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	Descri
project_id	A unique identifier for the proposed project. **Example:** `p03`
project_title	Title of the project. **Exam** • `Art Will Make You Ha` • `First Grade`
project_grade_category	Grade level of students for which the project is targeted. One of the foll enumerated va • `Grades Pr` • `Grades` • `Grades` • `Grades`
project_subject_categories	One or more (comma-separated) subject categories for the project fro following enumerated list of va • `Applied Lear` • `Care & Hu` • `Health & Sp` • `History & Ci` • `Literacy & Lang` • `Math & Sci` • `Music & The` • `Special N` • `Wa` **Exam** • `Music & The` • `Literacy & Language, Math & Sci`
school_state	State where school is located (Two-letter U.S. postal (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Postal_co **Example**
project_subject_subcategories	One or more (comma-separated) subject subcategories for the pr **Exam** • `Lite` • `Literature & Writing, Social Scie`
project_resource_summary	An explanation of the resources needed for the project. **Exan** • `My students need hands on literacy materials to mar sensory ne`
project_essay_1	First application e
project_essay_2	Second application e
project_essay_3	Third application e
project_essay_4	Fourth application e

Feature	Descri
project_submitted_datetime	Datetime when project application was submitted. **Example:** 2016-04 12:43:56
teacher_id	A unique identifier for the teacher of the proposed project. **Exa** bdf8baa8fedef6bfeec7ae4ff1c1
teacher_prefix	Teacher's title. One of the following enumerated va • • • • • • Teac
teacher_number_of_previously_posted_projects	Number of project applications previously submitted by the same tea **Exampl**

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

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

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

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

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

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

Notes on the Essay Data

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

- __project_essay_1:__ "Introduce us to your classroom"
- __project_essay_2:__ "Tell us more about your students"
- __project_essay_3:__ "Describe how your students will use the materials you're requesting"
- __project_essay_3:__ "Close by sharing why your project will make a difference"

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

- __project_essay_1:__ "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."
- __project_essay_2:__ "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

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

In [1]:
```
%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 chart_studio import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

1.1 Reading Data

In [2]:
```
project_data = pd.read_csv('train_data.csv') # Taking 80000 datapoints
resource_data = pd.read_csv('resources.csv')
```

In [3]:
```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
```

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

In [4]:
```
print("Number of data points in train data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

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

Out[4]:

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

1.2 preprocessing of `project_subject_categories`

In [5]:
```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflo
w.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-fr
om-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-strin
g-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 & Scienc
e", "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 ''(emp
ty) ex:"Math & Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the tra
iling spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

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

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

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

1.3 preprocessing of project_subject_subcategories

In [6]:
```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflo
w.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-fr
om-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-strin
g-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 & Scienc
e", "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 ''(emp
ty) ex:"Math & Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the tra
iling 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/2289859
5/4084039
my_counter = Counter()
for word in project_data['clean_subcategories'].values:
    my_counter.update(word.split())

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

1.3 Text preprocessing

In [7]:
```
# merge two column text dataframe:
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(str) + \
                        project_data["project_essay_4"].map(str)
```

In [8]:
```
project_data.head(2)
```

Out[8]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	

In [9]:
```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

In [10]:
```
# printing some random reviews
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
print(project_data['essay'].values[1000])
print("="*50)
print(project_data['essay'].values[20000])
print("="*50)
print(project_data['essay'].values[99999])
print("="*50)
```

My students are English learners that are working on English as their second or third languages. We are a melting pot of refugees, immigrants, and native-born Americans bringing the gift of language to our school. \r\n\r\n We have over 24 languages represented in our English Learner program with students at every level of mastery. We also have over 40 countries represented with the families within our school. Each student brings a wealth of knowledge and ex periences to us that open our eyes to new cultures, beliefs, and respect.\"Th e limits of your language are the limits of your world.\"-Ludwig Wittgenstein Our English learner's have a strong support system at home that begs for more resources. Many times our parents are learning to read and speak English alo ng side of their children. Sometimes this creates barriers for parents to be able to help their child learn phonetics, letter recognition, and other readi ng skills.\r\n\r\nBy providing these dvd's and players, students are able to continue their mastery of the English language even if no one at home is able to assist. All families with students within the Level 1 proficiency status, will be a offered to be a part of this program. These educational videos wil l be specially chosen by the English Learner Teacher and will be sent home re gularly to watch. The videos are to help the child develop early reading ski lls.\r\n\r\nParents that do not have access to a dvd player will have the opp ortunity to check out a dvd player to use for the year. The plan is to use t hese videos and educational dvd's for the years to come for other EL student s.\r\nnannan

==========

The 51 fifth grade students that will cycle through my classroom this year al l love learning, at least most of the time. At our school, 97.3% of the stude nts receive free or reduced price lunch. Of the 560 students, 97.3% are minor ity students. \r\nThe school has a vibrant community that loves to get togeth er and celebrate. Around Halloween there is a whole school parade to show off the beautiful costumes that students wear. On Cinco de Mayo we put on a big f estival with crafts made by the students, dances, and games. At the end of th e year the school hosts a carnival to celebrate the hard work put in during t he school year, with a dunk tank being the most popular activity.My students will use these five brightly colored Hokki stools in place of regular, statio nary, 4-legged chairs. As I will only have a total of ten in the classroom an d not enough for each student to have an individual one, they will be used in a variety of ways. During independent reading time they will be used as speci al chairs students will each use on occasion. I will utilize them in place of chairs at my small group tables during math and reading times. The rest of th e day they will be used by the students who need the highest amount of moveme nt in their life in order to stay focused on school.\r\n\r\nWhenever asked wh at the classroom is missing, my students always say more Hokki Stools. They c an't get their fill of the 5 stools we already have. When the students are si tting in group with me on the Hokki Stools, they are always moving, but at th e same time doing their work. Anytime the students get to pick where they can sit, the Hokki Stools are the first to be taken. There are always students wh o head over to the kidney table to get one of the stools who are disappointed as there are not enough of them. \r\n\r\nWe ask a lot of students to sit for 7 hours a day. The Hokki stools will be a compromise that allow my students t o do desk work and move at the same time. These stools will help students to meet their 60 minutes a day of movement by allowing them to activate their co re muscles for balance while they sit. For many of my students, these chairs will take away the barrier that exists in schools for a child who can't sit s till.nannan

==========

How do you remember your days of school? Was it in a sterile environment with plain walls, rows of desks, and a teacher in front of the room? A typical day in our room is nothing like that. I work hard to create a warm inviting theme

d room for my students look forward to coming to each day.\r\n\r\nMy class is made up of 28 wonderfully unique boys and girls of mixed races in Arkansas.\r\nThey attend a Title I school, which means there is a high enough percentage of free and reduced-price lunch to qualify. Our school is an \"open classroom\" concept, which is very unique as there are no walls separating the classrooms. These 9 and 10 year-old students are very eager learners; they are like sponges, absorbing all the information and experiences and keep on wanting more.With these resources such as the comfy red throw pillows and the whimsical nautical hanging decor and the blue fish nets, I will be able to help create the mood in our classroom setting to be one of a themed nautical environment. Creating a classroom environment is very important in the success in each and every child's education. The nautical photo props will be used with each child as they step foot into our classroom for the first time on Meet the Teacher evening. I'll take pictures of each child with them, have them developed, and then hung in our classroom ready for their first day of 4th grade. This kind gesture will set the tone before even the first day of school! The nautical thank you cards will be used throughout the year by the students as they create thank you cards to their team groups.\r\n\r\nYour generous donations will help me to help make our classroom a fun, inviting, learning environment from day one.\r\n\r\nIt costs lost of money out of my own pocket on resources to get our classroom ready. Please consider helping with this project to make our new school year a very successful one. Thank you!nannan

==========

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

==========

The mediocre teacher tells. The good teacher explains. The superior teacher demonstrates. The great teacher inspires. -William A. Ward\r\n\r\nMy school has 803 students which is makeup is 97.6% African-American, making up the largest segment of the student body. A typical school in Dallas is made up of 23.2% African-American students. Most of the students are on free or reduced lunch. We aren't receiving doctors, lawyers, or engineers children from rich backgrounds or neighborhoods. As an educator I am inspiring minds of young children and we focus not only on academics but one smart, effective, efficient, and disciplined students with good character.In our classroom we can utilize the Bluetooth for swift transitions during class. I use a speaker which doesn't amplify the sound enough to receive the message. Due to the volume of my speaker my students can't hear videos or books clearly and it isn't making the lessons as meaningful. But with the bluetooth speaker my students will be able to hear and I can stop, pause and replay it at any time.\r\nThe cart will allow me to have more room for storage of things that are needed for the day and has an extra part to it I can use. The table top chart has all of the letter, words and pictures for students to learn about different letters and it

```
is more accessible.nannan
==================================================
```

In [11]:
```
# https://stackoverflow.com/a/47091490/4084039
import re

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

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

In [12]:
```
sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

```
My kindergarten students have varied disabilities ranging from speech and lan
guage delays, cognitive delays, gross/fine motor delays, to autism. They are
eager beavers and always strive to work their hardest working past their limi
tations. \r\n\r\nThe materials we have are the ones I seek out for my student
s. I teach in a Title I school where most of the students receive free or red
uced price lunch.  Despite their disabilities and limitations, my students lo
ve coming to school and come eager to learn and explore.Have you ever felt li
ke you had ants in your pants and you needed to groove and move as you were i
n a meeting? This is how my kids feel all the time. The want to be able to mo
ve as they learn or so they say.Wobble chairs are the answer and I love then
because they develop their core, which enhances gross motor and in Turn fine
motor skills. \r\nThey also want to learn through games, my kids do not want
to sit and do worksheets. They want to learn to count by jumping and playing.
Physical engagement is the key to our success. The number toss and color and
shape mats can make that happen. My students will forget they are doing work
and just have the fun a 6 year old deserves.nannan
==================================================
```

In [13]:
```
# \r \n \t remove from string python: http://texthandler.com/info/remove-line-breaks-python/
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

```
My kindergarten students have varied disabilities ranging from speech and lan
guage delays, cognitive delays, gross/fine motor delays, to autism. They are
eager beavers and always strive to work their hardest working past their limi
tations.     The materials we have are the ones I seek out for my students. I
teach in a Title I school where most of the students receive free or reduced
price lunch.  Despite their disabilities and limitations, my students love co
ming to school and come eager to learn and explore.Have you ever felt like yo
u had ants in your pants and you needed to groove and move as you were in a m
eeting? This is how my kids feel all the time. The want to be able to move as
they learn or so they say.Wobble chairs are the answer and I love then becaus
e they develop their core, which enhances gross motor and in Turn fine motor
skills.   They also want to learn through games, my kids do not want to sit a
nd do worksheets. They want to learn to count by jumping and playing. Physica
l engagement is the key to our success. The number toss and color and shape m
ats can make that happen. My students will forget they are doing work and jus
t have the fun a 6 year old deserves.nannan
```

In [14]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

```
My kindergarten students have varied disabilities ranging from speech and lan
guage delays cognitive delays gross fine motor delays to autism They are eage
r beavers and always strive to work their hardest working past their limitati
ons The materials we have are the ones I seek out for my students I teach in
a Title I school where most of the students receive free or reduced price lun
ch Despite their disabilities and limitations my students love coming to scho
ol and come eager to learn and explore Have you ever felt like you had ants i
n your pants and you needed to groove and move as you were in a meeting This
is how my kids feel all the time The want to be able to move as they learn or
so they say Wobble chairs are the answer and I love then because they develop
their core which enhances gross motor and in Turn fine motor skills They also
want to learn through games my kids do not want to sit and do worksheets They
want to learn to count by jumping and playing Physical engagement is the key
to our success The number toss and color and shape mats can make that happen
My students will forget they are doing work and just have the fun a 6 year ol
d deserves nannan
```

In [15]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you'
, "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he'
, 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'it
self', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 't
hat', "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', 'becau
se', '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', 'a
ll', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'tha
n', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "shoul
d've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn',
"didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'm
a', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shoul
dn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [16]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████████████
| 109248/109248 [00:48<00:00, 2231.64it/s]
```

In [17]:
```
# after preprocesing
preprocessed_essays[20000]
# replacing essay data with cleaned and preprocessed data
project_data['essay'] = preprocessed_essays
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
project_data.drop(['project_essay_4'], axis=1, inplace=True)
```

1.4 Preprocessing of `project_title`

In [18]:
```
# similarly you can preprocess the titles also
def preprocess_text_func(text_data):
    sent = decontracted(text_data)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    return sent.lower()
```

In [19]:
```
preprocessed_titles = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['project_title'].values):
    preprocessed_titles.append(preprocess_text_func(sentance))
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 
109248/109248 [00:02<00:00, 48213.12it/s]
```

In [20]:
```
print(preprocessed_titles[5:12])
```

```
['flexible seating mrs jarvis terrific third graders', 'chromebooks special e
ducation reading program', 'it 21st century', 'targeting more success class',
'just for love reading pure pleasure', 'reading changes lives', 'elevating ac
ademics parent rapports through technology']
```

In [21]:
```
print(project_data["project_title"].values[5:12])
```

```
["Flexible Seating for Mrs. Jarvis' Terrific Third Graders!!"
 'Chromebooks for Special Education Reading Program'
 "It's the 21st Century" 'Targeting More Success in Class'
 'Just For the Love of Reading--\\r\\nPure Pleasure'
 'Reading Changes Lives'
 'Elevating Academics and Parent Rapports Through Technology']
```

In [22]:
```
# replacing project_title with cleaned data
project_data['UnCleaned_title']=project_data['project_title']
project_data['project_title']=preprocessed_titles
```

In [93]:
```
print(project_data["project_title"].values[5:12])
```

```
['flexible seating mrs jarvis terrific third graders'
 'chromebooks special education reading program' 'it 21st century'
 'targeting more success class' 'just for love reading pure pleasure'
 'reading changes lives'
 'elevating academics parent rapports through technology']
```

1.5 Preparing data for models

In [23]:
```
project_data.columns
```

Out[23]:
```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', 'project_titl
e',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'UnCleaned_titl
e'],
      dtype='object')
```

In [24]:
```
# dropping unwanted columns such as Unnamed
project_data.drop(['Unnamed: 0'], axis=1, inplace=True)
```

we are going to consider

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

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

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

1.5.1 Vectorizing Categorical data

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

In [25]:
```
# 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()), lowercas
e=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)
category_feature_names = list(vectorizer.get_feature_names())
```

```
['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 [26]:
```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowe
rcase=False, binary=True)
sub_categories_one_hot = vectorizer.fit_transform(project_data['clean_subcateg
ories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",sub_categories_one_hot.shape)
subcategory_feature_names = vectorizer.get_feature_names()
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement',
'Extracurricular', 'Civics_Government', 'ForeignLanguages', 'NutritionEducati
on', 'Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingArts', 'CharacterE
ducation', 'TeamSports', 'Other', 'College_CareerPrep', 'Music', 'History_Geo
graphy', 'Health_LifeScience', 'EarlyDevelopment', 'ESL', 'Gym_Fitness', 'Env
ironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences', 'Spec
ialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig  (109248, 30)
```

In [27]:
```
# you can do the similar thing with state, teacher_prefix and project_grade_ca
tegory also
def perform_one_hot_encoding(listdata, category,fillnan_value=""):
    vectorizer =  CountVectorizer(vocabulary=listdata, lowercase=False, binary
=True)
    vectorizer.fit(project_data[category].fillna(fillnan_value).values)
    print(vectorizer.get_feature_names())
    print("="*50)
    feature_names = vectorizer.get_feature_names()
    return vectorizer.transform(project_data[category].fillna(fillnan_value).v
alues), feature_names
```

In [28]:
```
# One hot encoding for school state
countries_list = sorted(project_data["school_state"].value_counts().keys())
school_state_one_hot,school_state_feature_names = perform_one_hot_encoding(cou
ntries_list, "school_state")
print("Shape of matrix after one hot encodig ",school_state_one_hot.shape)
```

```
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'I
A', 'ID', 'IL', 'IN', 'KS', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO',
'MS', 'MT', 'NC', '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 [29]:
```
# Project_Grade_Category - replacing hyphens, spaces with Underscores
project_data['project_grade_category'] = project_data['project_grade_category'
].map({'Grades PreK-2': 'Grades_PreK_2',

'Grades 6-8' : 'Grades_6_8',

'Grades 3-5' : 'Grades_3_5',

'Grades 9-12' : 'Grades_9_12'})
project_data['teacher_prefix'] = project_data['teacher_prefix'].map({'Mrs.':
'Mrs', 'Ms.': 'Ms', 'Mr.' : 'Mr',
                                                                     'Teacher'
: 'Teacher', 'Dr.' : 'Dr'})
```

In [30]:
```
# Replacing Null values with most repititive values
project_data["teacher_prefix"].fillna("Mrs", inplace=True)
# One hot encoding for teacher_prefix
teacher_prefix_list = sorted(project_data["teacher_prefix"].value_counts().key
s())
print (teacher_prefix_list)
teacher_prefix_one_hot, teacher_prefix_feature_names = perform_one_hot_encodin
g(teacher_prefix_list, "teacher_prefix", "Mrs.")
print("Shape of matrix after one hot encodig ",teacher_prefix_one_hot.shape)
```

```
['Dr', 'Mr', 'Mrs', 'Ms', 'Teacher']
['Dr', 'Mr', 'Mrs', 'Ms', 'Teacher']
==================================================
Shape of matrix after one hot encodig  (109248, 5)
```

In [31]:
```
# One hot encoding for project_grade_category
grade_list = sorted(project_data["project_grade_category"].value_counts().keys
())
grade_one_hot, grade_feature_names = perform_one_hot_encoding(grade_list, "pro
ject_grade_category")
print("Shape of matrix after one hot encodig ",grade_one_hot.shape)
```

```
['Grades_3_5', 'Grades_6_8', 'Grades_9_12', 'Grades_PreK_2']
==================================================
Shape of matrix after one hot encodig  (109248, 4)
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [32]:
```
# We are considering only the words which appeared in at least 10 documents(ro
ws 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)
bow_essay_feature_names = vectorizer.get_feature_names()
```

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

In [33]:
```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
vectorizer_titles = CountVectorizer(min_df=10)
text_bow_titles = vectorizer_titles.fit_transform(preprocessed_titles)
print("Shape of matrix after one hot encodig ",text_bow_titles.shape)
bow_titles_feature_names = vectorizer.get_feature_names()
```

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

1.5.2.2 TFIDF vectorizer

In [34]:
```
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)
tfidf_essay_feature_names = vectorizer.get_feature_names()
```

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

In [35]:
```
# TFIDF  Vectorizer for Preprocessed Title
vectorizer_titles = TfidfVectorizer(min_df=10)
text_tfidf_titles = vectorizer_titles.fit_transform(preprocessed_titles)
print("Shape of matrix after one hot encodig ",text_tfidf_titles.shape)
tfidf_titles_feature_names =vectorizer_titles.get_feature_names()
```

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

1.5.2.3 Using Pretrained Models: Avg W2V

In [36]:
```
'''
# Reading glove vectors in python: https://stackoverflow.com/a/38230349/408403
9
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 coup
us", \
      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-save-and-load-variables-in-python/

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

Out[36]:
```
"\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/40
84039\ndef loadGloveModel(gloveFile):\n    print ("Loading Glove Model")\n
f = open(gloveFile,\'r\', encoding="utf8")\n    model = {}\n    for line in t
qdm(f):\n        splitLine = line.split()\n        word = splitLine[0]\n
embedding = np.array([float(val) for val in splitLine[1:]])\n        model[wo
rd] = embedding\n    print ("Done.",len(model)," words loaded!")\n    return
model\nmodel = loadGloveModel(\'glove.42B.300d.txt\')\n\n# ==================
==========\nOutput:\n    \nLoading Glove 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 p
reproced_titles:\n    words.extend(i.split(\' \'))\nprint("all the words in t
he coupus", len(words))\nwords = set(words)\nprint("the unique words in the c
oupus", len(words))\n\ninter_words = set(model.keys()).intersection(words)\np
rint("The number of words that are present in both glove vectors and our coup
us",       len(inter_words),"(",np.round(len(inter_words)/len(words)*100,
3),"%)")\n\nwords_courpus = {}\nwords_glove = set(model.keys())\nfor i in wor
ds:\n    if i in words_glove:\n        words_courpus[i] = model[i]\nprint("wo
rd 2 vec length", len(words_courpus))\n\n\n\n# stronging variables into pickle
files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-v
ariables-in-python/\n\nimport pickle\nwith open(\'glove_vectors\', \'wb\') as
f:\n    pickle.dump(words_courpus, f)\n\n\n'
```

In [37]:
```
# stronging variables into pickle files python: http://www.jessicayung.com/how
-to-use-pickle-to-save-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 [38]:
```
# 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 [00:28<00:00, 3827.94it/s]

109248
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [39]:
```
# 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 [40]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in th
is 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/revie
w
    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 v
alue((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split
())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors.append(vector)

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

```
100%|██████████████████████████████████████████████████████████████████████
██| 109248/109248 [03:15<00:00, 557.81it/s]

109248
300
```

In [41]:
```
# Similarly you can vectorize for title also
```

1.5.3 Vectorizing Numerical features

In [42]:
```
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 [43]:
```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)

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

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

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

In [44]:
```
price_standardized
```

Out[44]:
```
array([[-0.3905327 ],
       [ 0.00239637],
       [ 0.59519138],
       ...,
       [-0.15825829],
       [-0.61243967],
       [-0.51216657]])
```

In [45]:
```
# Vectorizing teacher_number_of_previously_posted_projects
teacher_number_of_previously_posted_projects_scalar = StandardScaler()
teacher_number_of_previously_posted_projects_scalar.fit(project_data['teacher_number_of_previously_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.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_previously_posted_projects'].values.reshape(-1, 1))
```

```
Mean : 11.153165275336848, Standard deviation : 27.77702641477403
```

1.5.4 Merging all the above features

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

In [46]:
```
# Categorical
print(school_state_one_hot.shape)
print(categories_one_hot.shape)
print(sub_categories_one_hot.shape)
print(teacher_prefix_one_hot.shape)
print(grade_one_hot.shape)
print(text_bow_titles.shape)
print(text_bow.shape)
# Numerical
print(price_standardized.shape)
print(teacher_number_of_previously_posted_projects_standardized.shape)
```

```
(109248, 51)
(109248, 9)
(109248, 30)
(109248, 5)
(109248, 4)
(109248, 3329)
(109248, 16623)
(109248, 1)
(109248, 1)
```

In [47]:
```
# 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 den
se matirx :)
X = hstack((school_state_one_hot,categories_one_hot, sub_categories_one_hot, t
eacher_prefix_one_hot,
            grade_one_hot, text_bow_titles, text_bow, price_standardized,
            teacher_number_of_previously_posted_projects_standardized))
X.shape
```

Out[47]: (109248, 20053)

In [48]:
```
# please write all the code with proper documentation, and proper titles for e
ach subsection
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the rea
der
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

Assignment 4: Naive Bayes

1. **Apply Multinomial NaiveBayes on these feature sets**
    - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (BOW)
    - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
2. **The hyper paramter tuning(find best Alpha)**
    - Find the best hyper parameter which will give the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
    - Consider a wide range of alpha values for hyperparameter tuning, start as low as 0.00001
    - Find the best hyper paramter using k-fold cross validation or simple cross validation data
    - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning
3. **Feature importance**
    - Find the top 10 features of positive class and top 10 features of negative class for both feature sets Set 1 and Set 2 using values of `feature_log_prob_` parameter of MultinomialNB (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html) and print their corresponding feature names
4. **Representation of results**
    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure. Here on X-axis you will have alpha values, since they have a wide range, just to represent those alpha values on the graph, apply log function on those alpha values.
    - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
    - Along with plotting ROC curve, you need to print the confusion matrix (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.
    (https://seaborn.pydata.org/generated/seaborn.heatmap.html)
    (https://seaborn.pydata.org/generated/seaborn.heatmap.html)
(https://seaborn.pydata.org/generated/seaborn.heatmap.html)

(https://seaborn.pydata.org/generated/seaborn.heatmap.html)
5. **Conclusion** (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

(https://seaborn.pydata.org/generated/seaborn.heatmap.html)
    - 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 (https://seaborn.pydata.org/generated/seaborn.heatmap.html) link (http://zetcode.com/python/prettytable/)

2. Naive Bayes

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

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
# 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

# Seperating Labels from Project_Data dataframe
y = project_data['project_is_approved'].values
X = project_data.drop(['project_is_approved'], axis=1)
X.head(1)
```

Out[49]:

	id	teacher_id	teacher_prefix	school_state	project_submitted_dat
0	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs	IN	2016-12-05 13:

In [50]:
```
# Train Test Stratified Split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.33, strat
ify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.
33)
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)
```

```
(49041, 15) (49041,)
(24155, 15) (24155,)
(36052, 15) (36052,)
====================================================================================
======================
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [51]:
```
# Encoding School State - OHE
# School State
vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on tra
in data

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
school_state_feature_names = vectorizer.get_feature_names()
```

```
After vectorizations
(49041, 51) (49041,)
(24155, 51) (24155,)
(36052, 51) (36052,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'i
a', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo',
'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'or',
'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'wy']
====================================================================================
======================
```

In [96]:
```
len(school_state_feature_names)
```

Out[96]:
```
51
```

In [52]:
```
# Encoding Teacher Prefix OHE
# teacher_prefix
vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on t
rain data

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_cv_teacher_ohe.shape, y_cv.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
teacher_prefix_feature_names = vectorizer.get_feature_names()
```

```
After vectorizations
(49041, 5) (49041,)
(24155, 5) (24155,)
(36052, 5) (36052,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
====================================================================================
======================
```

In [97]:
```
len(teacher_prefix_feature_names)
```

Out[97]:
```
5
```

In [53]:
```
# Encoding project_grade_category
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen o
nly on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].val
ues)
X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].value
s)

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
grade_feature_names = vectorizer.get_feature_names()
```

```
After vectorizations
(49041, 4) (49041,)
(24155, 4) (24155,)
(36052, 4) (36052,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
====================================================================================
======================
```

In [98]:
```
len(grade_feature_names)
```

Out[98]:
```
4
```

In [54]:
```
# Encoding Categories
# clean_categories
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on
train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_category_ohe = vectorizer.transform(X_train['clean_categories'].values
)
X_cv_category_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_category_ohe = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_category_ohe.shape, y_train.shape)
print(X_cv_category_ohe.shape, y_cv.shape)
print(X_test_category_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
category_feature_names = vectorizer.get_feature_names()
```

```
After vectorizations
(49041, 9) (49041,)
(24155, 9) (24155,)
(36052, 9) (36052,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'litera
cy_language', 'math_science', 'music_arts', 'specialneeds', 'warmth']
====================================================================================
======================
```

In [99]:
```
len(category_feature_names)
```

Out[99]:
```
9
```

In [55]:
```
# Encoding sub categories
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen only
on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_subcategory_ohe = vectorizer.transform(X_train['clean_subcategories'].
values)
X_cv_subcategory_ohe = vectorizer.transform(X_cv['clean_subcategories'].values
)
X_test_subcategory_ohe = vectorizer.transform(X_test['clean_subcategories'].va
lues)

print("After vectorizations")
print(X_train_subcategory_ohe.shape, y_train.shape)
print(X_cv_subcategory_ohe.shape, y_cv.shape)
print(X_test_subcategory_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
subcategory_feature_names = vectorizer.get_feature_names()
```

```
After vectorizations
(49041, 30) (49041,)
(24155, 30) (24155,)
(36052, 30) (36052,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'e
nvironmentalscience', 'esl', 'extracurricular', 'financialliteracy', 'foreign
languages', 'gym_fitness', 'health_lifescience', 'health_wellness', 'history_
geography', 'literacy', 'literature_writing', 'mathematics', 'music', 'nutrit
ioneducation', 'other', 'parentinvolvement', 'performingarts', 'socialscience
s', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
==========================================================================
======================
```

In [100]:
```
len(subcategory_feature_names)
```

Out[100]:
```
30
```

Encoding Numerical Features

In [56]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['price'].values.reshape(1,-1))

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

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

```
After vectorizations
(1, 49041) (49041,)
(1, 24155) (24155,)
(1, 36052) (36052,)
====================================================================================================
```

In [57]:
```
# Checking out X_train is properly normalized or not
X_train_price_norm[0:5]
```

Out[57]:
```
array([[0.0087587 , 0.00213867, 0.00569307, ..., 0.00690668, 0.00155995,
        0.00374854]])
```

In [58]:
```
X_train_price_norm = X_train_price_norm.reshape(-1,1)
X_train_price_norm[0:5]
```

Out[58]:
```
array([[0.0087587 ],
       [0.00213867],
       [0.00569307],
       [0.00181757],
       [0.00093939]])
```

In [59]:
```
# reshaping the ndarrays to -1,1 to avoid concatenation problems
X_cv_price_norm = X_cv_price_norm.reshape(-1,1)
X_test_price_norm = X_test_price_norm.reshape(-1,1)
```

In [60]:
```
print(X_train_price_norm.shape, y_train.shape)
print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
```

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

In [61]:
```
# teacher previously posted projects
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.
reshape(1,-1))

X_train_teach_prev_norm = normalizer.transform(X_train['teacher_number_of_prev
iously_posted_projects'].values.reshape(1,-1))
X_cv_teach_prev_norm = normalizer.transform(X_cv['teacher_number_of_previously
_posted_projects'].values.reshape(1,-1))
X_test_teach_prev_norm = normalizer.transform(X_test['teacher_number_of_previo
usly_posted_projects'].values.reshape(1,-1))

print("After vectorizations")
print(X_train_teach_prev_norm.shape, y_train.shape)
print(X_cv_teach_prev_norm.shape, y_cv.shape)
print(X_test_teach_prev_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(1, 49041) (49041,)
(1, 24155) (24155,)
(1, 36052) (36052,)
====================================================================================
========================
```

In [62]:
```
# reshaping the ndarrays post normalization
X_train_teach_prev_norm = X_train_teach_prev_norm.reshape(-1,1)
X_cv_teach_prev_norm = X_cv_teach_prev_norm.reshape(-1,1)
X_test_teach_prev_norm = X_test_teach_prev_norm.reshape(-1,1)
print(X_train_teach_prev_norm.shape, y_train.shape)
print(X_cv_teach_prev_norm.shape, y_cv.shape)
print(X_test_teach_prev_norm.shape, y_test.shape)
```

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

2.3 Make Data Model Ready: encoding eassay, and project_title

In [101]:
```
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data
```

Out[101]:
```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.int64'>, encoding='utf-8', input='conten
t',
                lowercase=True, max_df=1.0, max_features=None, min_df=10,
                ngram_range=(1, 1), preprocessor=None, stop_words=None,
                strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, vocabulary=None)
```

In [102]:
```
# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['essay'].values)
X_cv_essay_bow = vectorizer.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer.transform(X_test['essay'].values)
bow_essay_feature_names = vectorizer.get_feature_names()
```

In [103]:
```
print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 12142) (49041,)
(24155, 12142) (24155,)
(36052, 12142) (36052,)
====================================================================================
======================
```

In [104]:
```
len(bow_essay_feature_names)
```

Out[104]:
```
12142
```

In [105]:
```
# Preprocessing project_title
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(X_train['project_title'].values) # fit has to happen only on tr
ain data
```

Out[105]:
```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.int64'>, encoding='utf-8', input='conten
t',
                lowercase=True, max_df=1.0, max_features=None, min_df=10,
                ngram_range=(1, 1), preprocessor=None, stop_words=None,
                strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, vocabulary=None)
```

In [106]:
```
# we use the fitted CountVectorizer to convert the text to vector
X_train_pj_title_bow = vectorizer.transform(X_train['project_title'].values)
X_cv_pj_title_bow = vectorizer.transform(X_cv['project_title'].values)
X_test_pj_title_bow = vectorizer.transform(X_test['project_title'].values)
bow_titles_feature_names = vectorizer.get_feature_names()
```

In [107]:
```
print("After vectorizations")
print(X_train_pj_title_bow.shape, y_train.shape)
print(X_cv_pj_title_bow.shape, y_cv.shape)
print(X_test_pj_title_bow.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 2093) (49041,)
(24155, 2093) (24155,)
(36052, 2093) (36052,)
====================================================================================================
```

In [108]:
```
len(bow_titles_feature_names)
```

Out[108]: 2093

2.4 Appling NB() on different kind of featurization as mentioned in the instructions

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

2.4.1 Applying Naive Bayes on BOW, SET 1

In [109]:
```
# concatinating all the features
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_bow, X_train_state_ohe, X_train_teacher_ohe,
               X_train_grade_ohe, X_train_price_norm, X_train_category_ohe,
               X_train_subcategory_ohe, X_train_teach_prev_norm,
               X_train_pj_title_bow)).tocsr()

X_cr = hstack((X_cv_essay_bow, X_cv_state_ohe, X_cv_teacher_ohe,
               X_cv_grade_ohe, X_cv_category_ohe, X_cv_subcategory_ohe,
               X_cv_price_norm, X_cv_teach_prev_norm, X_cv_pj_title_bow)).tocs
r()

X_te = hstack((X_test_essay_bow, X_test_state_ohe, X_test_teacher_ohe,
               X_test_grade_ohe, X_test_category_ohe, X_test_subcategory_ohe,
               X_test_price_norm, X_test_teach_prev_norm,
               X_test_pj_title_bow)).tocsr()

# concatenating all feature names which is used later to find the best 10 feat
ures
bow_feature_names_list = []
bow_feature_names_list.extend(bow_essay_feature_names)
bow_feature_names_list.extend(school_state_feature_names)
bow_feature_names_list.extend(teacher_prefix_feature_names)
bow_feature_names_list.extend(grade_feature_names)
bow_feature_names_list.extend("Price")
bow_feature_names_list.extend(category_feature_names)
bow_feature_names_list.extend(subcategory_feature_names)
bow_feature_names_list.extend("Teacher Previously submitted projects")
bow_feature_names_list.extend(bow_titles_feature_names)
print (len(bow_feature_names_list))
```

```
14376
```

In [110]:
```
print("Final Data matrix - for set 1")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix - for set 1
(49041, 14336) (49041,)
(24155, 14336) (24155,)
(36052, 14336) (36052,)
====================================================================================
======================
```

In [111]:
```
# function to perform batch predict
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability e
stimates of the positive class
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 4904
1%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [112]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
```

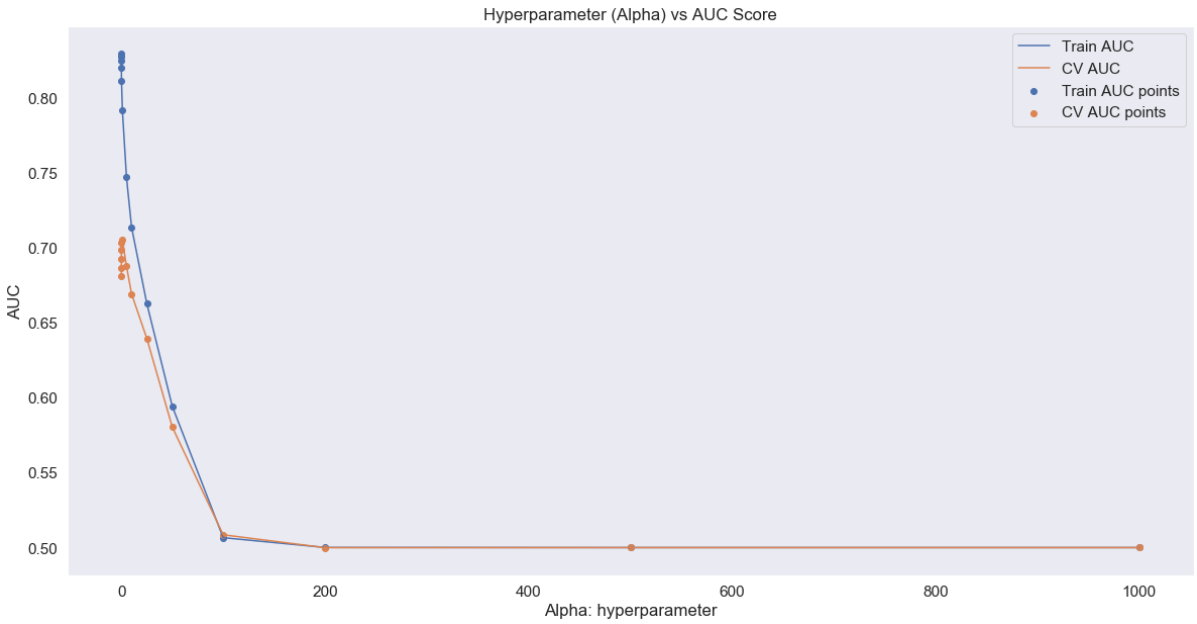
In [113]:
```
# Performing Naive Bayes on a wide range of alphas
train_auc = []
cv_auc = []
alpha = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 5, 10, 25, 50, 100, 200, 500, 1
000]
for i in tqdm(alpha):
    nb_output = MultinomialNB(alpha=i,class_prior=[0.5,0.5]) # class_prior is
 used since there is an imbalance in the dataset
    nb_output.fit(X_tr, y_train)

    y_train_pred = nb_output.predict_proba(X_tr)[:,1] # Returning the probabli
ty score of greater class label
    y_cv_pred = nb_output.predict_proba(X_cr)[:,1]

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability e
stimates of the positive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
100%|██████████| 14/14 [00:01<00:00, 11.59it/s]
```

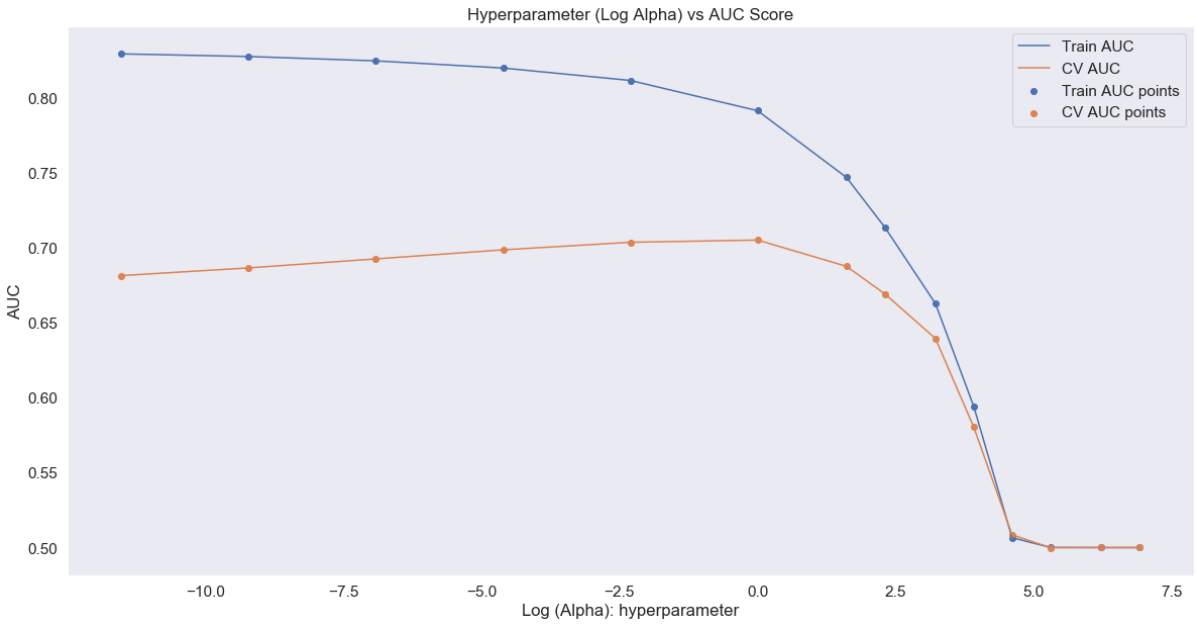
In [114]:
```
# Since plotting the alphas values directly doesn't yield good graph
# lets convert them to their log values and then plot it
# reference taken from - https://stackoverflow.com/questions/30837040/convert-float-to-log-space-in-python
from math import log
log_alphas = [log(alph) for alph in alpha]
print (alpha)
print (log_alphas)
```

```
[1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 5, 10, 25, 50, 100, 200, 500, 1000]
[-11.512925464970229, -9.210340371976182, -6.907755278982137, -4.605170185988
091, -2.3025850929940455, 0.0, 1.6094379124341003, 2.302585092994046, 3.21887
58248682006, 3.912023005428146, 4.605170185988092, 5.298317366548036, 6.21460
8098422191, 6.907755278982137]
```

In [115]:
```
plt.figure(figsize=(20,10))
plt.plot(alpha, train_auc, label='Train AUC')
plt.plot(alpha, cv_auc, label='CV AUC')

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

plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyperparameter (Alpha) vs AUC Score")
plt.grid()
plt.show()
```



In [116]:
```
plt.figure(figsize=(20,10))
plt.plot(log_alphas, train_auc, label='Train AUC')
plt.plot(log_alphas, cv_auc, label='CV AUC')

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

plt.legend()
plt.xlabel("Log (Alpha): hyperparameter")
plt.ylabel("AUC")
plt.title("Hyperparameter (Log Alpha) vs AUC Score")
plt.grid()
plt.show()
```

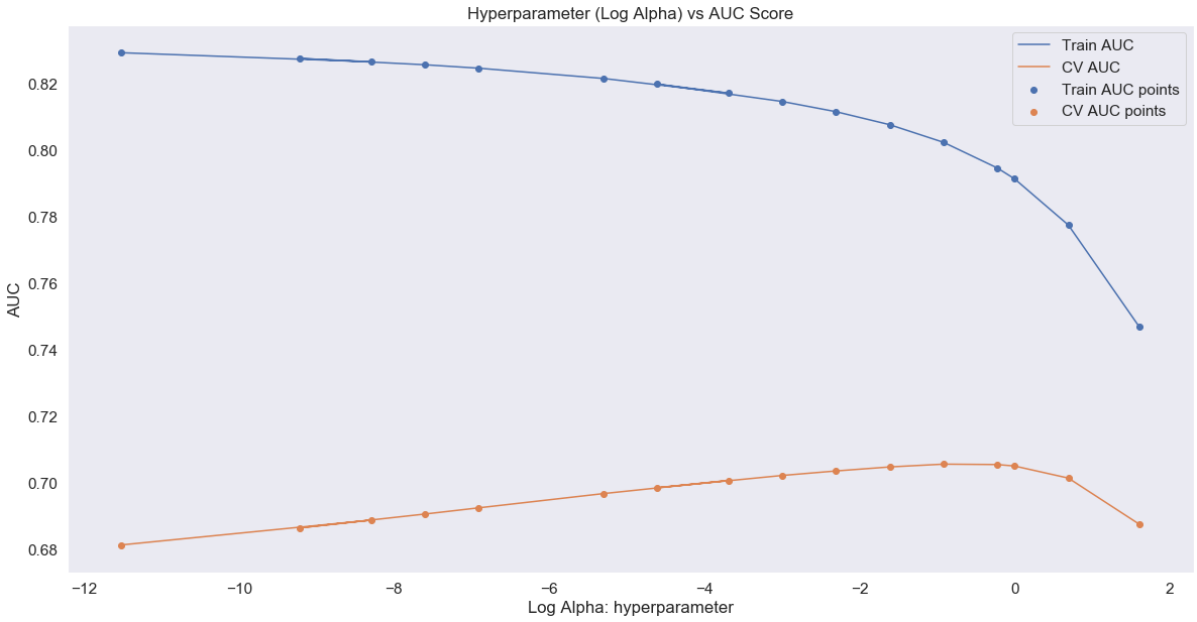


Since the alpha values at near zero is so congested, re computing the AUC with better alpha values

In [117]:
```
train_auc = []
cv_auc = []
alpha = [0.00001,0.00025, 0.0001, 0.0005, 0.001, 0.005, 0.025, 0.01, 0.05, 0.1
, 0.2, 0.4, 0.8, 1,2,5]
for i in tqdm(alpha):
    nb_output = MultinomialNB(alpha=i,class_prior=[0.5,0.5]) # class_prior is
 used since there is an imbalance in the dataset
    nb_output.fit(X_tr, y_train)

    y_train_pred = nb_output.predict_proba(X_tr)[:,1] # Returning the probabli
ty score of greater class label
    y_cv_pred = nb_output.predict_proba(X_cr)[:,1]

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability e
stimates of the positive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
100%|██████████| 16/16 [00:01<00:00, 11.59it/s]
```

In [118]:
```
# Numeric to log space conversion
log_alphas = [log(alph) for alph in alpha] # Converting alpha to log(alpha)
plt.figure(figsize=(20,10))
plt.plot(log_alphas, train_auc, label='Train AUC')
plt.plot(log_alphas, cv_auc, label='CV AUC')

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

plt.legend()
plt.xlabel("Log Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyperparameter (Log Alpha) vs AUC Score")
plt.grid()
plt.show()
```



It is evident from above two plots is that, AUC is better or infact highest at alpha = 1.0 with minum difference between Train and CV AUCs.

In [119]:
```
# Lets use GridSearchCV to find the best hyperparameter
from sklearn.model_selection import GridSearchCV
mnb_output =MultinomialNB(class_prior=[0.5,0.5])
parameters = {"alpha":np.arange(0.00001,50,0.5)}
clf = GridSearchCV(mnb_output, parameters, cv= 5, scoring='roc_auc',return_tra
in_score=True)
clf.fit(X_tr, y_train)
```

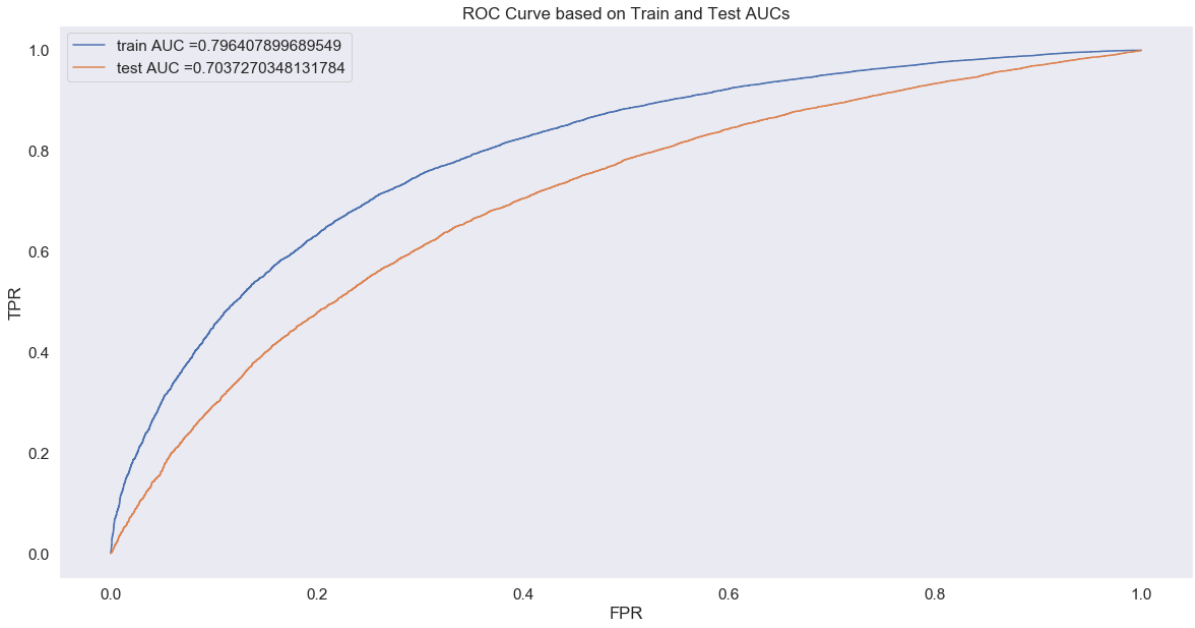
Out[119]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=MultinomialNB(alpha=1.0, class_prior=[0.5, 0.5],
                                     fit_prior=True),
             iid='warn', n_jobs=None,
             param_grid={'alpha': array([1.000000e-05, 5.000100e-01, 1.000010
e+00, 1.500010e+00,
       2.000010e+00, 2.500010e+00, 3.000010e+00, 3.500010e+00,
       4.000010e+00, 4.500010e+00, 5.000010e+00, 5.500010e+00,
       6.000010e+00, 6.500010e...
       4.000001e+01, 4.050001e+01, 4.100001e+01, 4.150001e+01,
       4.200001e+01, 4.250001e+01, 4.300001e+01, 4.350001e+01,
       4.400001e+01, 4.450001e+01, 4.500001e+01, 4.550001e+01,
       4.600001e+01, 4.650001e+01, 4.700001e+01, 4.750001e+01,
       4.800001e+01, 4.850001e+01, 4.900001e+01, 4.950001e+01])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [120]:
```
train_auc= clf.cv_results_['mean_train_score']
train_auc_std = clf.cv_results_['std_train_score']
test_auc = clf.cv_results_['mean_test_score']
test_auc_std = clf.cv_results_['std_test_score']

#Output of GridSearchCV
print('Best score: ',clf.best_score_)
print('k value with best score: ',clf.best_params_)
print('='*75)
print('Train AUC scores')
print(clf.cv_results_['mean_train_score'])
print('CV AUC scores')
print(clf.cv_results_['mean_test_score'])
```

```
Best score:  0.7049665906806502
k value with best score:  {'alpha': 0.50001}
===========================================================================
Train AUC scores
[0.854412   0.8155791  0.80416236 0.79482748 0.78659266 0.77913222
 0.77228953 0.76596603 0.76008777 0.75461359 0.7494908  0.74468517
 0.74017093 0.73590884 0.73187959 0.72806439 0.7244421  0.72099392
 0.71770636 0.71457051 0.71156921 0.70869644 0.70593781 0.70328827
 0.70073887 0.69828818 0.69592203 0.69363993 0.69144251 0.68932593
 0.68726422 0.68527572 0.68334388 0.68145625 0.67963462 0.67788727
 0.67617782 0.67452156 0.67287835 0.67131734 0.66976994 0.66827186
 0.66676334 0.66536771 0.66389006 0.66233099 0.66093832 0.65952231
 0.65813195 0.6566575  0.65532099 0.65389631 0.65250793 0.65117597
 0.64983339 0.64834062 0.64676972 0.64521317 0.64351891 0.64195258
 0.63997609 0.63806833 0.6364198  0.63471665 0.63309058 0.63076642
 0.62888572 0.62754977 0.62564588 0.62392307 0.62163618 0.61956725
 0.61713558 0.61524804 0.61244961 0.61034586 0.60782216 0.60559237
 0.60301024 0.60056545 0.59819836 0.59571129 0.59366576 0.59168458
 0.58996665 0.58778821 0.58579106 0.58365289 0.58157274 0.57944658
 0.57711866 0.57445338 0.57235602 0.57024522 0.5677889  0.56569722
 0.56352936 0.56140532 0.55921296 0.55746746]
CV AUC scores
[0.67076426 0.70496659 0.70295657 0.70004076 0.69690757 0.69371128
 0.69050211 0.68743795 0.68446687 0.68162219 0.67888721 0.67627196
 0.67380288 0.67144108 0.66916235 0.6669841  0.66490735 0.66293381
 0.66103178 0.65920985 0.65743704 0.65574126 0.65410319 0.65250257
 0.65097812 0.64951072 0.6480974  0.64672162 0.64539114 0.6441043
 0.64284513 0.64160714 0.64045122 0.639282   0.63817524 0.63711721
 0.63608411 0.63505675 0.6340896  0.63310911 0.63216718 0.63118025
 0.63022582 0.62919082 0.62826998 0.62725248 0.62648588 0.6254993
 0.62457751 0.6235181  0.62280253 0.62164512 0.62072796 0.61950748
 0.61864268 0.61745585 0.61609026 0.6147207  0.61353859 0.61249685
 0.61111925 0.61055256 0.60910342 0.60695977 0.60555481 0.60438847
 0.60274971 0.60142529 0.59917535 0.59694967 0.59640774 0.59467832
 0.59281608 0.59137477 0.58908933 0.58737845 0.58513541 0.58317544
 0.58081008 0.57868556 0.57677897 0.57444302 0.57389994 0.57317785
 0.57105632 0.56863466 0.56718476 0.56570159 0.56404213 0.56218375
 0.55903287 0.55687459 0.55544008 0.55353836 0.55213897 0.54944676
 0.54819108 0.54652895 0.54494655 0.54315143]
```

In [121]:
```
best_alpha = 0.70
from sklearn.metrics import roc_curve, auc


nb_output = MultinomialNB(alpha = best_alpha)
nb_output.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estim
ates of the positive class
# not the predicted outputs

y_train_pred = nb_output.predict_proba(X_tr)[:,1]  # returning probability est
imates of positive class
y_test_pred = nb_output.predict_proba(X_te)[:,1]

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

In [122]:
```
plt.figure(figsize=(20,10))
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tp
r)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve based on Train and Test AUCs")
plt.grid()
plt.show()
```



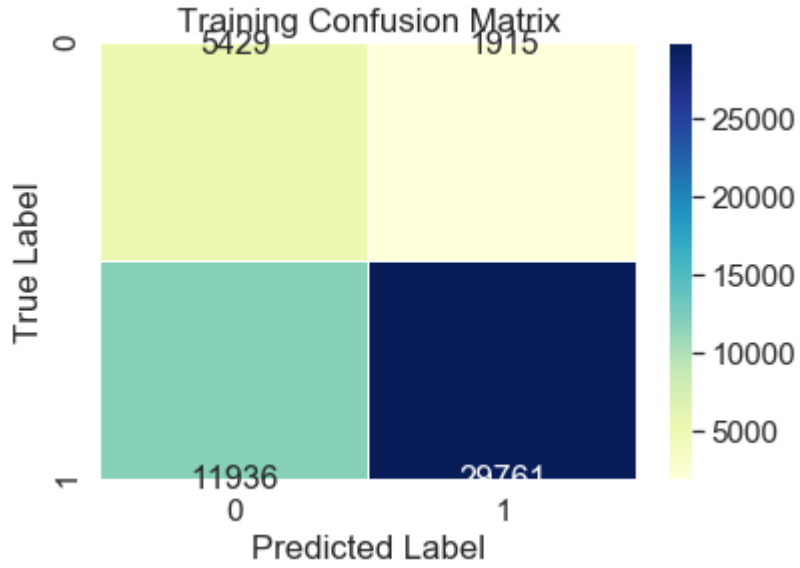
In [123]:
```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very hi
gh
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshol
d", 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 [124]:
```
# Drawing the confusion matrix as a Seaborn Heatmap
import seaborn as sns
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
Train_CM = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t
))
Test_CM = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
print("Train confusion matrix")
print(Train_CM)
print("Test confusion matrix")
print(Test_CM)
```

```
====================================================================================
======================
the maximum value of tpr*(1-fpr) 0.52763048957875 for threshold 0.872
Train confusion matrix
[[ 5429  1915]
 [11936 29761]]
Test confusion matrix
[[ 3277  2182]
 [ 9020 21573]]
```

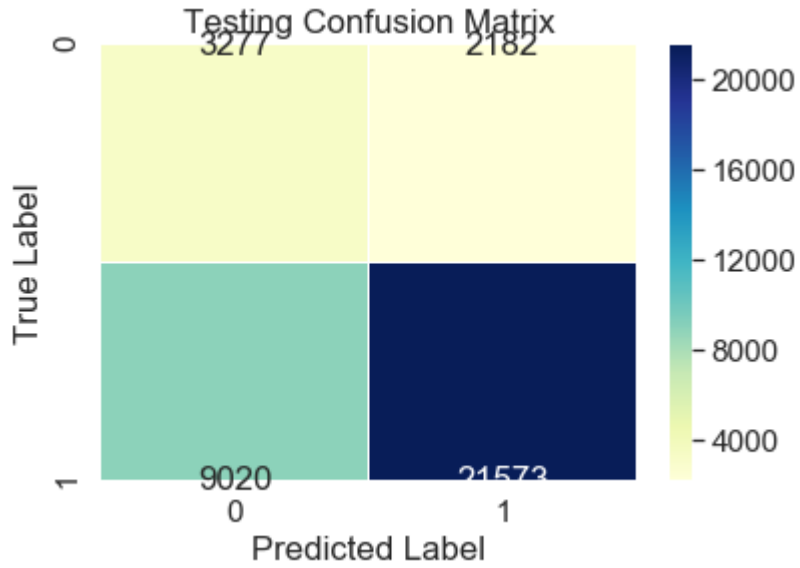
In [125]:
```
sns.set(font_scale=1.4)
sns.heatmap(Train_CM,annot=True,cbar=True,fmt="g", annot_kws = {"size":16},lin
ewidths=.5,cmap="YlGnBu")
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Training Confusion Matrix')
```

Out[125]: Text(0.5, 1, 'Training Confusion Matrix')



In [126]:
```
sns.heatmap(Test_CM,annot=True,cbar=True,fmt="d", linewidths=.5,cmap="YlGnBu")
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Testing Confusion Matrix')
```

Out[126]: Text(0.5, 1, 'Testing Confusion Matrix')



Code to find the top 10 features from each class

2.4.1.1 Top 10 important features of zero class from SET 1

In [127]:
```
# the attribute feature_log_prob_ contains the log probabilities of each featu
re.
# From X_test.shape, you can see that there were 10101 features
nb_output.feature_log_prob_
```

Out[127]:
```
array([[-10.83683476,  -9.85888985,  -8.53102905, ..., -12.92089525,
        -12.36658451, -12.0120395 ],
       [-11.12474431,  -9.81108736,  -8.45807641, ..., -12.82279307,
        -12.66622401, -12.44986672]])
```

In [128]:
```
# Length of (nb_output.feature_log_prob_[0] == nb_output.feature_log_prob_[1]
 ==  No.of features == 10101)
print (len(nb_output.feature_log_prob_[0]))
len(nb_output.feature_log_prob_[1])
```

```
14336
```

Out[128]:
```
14336
```

In [129]:
```
nb_output.classes_
```

Out[129]:
```
array([0, 1], dtype=int64)
```

In [130]:
```
# Using Numpy we can sort these arrays and retrieve the indices which result i
n the highest log probability
# code snippet from https://stackoverflow.com/questions/6910641/how-do-i-get-i
ndices-of-n-maximum-values-in-a-numpy-array
# using numpy argpartitition to retrieve top 10 features
class_zero_top_10_features = np.argpartition(nb_output.feature_log_prob_[0], -
10)[-10:]
print (class_zero_top_10_features) # top 10 args
print (nb_output.feature_log_prob_[0][class_zero_top_10_features]) # respectiv
e values
```

```
[10883  5154 10917  6256  7305 10462  6260  7118  2043  9490]
[-4.90718434 -4.89827771 -4.87664927 -4.86131463 -4.83657262 -3.08442435
 -4.49535449 -4.55921292 -4.65105292 -4.17814956]
```

2.4.1.2 Top 10 important features of one class from SET 1

In [131]:
```
class_one_top_10_features = np.argpartition(nb_output.feature_log_prob_[1], -1
0)[-10:]
print (class_one_top_10_features) # top 10 args
print (nb_output.feature_log_prob_[1][class_one_top_10_features]) # respective
values
```

```
[ 5154  6256  7305 10917 10883  6260  9490  2043 10462  7118]
[-4.95180525 -4.91318739 -4.87080214 -4.86390607 -4.81914671 -4.57254626
 -4.2119274  -4.60577854 -3.07028707 -4.52661716]
```

In [132]:
```
print('Top 10 features from negative class:')
print(np.take(bow_feature_names_list, class_zero_top_10_features))
print('-*'*50)
print('Top 10 features from positive class:')
print(np.take(bow_feature_names_list, class_one_top_10_features))
```

```
Top 10 features from negative class:
['the' 'help' 'they' 'learn' 'not' 'students' 'learning' 'my' 'classroom'
 'school']
-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
*-*-*-*-*-*-*-*-*-*-*-*
Top 10 features from positive class:
['help' 'learn' 'not' 'they' 'the' 'learning' 'school' 'classroom'
 'students' 'my']
```

2.4.2 Applying Naive Bayes on TFIDF, SET 2

In [133]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train["essay"].values)
X_train_essay_tfidf = vectorizer.transform(X_train['essay'].values)
X_cv_essay_tfidf = vectorizer.transform(X_cv['essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values)

print("Shape of Datamatrix after TFIDF Vectorization")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
tfidf_essay_feature_names = vectorizer.get_feature_names()
```

```
Shape of Datamatrix after TFIDF Vectorization
(49041, 5000) (49041,)
(24155, 5000) (24155,)
(36052, 5000) (36052,)
=============================================================================
=======================
```

In [134]:
```
# Similarly you can vectorize for title also
vectorizer_titles = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_features
=5000)
vectorizer_titles.fit(X_train["project_title"])

X_train_pj_title_tfidf = vectorizer.transform(X_train['project_title'].values)
X_cv_pj_title_tfidf = vectorizer.transform(X_cv['project_title'].values)
X_test_pj_title_tfidf = vectorizer.transform(X_test['project_title'].values)

print("Shape of Datamatrix after TFIDF Vectorization")
print(X_train_pj_title_tfidf.shape, y_train.shape)
print(X_cv_pj_title_tfidf.shape, y_cv.shape)
print(X_test_pj_title_tfidf.shape, y_test.shape)
print("="*100)
tfidf_pj_titles_feature_names = vectorizer_titles.get_feature_names()
```

```
Shape of Datamatrix after TFIDF Vectorization
(49041, 5000) (49041,)
(24155, 5000) (24155,)
(36052, 5000) (36052,)
====================================================================================
======================
```

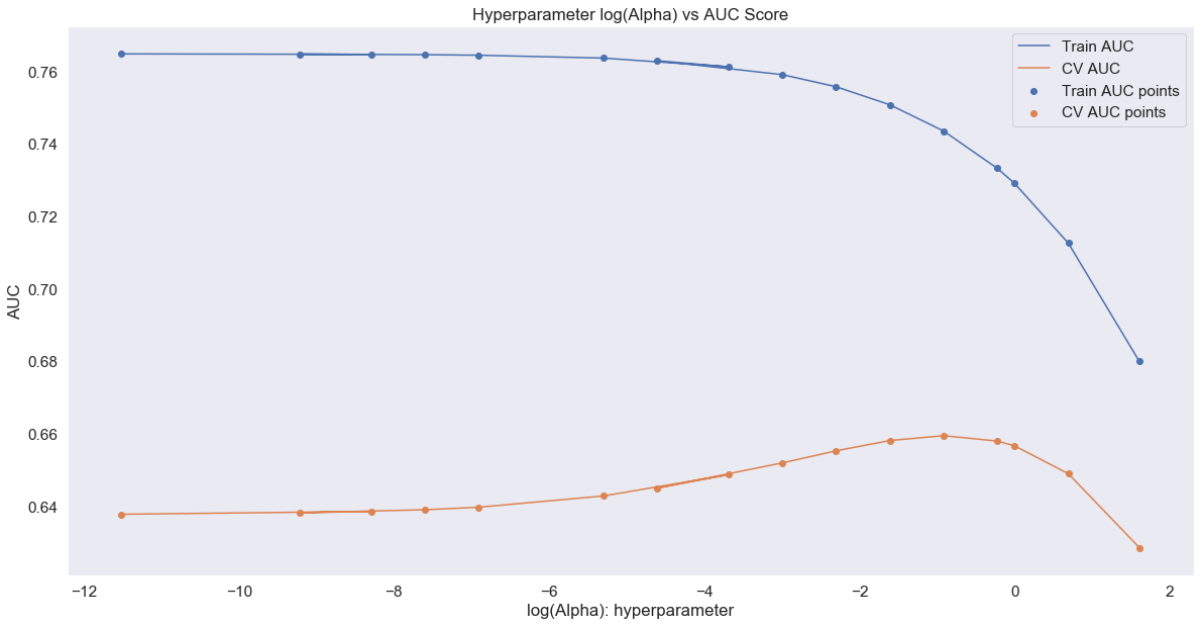
In [135]:
```
# Concatinating all the features for Set 2

X_tr = hstack((X_train_essay_tfidf, X_train_state_ohe, X_train_teacher_ohe,
               X_train_grade_ohe, X_train_price_norm, X_train_category_ohe,
               X_train_subcategory_ohe, X_train_teach_prev_norm,
               X_train_pj_title_tfidf)).tocsr()

X_cr = hstack((X_cv_essay_tfidf, X_cv_state_ohe, X_cv_teacher_ohe,
               X_cv_grade_ohe, X_cv_category_ohe, X_cv_subcategory_ohe,
               X_cv_price_norm, X_cv_teach_prev_norm, X_cv_pj_title_tfidf)).to
csr()

X_te = hstack((X_test_essay_tfidf, X_test_state_ohe, X_test_teacher_ohe,
               X_test_grade_ohe, X_test_category_ohe, X_test_subcategory_ohe,
               X_test_price_norm, X_test_teach_prev_norm,
               X_test_pj_title_tfidf)).tocsr()

tfidf_feature_names_list = []
tfidf_feature_names_list.extend(tfidf_essay_feature_names)
tfidf_feature_names_list.extend(school_state_feature_names)
tfidf_feature_names_list.extend(teacher_prefix_feature_names)
tfidf_feature_names_list.extend(grade_feature_names)
tfidf_feature_names_list.extend("Price")
tfidf_feature_names_list.extend(category_feature_names)
tfidf_feature_names_list.extend(subcategory_feature_names)
tfidf_feature_names_list.extend("Teacher Previously submitted projects")
tfidf_feature_names_list.extend(tfidf_pj_titles_feature_names)
print (len(tfidf_feature_names_list))
```

```
9201
```

In [138]:
```
train_auc = []
cv_auc = []
alpha = [0.00001,0.00025, 0.0001, 0.0005, 0.001, 0.005, 0.025, 0.01, 0.05, 0.1
, 0.2, 0.4, 0.8, 1, 2, 5]
for i in tqdm(alpha):
    nb_output = MultinomialNB(alpha=i,class_prior=[0.5,0.5]) # class_prior is
 used since there is an imbalance in the dataset
    nb_output.fit(X_tr, y_train)

    y_train_pred = nb_output.predict_proba(X_tr)[:,1] # Returning the probabli
ty score of greater class label
    y_cv_pred = nb_output.predict_proba(X_cr)[:,1]

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability e
stimates of the positive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

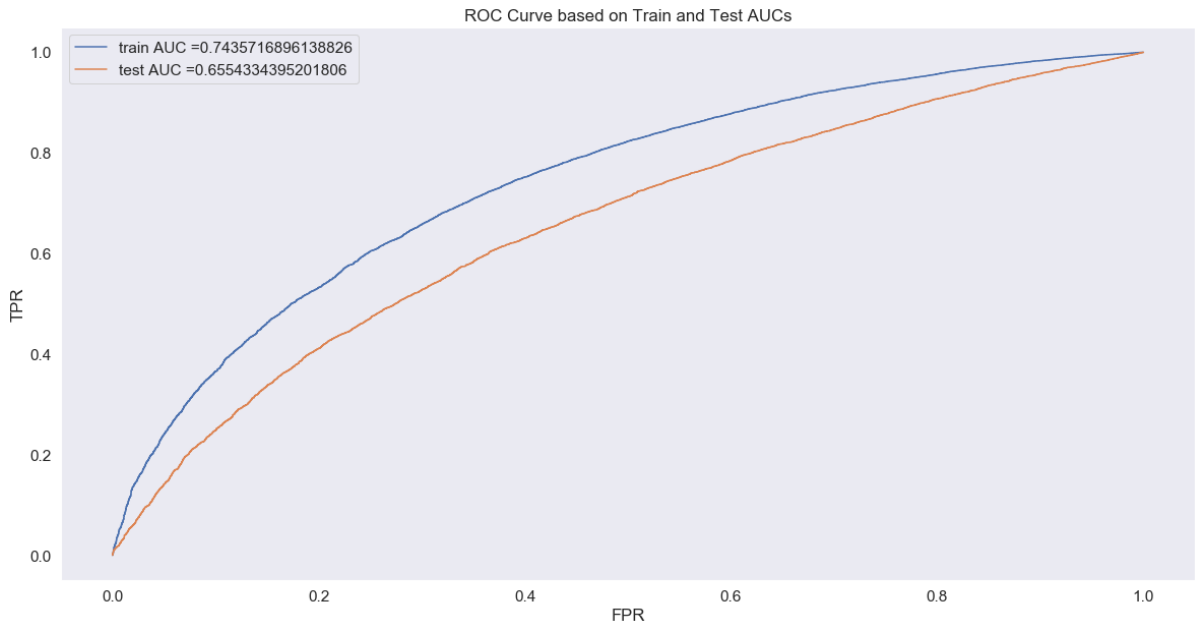
```
100%|██████████| 16/16 [00:01<00:00, 10.12it/s]
```

In [139]:
```
log_alphas = [log(alph) for alph in alpha]
plt.figure(figsize=(20,10))
plt.plot(log_alphas, train_auc, label='Train AUC')
plt.plot(log_alphas, cv_auc, label='CV AUC')

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

plt.legend()
plt.xlabel("log(Alpha): hyperparameter")
plt.ylabel("AUC")
plt.title("Hyperparameter log(Alpha) vs AUC Score")
plt.grid()
plt.show()
```



In [140]:
```
best_alpha = 0.4 # from graph it looks like 0.4 is best alpha
from sklearn.metrics import roc_curve, auc


nb_output = MultinomialNB(alpha = best_alpha)
nb_output.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estim
ates of the positive class
# not the predicted outputs

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

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

In [141]:
```
plt.figure(figsize=(20,10))
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tp
r)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve based on Train and Test AUCs")
plt.grid()
plt.show()
```

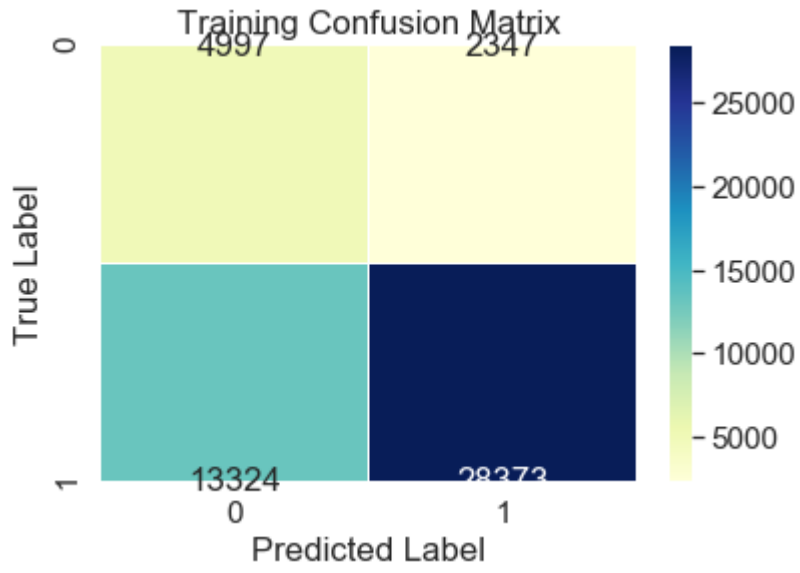


In [142]:
```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
Train_CM = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t
))
Test_CM = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
print("Train confusion matrix")
print(Train_CM)
print("Test confusion matrix")
print(Test_CM)
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.46299588344129916 for threshold 0.859
Train confusion matrix
[[ 4997  2347]
 [13324 28373]]
Test confusion matrix
[[ 2501  2958]
 [ 7773 22820]]
```

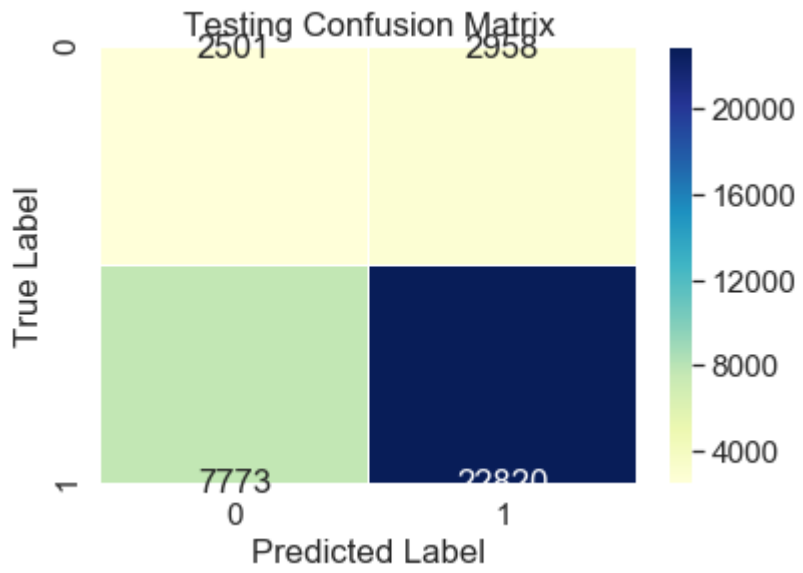
In [143]:
```
sns.heatmap(Train_CM,annot=True,cbar=True,fmt="d", linewidths=.5,cmap="YlGnBu"
)
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Training Confusion Matrix')
```

Out[143]: Text(0.5, 1, 'Training Confusion Matrix')



In [144]:
```
sns.heatmap(Test_CM,annot=True,cbar=True,fmt="d", linewidths=.5,cmap="YlGnBu")
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Testing Confusion Matrix')
```

Out[144]: Text(0.5, 1, 'Testing Confusion Matrix')



In [145]:
```
# the attribute feature_log_prob_ contains the log probabilities of each featu
re.
# From X_test.shape, you can see that there were 10101 features
nb_output.feature_log_prob_
```

Out[145]:
```
array([[-10.02318744,  -8.98982875,  -8.38722971, ..., -12.72689049,
        -12.72689049, -10.74304133],
       [ -9.97525177,  -8.89554082,  -8.38968025, ..., -14.45055565,
        -14.45055565, -10.96125015]])
```

2.4.2.1 Top 10 important features of positive class from SET 2

In [146]:
```
# Please write all the code with proper documentation
class_one_top_10_features = np.argpartition(nb_output.feature_log_prob_[1], -1
0)[-10:]
print (class_one_top_10_features) # top 10 args
print (nb_output.feature_log_prob_[1][class_one_top_10_features]) # respective
values
```

```
[5057 5088 5089 5087 5056 5054 5066 5053 5059 5065]
[-4.76485104 -4.47901233 -4.25314927 -4.0480064  -3.96085211 -3.93293707
 -3.86926301 -3.53594912 -3.80761002 -3.61641106]
```

2.4.2.2 Top 10 important features of negative class from SET 2

In [147]:
```
# Please write all the code with proper documentation
class_zero_top_10_features = np.argpartition(nb_output.feature_log_prob_[0], -
10)[-10:]
print (class_zero_top_10_features) # top 10 args
print (nb_output.feature_log_prob_[0][class_zero_top_10_features]) # respectiv
e values
```

```
[5057 5088 5087 5056 5089 5054 5066 5059 5065 5053]
[-4.73868689 -4.57541299 -4.26499862 -4.03012953 -4.23995034 -3.90316894
 -3.82566014 -3.79944368 -3.7703462  -3.59386322]
```

In [148]:
```
print('Top 10 features from negative class:')
print(np.take(bow_feature_names_list, class_zero_top_10_features))
print('-*'*50)
print('Top 10 features from positive class:')
print(np.take(bow_feature_names_list, class_one_top_10_features))
```

```
Top 10 features from negative class:
['happiness' 'hatching' 'hatched' 'happily' 'hate' 'happier' 'hardships'
 'harbor' 'hardship' 'happens']
-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
*-*-*-*-*-*-*-*-*-*-*-*
Top 10 features from positive class:
['happiness' 'hatching' 'hate' 'hatched' 'happily' 'happier' 'hardships'
 'happens' 'harbor' 'hardship']
```

3. Conclusions

In [149]:
```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyperparameter-Alpha", "Train AUC",
"Test AUC"]

x.add_row(["BOW","Brute",1,0.79,0.70])
x.add_row(["TFIDF","Brute",0.4,0.74,0.65])
print(x)
```

```
+------------+-------+----------------------+-----------+----------+
| Vectorizer | Model | Hyperparameter-Alpha | Train AUC | Test AUC |
+------------+-------+----------------------+-----------+----------+
|    BOW     | Brute |          1           |    0.79   |   0.7    |
|   TFIDF    | Brute |         0.4          |    0.74   |   0.65   |
+------------+-------+----------------------+-----------+----------+
```