------------------+
| Vectorizer | Model | Hyperparameter |        AUC         |
+------------+-------+----------------+--------------------+
|   Brute    |  kNN  |      105       | 0.6006272716100003 |
|   Brute    |  kNN  |       97       | 0.5972478666307289 |
|   Brute    |  kNN  |       97       | 0.4963273492070866 |
|   Brute    |  kNN  |      105       | 0.5946352389242309 |
+------------+-------+----------------+--------------------+
```

In [ ]: