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](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:** 2016-0 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 te **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')
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://stackoverflow.com/a/47301924/4084039

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

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [6]:

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

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

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

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

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

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

1.3 Text preprocessing

In [7]:

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

In [8]:

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

Out[8]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_
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 classro
oms. 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 mo
re.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 chil
d 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 t
hank you cards will be used throughout the year by the students as they creat
e thank you cards to their team groups.\r\n\r\nYour generous donations will h
elp 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 g
et 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 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 don't want t
o 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 d
emonstrates. The great teacher inspires. -William A. Ward\r\n\r\nMy school ha
s 803 students which is makeup is 97.6% African-American, making up the large
st 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 lun
ch. We aren't receiving doctors, lawyers, or engineers children from rich bac
kgrounds or neighborhoods. As an educator I am inspiring minds of young child
ren and we focus not only on academics but one smart, effective, efficient, a
nd disciplined students with good character.In our classroom we can utilize t
he Bluetooth for swift transitions during class. I use a speaker which does
n'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 th
e lessons as meaningful. But with the bluetooth speaker my students will be a
ble 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 le
tter, words and pictures for students to learn about different letters and it
```

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

In [11]:

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

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

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

In [12]:

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

```
My kindergarten students have varied disabilities ranging from speech and 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, 2269.98it/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))
project_data['project_title']=preprocessed_titles
```

```
100%|████████████████████████████████████████████████████████████████|
109248/109248 [00:02<00:00, 48383.79it/s]
```

1.5 Preparing data for models

In [20]:

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

Out[20]:

```
Index(['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'],
      dtype='object')
```

we are going to consider

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

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

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

1.5.1 Vectorizing Categorical data

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

In [21]:

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

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

In [22]:
```
# 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)
```

```
['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 [23]:
```
# 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)
    return vectorizer.transform(project_data[category].fillna(fillnan_value).v
alues)
```

In [24]:
```
# One hot encoding for school state
countries_list = sorted(project_data["school_state"].value_counts().keys())
school_state_one_hot = perform_one_hot_encoding(countries_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 [25]:
```
# 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 [26]:

```python
# 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().keys())
print (teacher_prefix_list)
teacher_prefix_one_hot = perform_one_hot_encoding(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 [27]:

```python
# One hot encoding for project_grade_category
grade_list = sorted(project_data["project_grade_category"].value_counts().keys())
grade_one_hot = perform_one_hot_encoding(grade_list, "project_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 [28]:

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

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

In [29]:

```python
# 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 [30]:

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

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

In [31]:

```
# 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)
```

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

1.5.2.3 Using Pretrained Models: Avg W2V

In [32]:

```
'''
# Reading glove vectors in python: https://stackoverflow.com/a/38230349/408403
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[32]:

```
'\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# 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 [33]:

```python
# 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 [34]:

```python
# 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 [00:26<00:00, 4117.90it/s]

109248
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [35]:

```
# 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 [36]:

```
# 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 [03:06<00:00, 586.30it/s]

109248
300
```

In [37]:

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

In [38]:
```
tfidf_w2v_vectors_titles = []; # the avg-w2v for each project_title is stored
 in this list
for sentence in tqdm(preprocessed_titles): # for each project_title
    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_titles.append(vector)

print(len(tfidf_w2v_vectors_titles))
print(len(tfidf_w2v_vectors_titles[0]))
```

```
100%|████████████████████████████████████████████████████████████████████|
109248/109248 [00:02<00:00, 38381.31it/s]

109248
300
```

1.5.3 Vectorizing Numerical features

In [39]:
```
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 [40]:
```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/s
klearn.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 32
9.   ... 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 mea
n and standard deviation of this data
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_sc
alar.var_[0])}")

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

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

In [41]:
```
price_standardized
```

Out[41]:
```
array([[-0.3905327 ],
       [ 0.00239637],
       [ 0.59519138],
       ...,
       [-0.15825829],
       [-0.61243967],
       [-0.51216657]])
```

In [42]:
```
# 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 mea
n 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_sca
lar.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_pr
eviously_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 [43]:

```
# 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 [44]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a 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[44]:

```
(109248, 20053)
```

In [45]:

```
# 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
```

Computing Sentiment Scores

In [46]:
```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

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

sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i teach the
smallest students with the biggest enthusiasm \
for learning my students learn in many different ways using all of our senses
 and multiple intelligences i use a wide range\
of techniques to help all my students succeed students in my class come from a
variety of different backgrounds which makes\
for wonderful sharing of experiences and cultures including native americans o
ur school is a caring community of successful \
learners which can be seen through collaborative student project based learnin
g in and out of the classroom kindergarteners \
in my class love to work with hands on materials and have many different oppor
tunities to practice a skill before it is\
mastered having the social skills to work cooperatively with friends is a cruc
ial aspect of the kindergarten curriculum\
montana is the perfect place to learn about agriculture and nutrition my stude
nts love to role play in our pretend kitchen\
in the early childhood classroom i have had several kids ask me can we try coo
king with real food i will take their idea \
and create common core cooking lessons where we learn important math and writi
ng concepts while cooking delicious healthy \
food for snack time my students will have a grounded appreciation for the work
that went into making the food and knowledge \
of where the ingredients came from as well as how it is healthy for their bodi
es this project would expand our learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to
make homemade applesauce make our own bread \
and mix up healthy plants from our classroom garden in the spring we will also
create our own cookbooks to be printed and \
shared with families students will gain math and literature skills as well as
 a life long enjoyment for healthy cooking \
nannan'
ss = sid.polarity_scores(for_sentiment)

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

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

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

Assignment 7: SVM

1. **[Task-1] Apply Support Vector Machines(SGDClassifier with hinge loss: Linear SVM) 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)
    - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
    - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)

2. **The hyper paramter tuning (best alpha in range [10^-4 to 10^4], and the best penalty among 'l1', 'l2')**

    - 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
    - Find the best hyper paramter using k-fold cross validation or simple cross validation data
    - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

3. **Representation of results**

    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.
    - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
    - Along with plotting ROC curve, you need to print the confusion matrix (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)

4. **[Task-2] Apply the Support Vector Machines on these features by finding the best hyper paramter as suggested in step 2 and step 3** (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

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

    - Consider these set of features Set 5 : (https://seaborn.pydata.org/generated/seaborn.heatmap.html)
        - **school_state** : categorical data
        - **clean_categories** : categorical data
        - **clean_subcategories** : categorical data
        - **project_grade_category** :categorical data
        - **teacher_prefix** : categorical data
        - **quantity** : numerical data
        - **teacher_number_of_previously_posted_projects** : numerical data
        - **price** : numerical data
        - **sentiment score's of each of the essay** : numerical data

- **number of words in the title** : numerical data
- **number of words in the combine essays** : numerical data (https://seaborn.pydata.org/generated/seaborn.heatmap.html)
- **Apply** (https://seaborn.pydata.org/generated/seaborn.heatmap.html)**TruncatedSVD (http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html) on TfidfVectorizer (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html) of essay text, choose the number of components (`n_components`) using elbow method (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/pca-code-example-using-non-visualization/)** : numerical data

- **Conclusion**
  - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link (http://zetcode.com/python/prettytable/)

Note: Data Leakage

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

2. Support Vector Machines

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

In [47]:

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

# 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[47]:

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

In [48]:

```python
# 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, stratify=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, 14) (49041,)
(24155, 14) (24155,)
(36052, 14) (36052,)
=============================================================================
=======================
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [49]:

```
# Encoding School State - OHE
# School State
vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train 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)
```

```
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 [50]:

```
# Encoding Teacher Prefix OHE
# teacher_prefix
vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on train 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)
```

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

In [51]:

```python
# Encoding project_grade_category
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only 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'].values)
X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].values)

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)
```

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

In [52]:

```python
# 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)
```

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

In [53]:

```python
# 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'].values)

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)
```

```
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']
=============================================================================
=======================
```

Encoding Numerical features

In [54]:

```
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))

X_train_price_norm = X_train_price_norm.reshape(-1,1)
X_cv_price_norm = X_cv_price_norm.reshape(-1,1)
X_test_price_norm = X_test_price_norm.reshape(-1,1)

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

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

In [55]:
```
# 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_previously_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_previously_posted_projects'].values.reshape(1,-1))

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

2.3 Make Data Model Ready: encoding eassay, and project_title

In [56]:
```
vectorizer = CountVectorizer(min_df=10, ngram_range=(2,2), max_features=5000)
# Just bigrams for Essay
vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data
```

Out[56]:
```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                lowercase=True, max_df=1.0, max_features=5000, min_df=10,
                ngram_range=(2, 2), preprocessor=None, stop_words=None,
                strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, vocabulary=None)
```

In [57]:

```
# 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)
```

In [58]:

```
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, 5000) (49041,)
(24155, 5000) (24155,)
(36052, 5000) (36052,)
=============================================================================
=======================
```

In [59]:

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

Out[59]:

```
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=1,
                ngram_range=(1, 1), preprocessor=None, stop_words=None,
                strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, vocabulary=None)
```

In [60]:

```
# 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)
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, 11565) (49041,)
(24155, 11565) (24155,)
(36052, 11565) (36052,)
=============================================================================
=======================
```

2.4 Appling Support Vector Machines on different kind of featurization as mentioned in the instructions

Apply Support Vector Machines 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

Set 1 BOW

In [61]:

```
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()
```

In [62]:

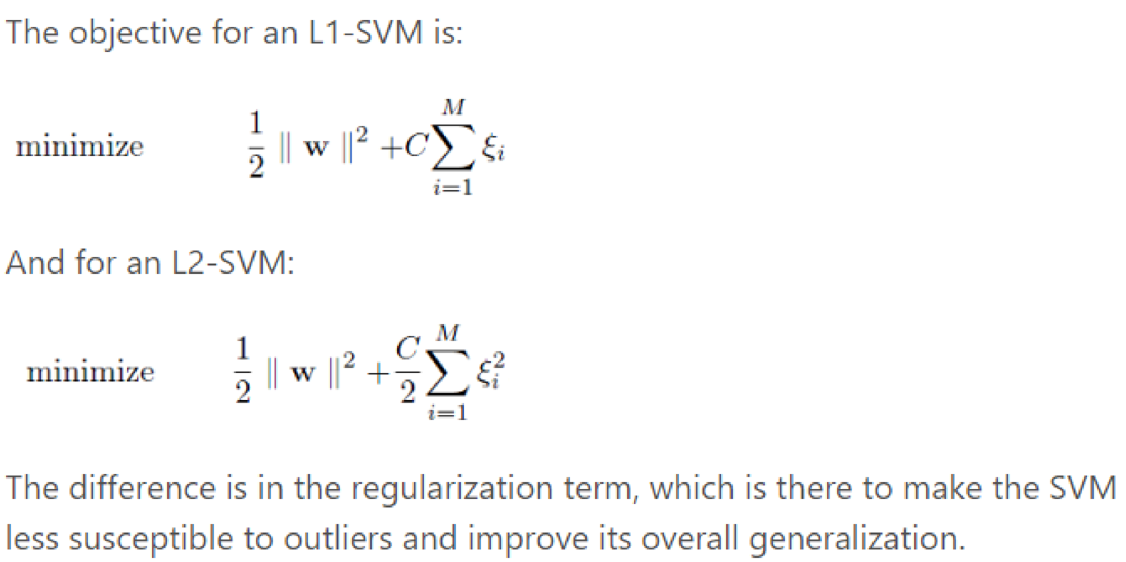
```
print("Final Data matrix - for set 1")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix - for set 1
(49041, 16666) (49041,)
(24155, 16666) (24155,)
(36052, 16666) (36052,)
=============================================================================
=======================
```

In [63]:

```
# Necessary Package imports
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn.calibration import CalibratedClassifierCV
```

Important Note: predict_proba is written for log loss and modified huber loss and not for hinge loss



The objective for an L1-SVM is:

$$\text{minimize} \qquad \frac{1}{2} \parallel \mathbf{w} \parallel^2 + C \sum_{i=1}^{M} \xi_i$$

And for an L2-SVM:

$$\text{minimize} \qquad \frac{1}{2} \parallel \mathbf{w} \parallel^2 + \frac{C}{2} \sum_{i=1}^{M} \xi_i^2$$

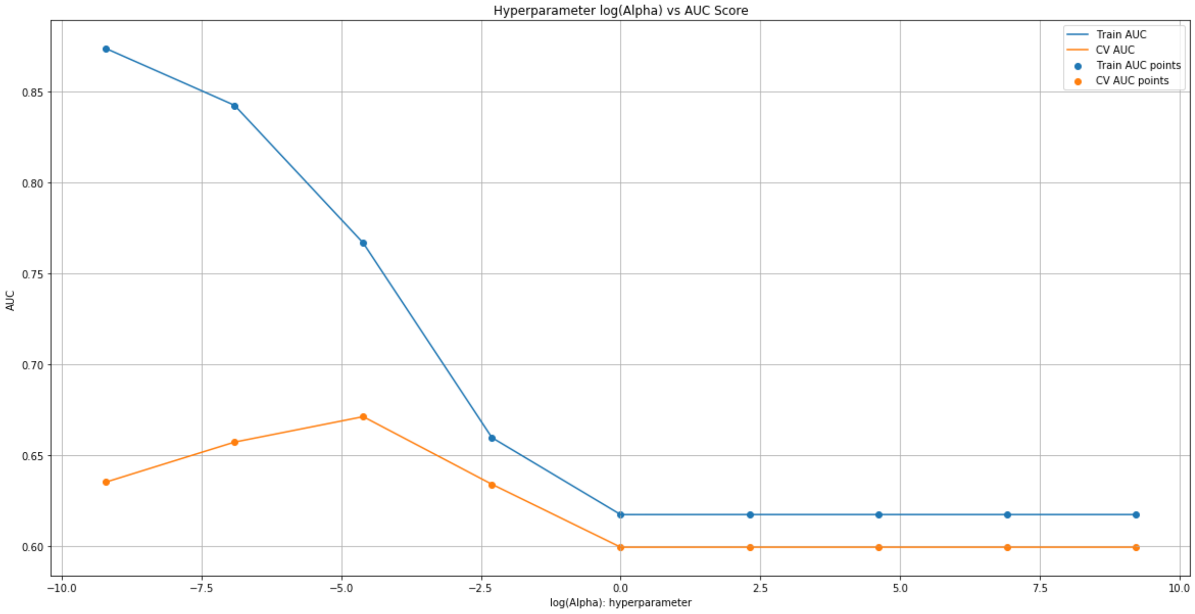
The difference is in the regularization term, which is there to make the SVM less susceptible to outliers and improve its overall generalization.

In [64]:
```
# SVM With L2 Penality
# L2 regulaization is default penality

# L1 Regularization - Lasso
# L2 Regularization - Ridge
train_auc = []
cv_auc = []
alpha = [10**-4, 10**-3,10**-2,10**-1,1,10,10**2,10**3,10**4]#alpha=1/C
for i in tqdm(alpha):
    base_estimator_svm_output_bow = SGDClassifier(loss="hinge", penalty='l2',
class_weight='balanced', alpha=i)
    # Since SGDClassifier with Hinge loss doesn't have a predict_porba functio
n lets consider this
    # as a base estimator and have a CalibrateClassiferCV on top of this
    svm_output_bow = CalibratedClassifierCV(base_estimator_svm_output_bow, cv=
3)
    svm_output_bow.fit(X_tr, y_train)

    y_train_pred = svm_output_bow.predict_proba(X_tr)[:,1] # Returning the pro
bablity score of greater class label
    y_cv_pred = svm_output_bow.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%|███████████████████████████████████████████████████████████████████████
████████████| 9/9 [00:06<00:00,  1.46it/s]
```

In [65]:

```python
# Since plotting the alphas values directly doesn't yield good graph
# lets convert them to their log values and then plot it
from math import log
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()
```

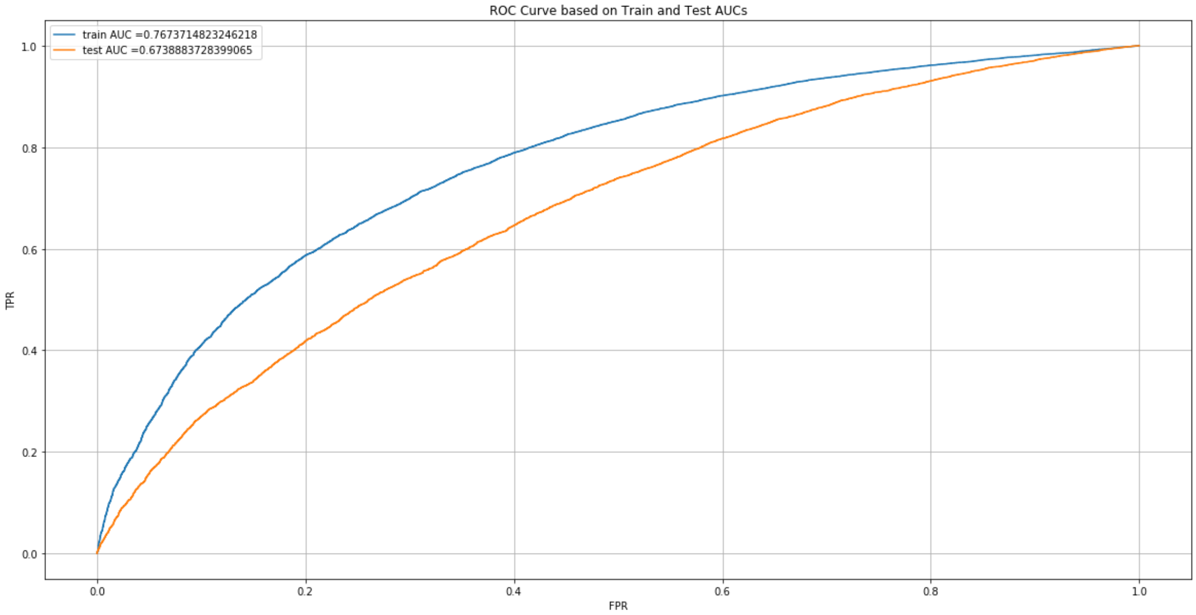


In [66]:

```
# best alpha appears to be at third point
best_alpha = 10**-2

from sklearn.metrics import roc_curve, auc


base_estimator_svm_output_bow = SGDClassifier(loss="hinge", penalty='l2', clas
s_weight='balanced', alpha=best_alpha)
svm_output_bow = CalibratedClassifierCV(base_estimator_svm_output_bow, cv=3)
svm_output_bow.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 = svm_output_bow.predict_proba(X_tr)[:,1]  # returning probabilit
y estimates of positive class
y_test_pred = svm_output_bow.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 [67]:

```
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 [68]:

```
# 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
```

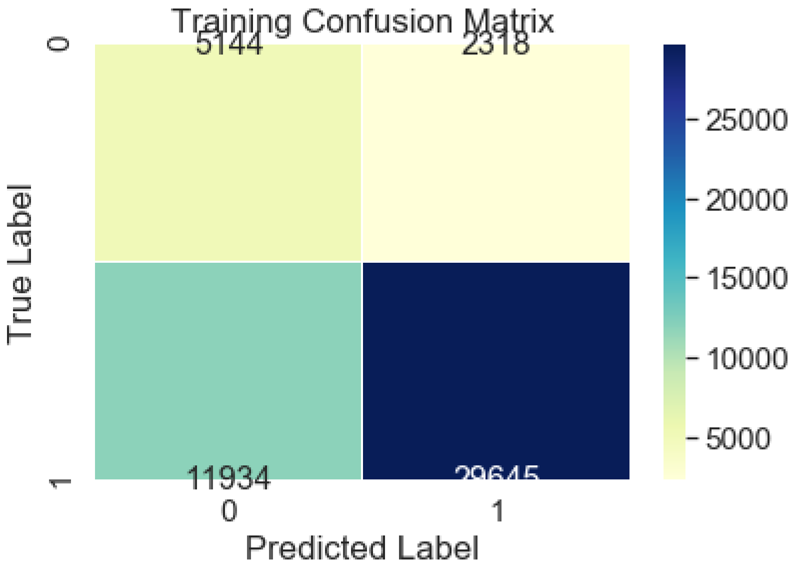
In [69]:

```
# 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) 0.49149955596631606 for threshold 0.83
Train confusion matrix
[[ 5144  2318]
 [11934 29645]]
Test confusion matrix
[[ 2724  2735]
 [ 7965 22628]]
```

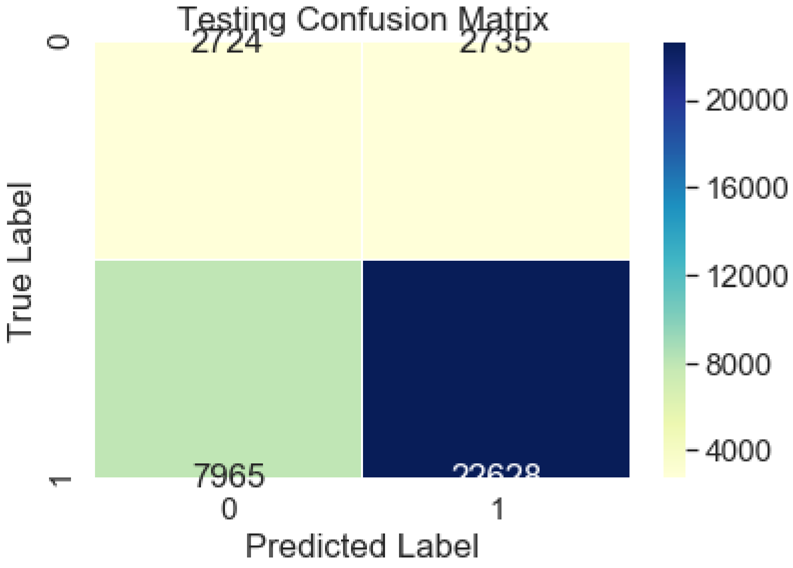
In [70]:
```
sns.set(font_scale=1.4)
sns.heatmap(Train_CM,annot=True,cbar=True,fmt="g", annot_kws = {"size":16},lin
ewidths=.5,cmap="YlGnBu")
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Training Confusion Matrix')
```

Out[70]:
```
Text(0.5, 1, 'Training Confusion Matrix')
```



In [71]:
```
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[71]:
```
Text(0.5, 1, 'Testing Confusion Matrix')
```



Set -1 BOW with L1 Penality

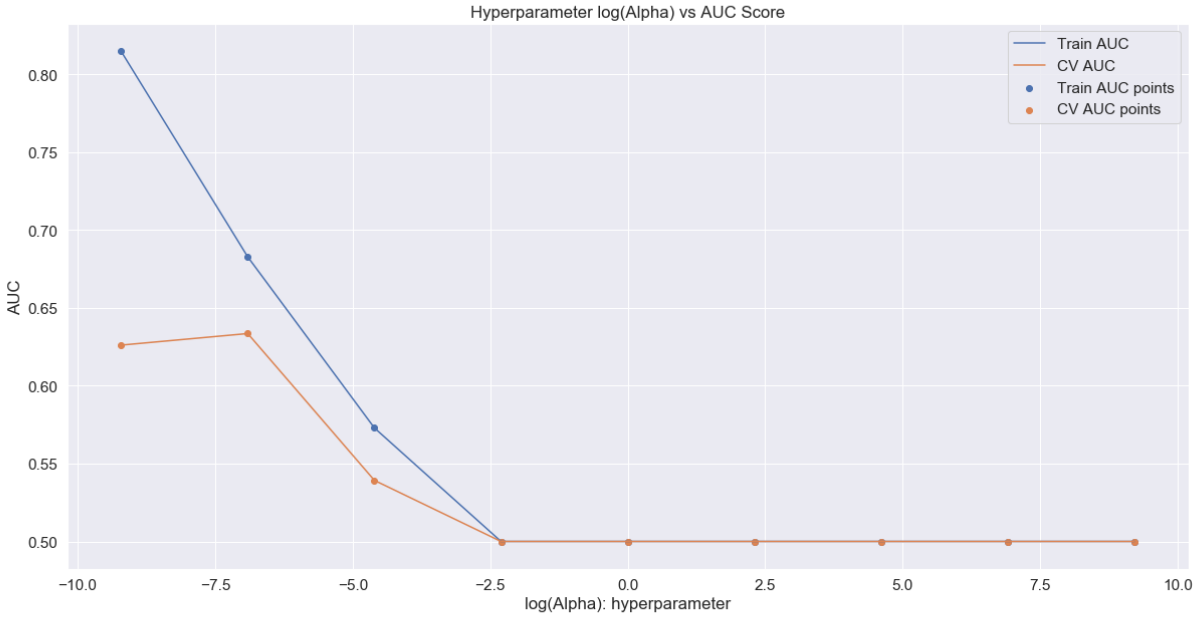
In [72]:

```
# SVM With L2 Penality
# L2 regulaization is default penality

# L1 Regularization - Lasso
# L2 Regularization - Ridge
train_auc = []
cv_auc = []
alpha = [10**-4, 10**-3,10**-2,10**-1,1,10,10**2,10**3,10**4]#alpha=1/C
for i in tqdm(alpha):
    base_estimator_svm_output_bow = SGDClassifier(loss="hinge", penalty='l1', class_weight='balanced', alpha=i)
    # Since SGDClassifier with Hinge loss doesn't have a predict_porba function lets consider this
    # as a base estimator and have a CalibrateClassiferCV on top of this
    svm_output_bow = CalibratedClassifierCV(base_estimator_svm_output_bow, cv=3)
    svm_output_bow.fit(X_tr, y_train)

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

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates 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%|████████████████████████████████████████████████████████████
████████| 9/9 [00:47<00:00,  5.26s/it]
```

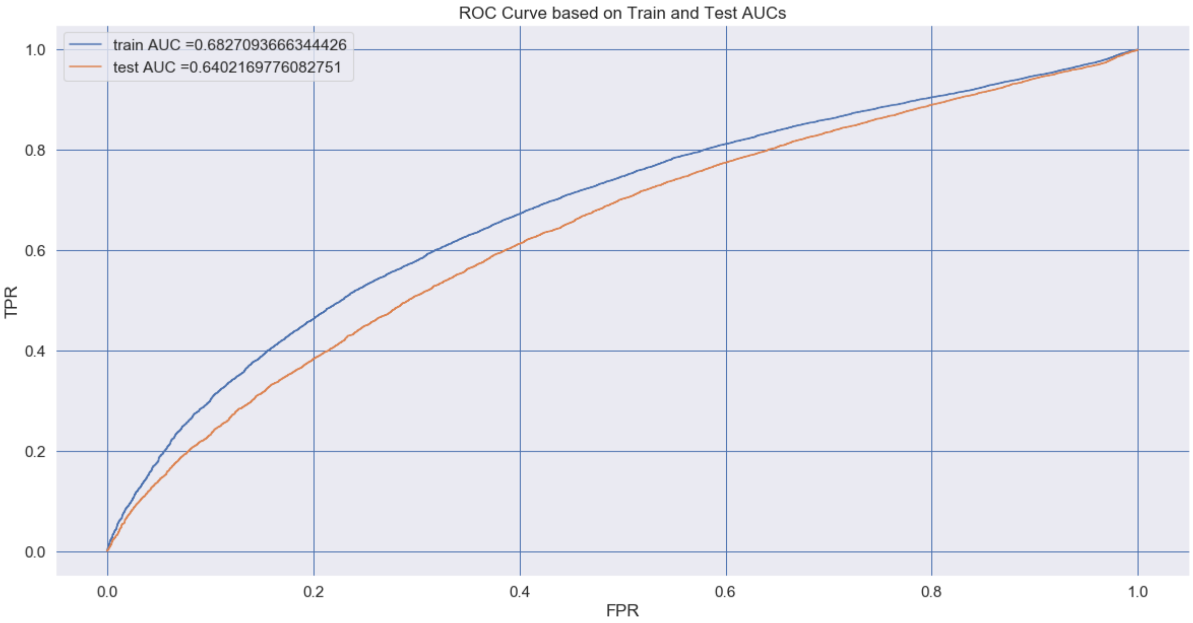
In [73]:

```
# Since plotting the alphas values directly doesn't yield good graph
# lets convert them to their log values and then plot it
from math import log
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(b=True)
plt.show()
```



In [74]:
```
# best alpha appears to be at second point
best_alpha = 10**-3

from sklearn.metrics import roc_curve, auc


base_estimator_svm_output_bow = SGDClassifier(loss="hinge", penalty='l1', clas
s_weight='balanced', alpha=best_alpha)
svm_output_bow = CalibratedClassifierCV(base_estimator_svm_output_bow, cv=3)
svm_output_bow.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 = svm_output_bow.predict_proba(X_tr)[:,1]  # returning probabilit
y estimates of positive class
y_test_pred = svm_output_bow.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 [75]:
```
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(b=True, color='b')
plt.show()
```



In [76]:

```
# 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)
```

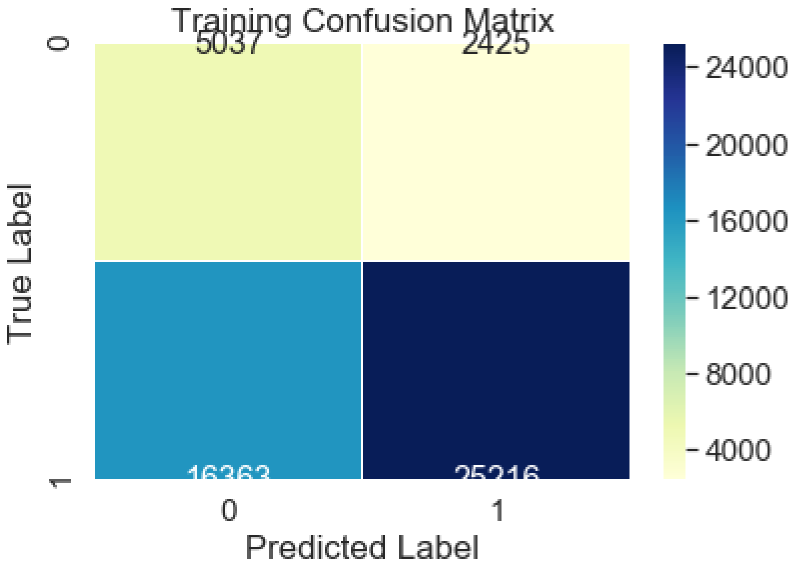
```
=============================================================================
=======================
the maximum value of tpr*(1-fpr) 0.40937268544779143 for threshold 0.849
Train confusion matrix
[[ 5037  2425]
 [16363 25216]]
Test confusion matrix
[[ 3179  2280]
 [11356 19237]]
```

In [77]:

```
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[77]:

```
Text(0.5, 1, 'Training Confusion Matrix')
```

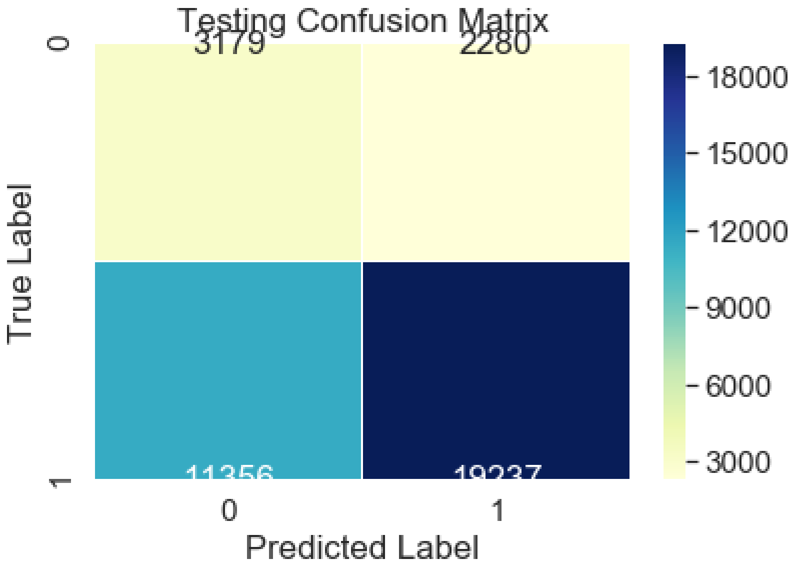


In [78]:

```
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[78]:

```
Text(0.5, 1, 'Testing Confusion Matrix')
```



L2 regularizer is better compared with L1 on set 1

In [79]:

```
# Lets do the same BOW model with GridSearchCV to find best alpha
from sklearn.model_selection import GridSearchCV
sgd_output =SGDClassifier(max_iter=1000, loss="hinge",class_weight='balanced')
parameters = {"alpha":np.arange(10**-4,10**4,5)}
clf = GridSearchCV(sgd_output, parameters, cv= 5, scoring='roc_auc',return_tra
in_score=True)
clf.fit(X_tr, y_train)
```

Out[79]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=SGDClassifier(alpha=0.0001, average=False,
                                     class_weight='balanced',
                                     early_stopping=False, epsilon=0.1,
                                     eta0=0.0, fit_intercept=True,
                                     l1_ratio=0.15, learning_rate='optimal',
                                     loss='hinge', max_iter=1000,
                                     n_iter_no_change=5, n_jobs=None,
                                     penalty='l2', power_t=0.5,
                                     random_state=None, shuffle=True, tol=0.0
01,
                                     validation_fraction=0.1, verbose=0,
                                     warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'alpha': array([1.0000000e-04, 5.0001000e+00, 1.0000
100e+01, ..., 9.9850001e+03,
       9.9900001e+03, 9.9950001e+03])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [80]:

```
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'])
```

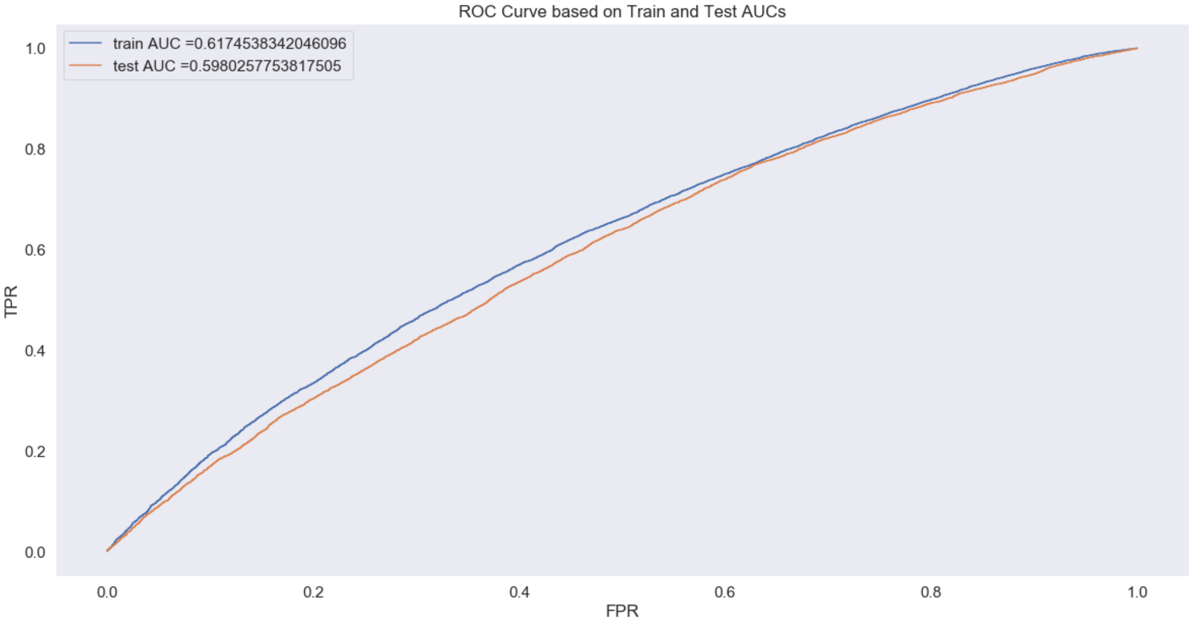
```
Best score:  0.6278003494197563
k value with best score:  {'alpha': 0.0001}
===========================================================================
Train AUC scores
[0.89667429 0.62081632 0.6208046  ... 0.62081481 0.62081481 0.62081481]
CV AUC scores
[0.62780035 0.60034562 0.60033662 ... 0.60034419 0.60034419 0.60034419]
```

In [81]:

```
# best alpha appears to be at second point
best_alpha = clf.best_score_

from sklearn.metrics import roc_curve, auc


base_estimator_svm_output_bow = SGDClassifier(loss="hinge", penalty='l2', class_weight='balanced', alpha=best_alpha)
svm_output_bow = CalibratedClassifierCV(base_estimator_svm_output_bow, cv=3)
svm_output_bow.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = svm_output_bow.predict_proba(X_tr)[:,1]  # returning probability estimates of positive class
y_test_pred = svm_output_bow.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 [82]:

```
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()
```



Set 2 TFIDF

In [83]:

```
# Vectroizing Essay and Project_Title in TFIDF Form
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
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, 41013) (49041,)
(24155, 41013) (24155,)
(36052, 41013) (36052,)
==============================================================================
======================
```

In [84]:

```
# Similarly you can vectorize for title also
vectorizer_titles = TfidfVectorizer()
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)
```

```
Shape of Datamatrix after TFIDF Vectorization
(49041, 41013) (49041,)
(24155, 41013) (24155,)
(36052, 41013) (36052,)
=============================================================================
=======================
```

In [85]:

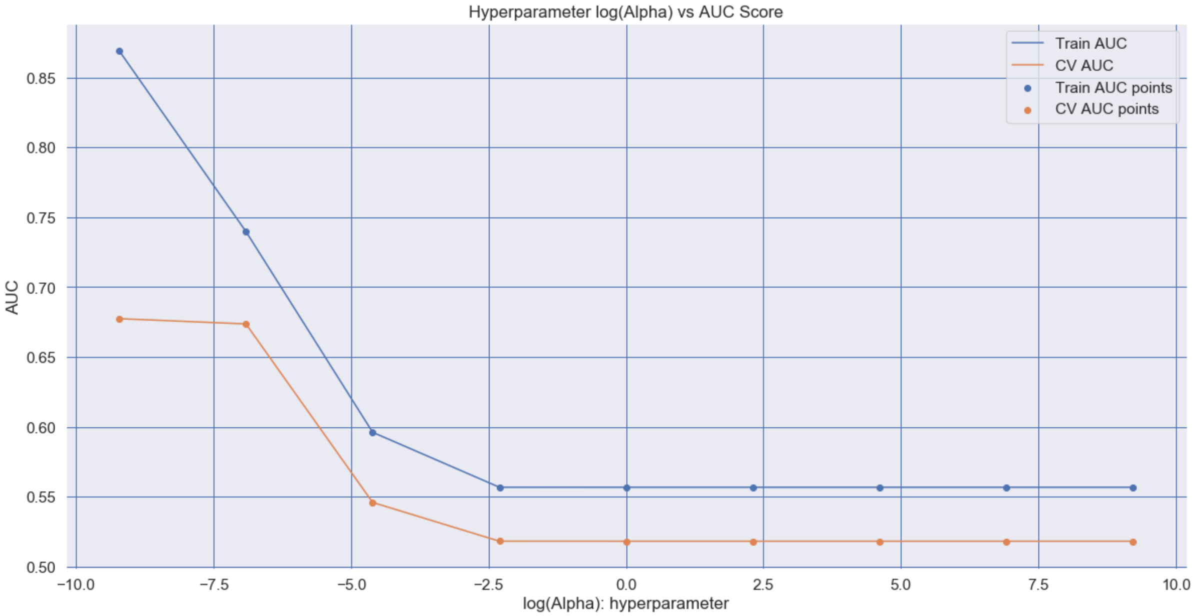
```
# Concatinating all the features for Set 2

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()
```

In [86]:
```
# SVM With L2 Penality
# L2 regulaization is default penality

# L1 Regularization - Lasso
# L2 Regularization - Ridge
train_auc = []
cv_auc = []
alpha = [10**-4, 10**-3,10**-2,10**-1,1,10,10**2,10**3,10**4]#alpha=1/C
for i in tqdm(alpha):
    base_estimator_svm_output_tfidf = SGDClassifier(loss="hinge", penalty='l2'
, class_weight='balanced', alpha=i)
    # Since SGDClassifier with Hinge loss doesn't have a predict_porba functio
n lets consider this
    # as a base estimator and have a CalibrateClassiferCV on top of this
    svm_output_tfidf = CalibratedClassifierCV(base_estimator_svm_output_tfidf,
cv=3)
    svm_output_tfidf.fit(X_tr, y_train)

    y_train_pred = svm_output_tfidf.predict_proba(X_tr)[:,1] # Returning the p
robablity score of greater class label
    y_cv_pred = svm_output_tfidf.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%|████████████████████████████████████████████████
████████| 9/9 [00:08<00:00,  1.05it/s]
```

In [87]:

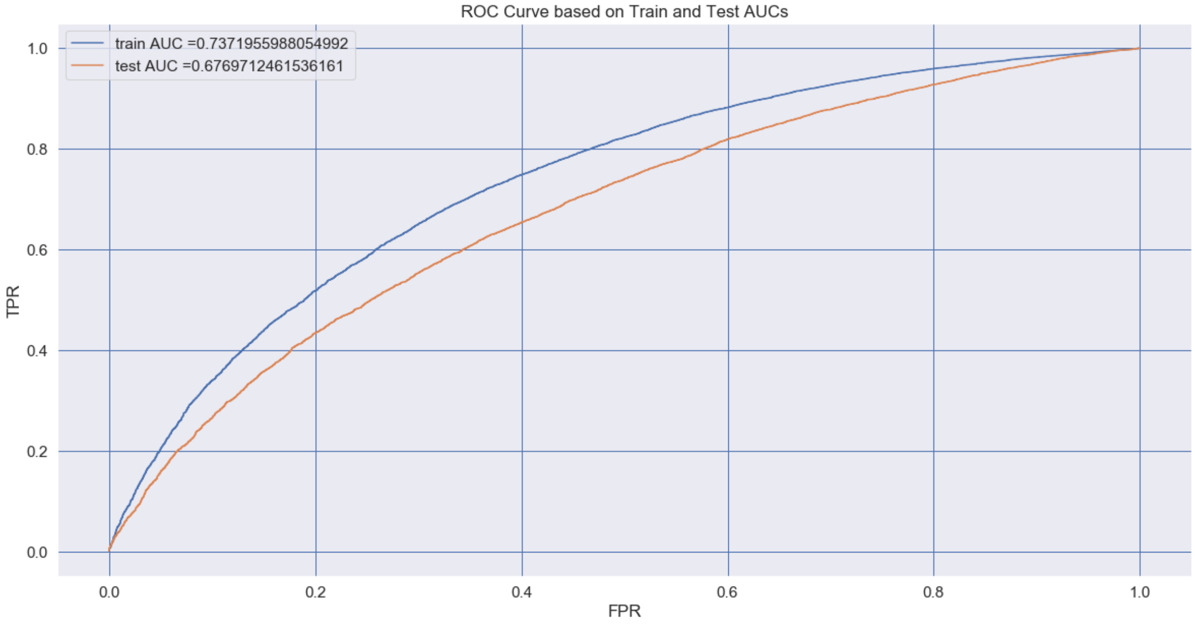
```
log_alphas = [log(alph) for alph in alpha]
plt.figure(figsize=(20,10))
plt.plot(log_alphas, train_auc, label='Train AUC')
plt.plot(log_alphas, cv_auc, label='CV AUC')

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(b=True, color="b")
plt.show()
```



In [88]:

```
# best alpha appears to be at second point
best_alpha = 10**-3

from sklearn.metrics import roc_curve, auc


base_estimator_svm_output_bow = SGDClassifier(loss="hinge", penalty='l2', clas
s_weight='balanced', alpha=best_alpha)
svm_output_bow = CalibratedClassifierCV(base_estimator_svm_output_bow, cv=3)
svm_output_bow.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 = svm_output_bow.predict_proba(X_tr)[:,1]  # returning probabilit
y estimates of positive class
y_test_pred = svm_output_bow.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 [89]:
```
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(b=True, color='b')
plt.show()
```



In [90]:
```
# 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)
```

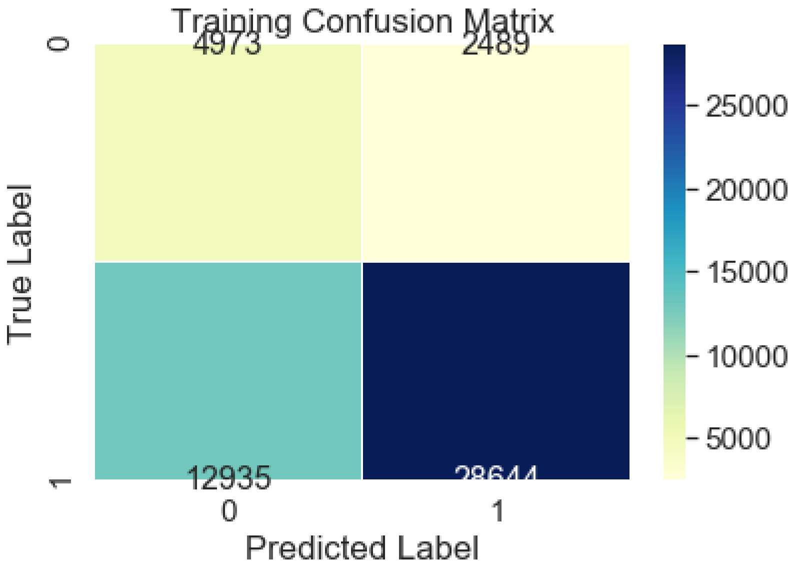
```
=============================================================================
=======================
the maximum value of tpr*(1-fpr) 0.45911643501303856 for threshold 0.835
Train confusion matrix
[[ 4973  2489]
 [12935 28644]]
Test confusion matrix
[[ 2953  2506]
 [ 8969 21624]]
```

In [91]:

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

Out[91]:

```
Text(0.5, 1, 'Training Confusion Matrix')
```

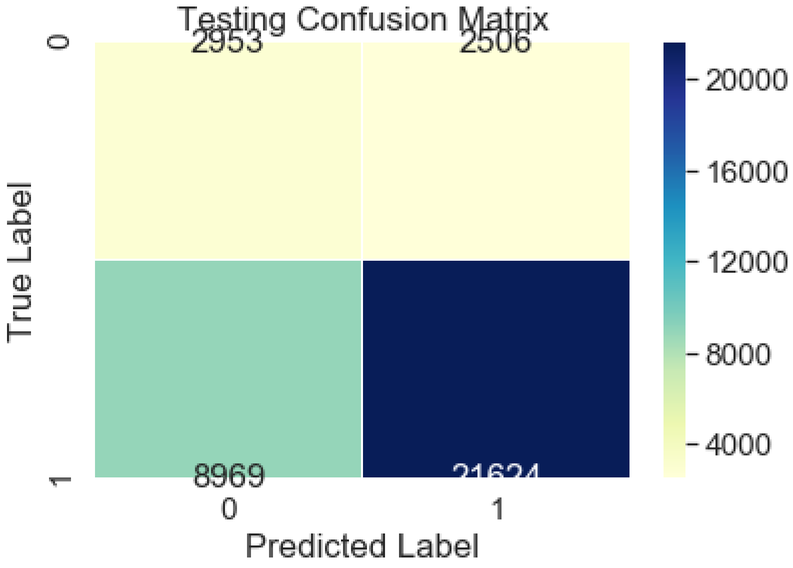


In [92]:

```
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[92]:

```
Text(0.5, 1, 'Testing Confusion Matrix')
```

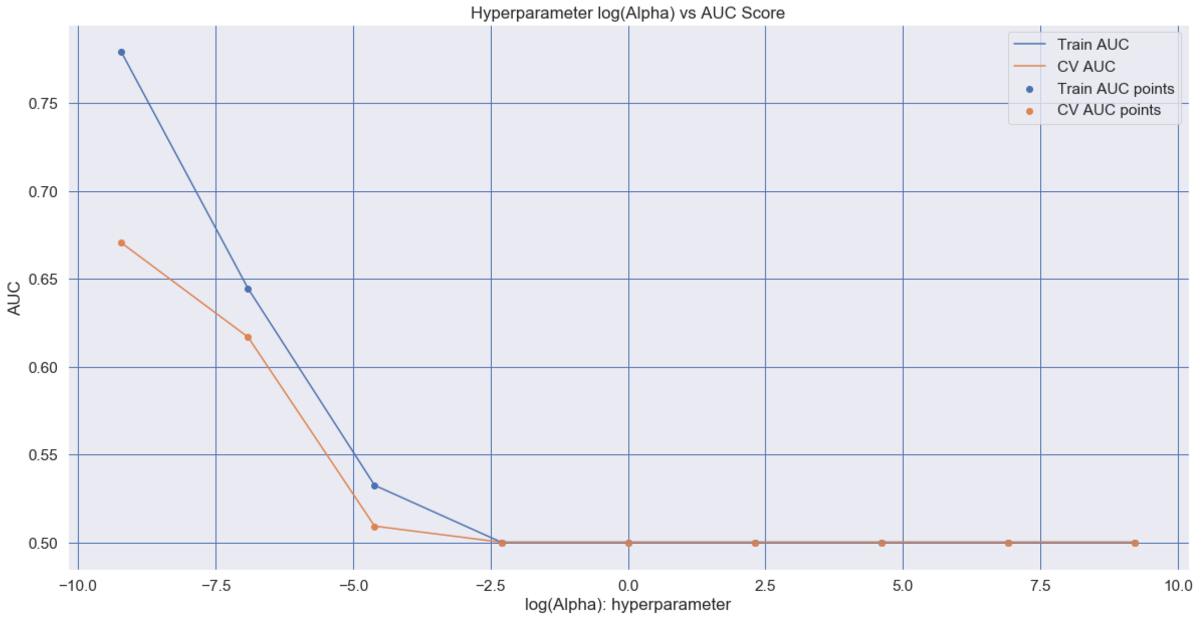


In [93]:

```
# SVM With L1 Penality
# L2 regulaization is default penality

# L1 Regularization - Lasso
# L2 Regularization - Ridge
train_auc = []
cv_auc = []
alpha = [10**-4, 10**-3,10**-2,10**-1,1,10,10**2,10**3,10**4]#alpha=1/C
for i in tqdm(alpha):
    base_estimator_svm_output_tfidf = SGDClassifier(loss="hinge", penalty='l1'
, class_weight='balanced', alpha=i)
    # Since SGDClassifier with Hinge loss doesn't have a predict_porba functio
n lets consider this
    # as a base estimator and have a CalibrateClassiferCV on top of this
    svm_output_tfidf = CalibratedClassifierCV(base_estimator_svm_output_tfidf,
cv=3)
    svm_output_tfidf.fit(X_tr, y_train)

    y_train_pred = svm_output_tfidf.predict_proba(X_tr)[:,1] # Returning the p
robablity score of greater class label
    y_cv_pred = svm_output_tfidf.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%|████████████████████████████████████████████████████████████████████████
████████████| 9/9 [00:13<00:00,  1.52s/it]
```

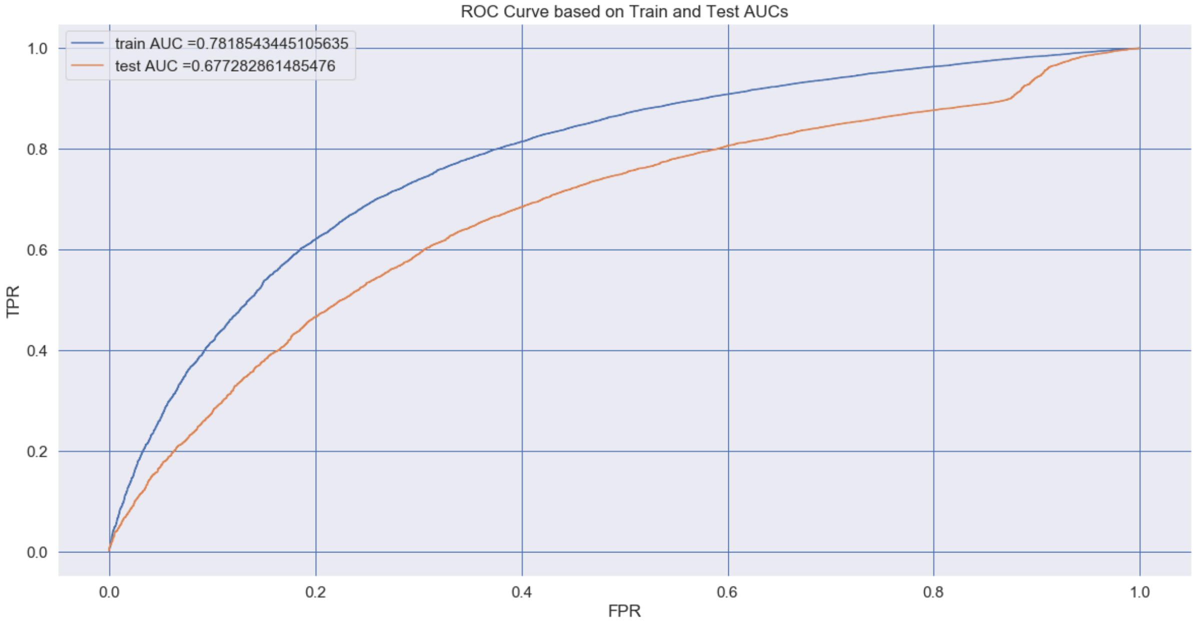
In [94]:
```
log_alphas = [log(alph) for alph in alpha]
plt.figure(figsize=(20,10))
plt.plot(log_alphas, train_auc, label='Train AUC')
plt.plot(log_alphas, cv_auc, label='CV AUC')

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(b=True, color="b")
plt.show()
```



In [95]:
```
# best alpha appears to be at first point
best_alpha = 10**-4

from sklearn.metrics import roc_curve, auc


base_estimator_svm_output_tfidf = SGDClassifier(loss="hinge", penalty='l1', cl
ass_weight='balanced', alpha=best_alpha)
svm_output_tfidf = CalibratedClassifierCV(base_estimator_svm_output_tfidf, cv=
3)
svm_output_tfidf.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 = svm_output_tfidf.predict_proba(X_tr)[:,1]  # returning probabil
ity estimates of positive class
y_test_pred = svm_output_tfidf.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 [96]:

```
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(b=True, color='b')
plt.show()
```



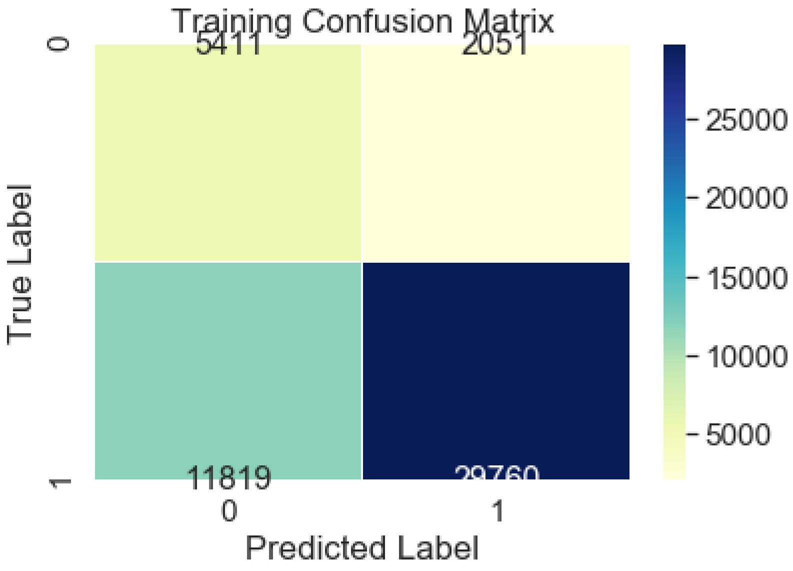
In [97]:

```
# 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) 0.5190165135587866 for threshold 0.831
Train confusion matrix
[[ 5411  2051]
 [11819 29760]]
Test confusion matrix
[[ 3327  2132]
 [ 9877 20716]]
```

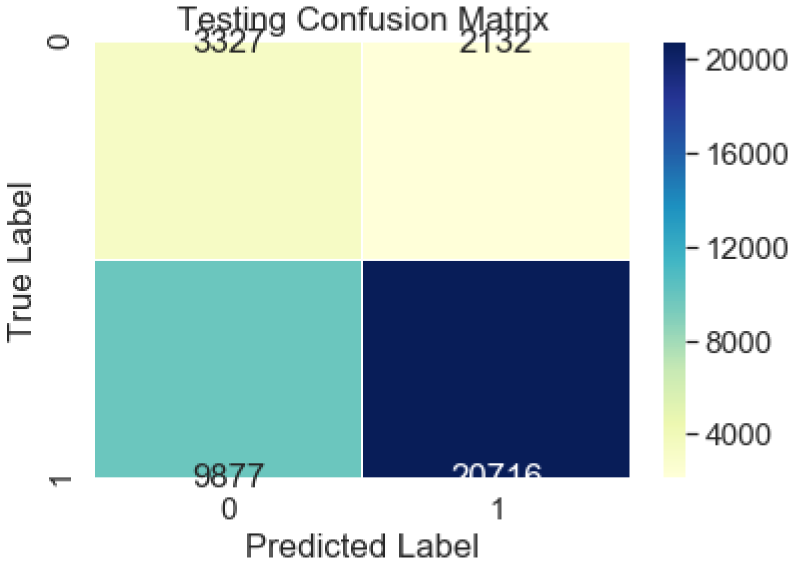
In [98]:
```
sns.set(font_scale=1.4)
sns.heatmap(Train_CM,annot=True,cbar=True,fmt="g", annot_kws = {"size":16},linewidths=.5,cmap="YlGnBu")
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Training Confusion Matrix')
```

Out[98]:
```
Text(0.5, 1, 'Training Confusion Matrix')
```



In [99]:
```
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[99]:
```
Text(0.5, 1, 'Testing Confusion Matrix')
```



Set - 3 AvgW2V

In [100]:

```
# Please write all the code with proper documentation
# 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 [101]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored i
n this list
for sentence in tqdm(X_train['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train.append(vector)

print(len(avg_w2v_vectors_train))
print(len(avg_w2v_vectors_train[0]))
print(avg_w2v_vectors_train[0])
```

```
100%|██████████████████████████████████████████████████████
██| 49041/49041 [00:10<00:00, 4524.49it/s]
```

```
49041
300
[ 3.31194164e-02  4.15624208e-02  3.31243881e-02 -1.00119567e-01
 -2.18907761e-02 -2.12810553e-02 -2.94911717e+00  1.42775689e-01
  4.78199365e-02  4.05099195e-02 -3.42107050e-02  1.35214070e-02
  1.63249553e-02 -1.27945182e-01 -7.81456547e-02 -1.86213270e-02
  3.05201220e-02 -1.52514214e-03  5.48681811e-02  1.35741562e-02
  3.10862913e-02 -2.54518453e-02 -3.36852836e-02  3.19574566e-02
 -4.59677286e-02 -5.57952323e-02  1.22484764e-01 -9.18642143e-02
 -1.91273605e-02 -7.72708591e-02 -2.43513601e-01 -6.72001319e-02
  2.97778950e-02  2.22960226e-02 -2.84918553e-02 -2.53407730e-02
 -7.04865767e-02 -6.04069277e-02 -4.43727125e-02 -4.85878717e-02
 -3.72745805e-02  3.42568289e-02  6.57715484e-02 -1.55513331e-01
  2.77018439e-03 -6.53233897e-02  7.07324214e-02 -1.22143762e-02
  3.44679262e-02 -6.00795397e-02 -2.02775931e-02 -3.02068931e-02
  3.04438333e-02 -6.52171780e-02  6.00671509e-03 -4.80679459e-02
  3.30483006e-02  3.82027535e-02 -1.05209552e-01  9.71785519e-02
 -3.36969547e-02  3.94021284e-03  4.65889261e-02 -5.63850893e-02
  5.05643648e-03  5.94684800e-02  7.54695572e-02 -4.61625075e-02
  1.43019266e-01 -1.35941081e-01 -1.11665052e-01 -1.63107855e-02
  7.01472245e-03 -7.46082786e-02 -4.81776818e-02 -1.37522255e-01
  2.11571491e-03 -1.63833321e-02  3.38290063e-03 -8.45297894e-02
  7.12144755e-02 -3.39087059e-01 -4.86887098e-02 -1.87912918e-02
 -1.27705162e-01 -3.65325994e-02  7.89948428e-02 -9.44885717e-02
  1.13488504e-01 -2.71230667e-02  6.19885192e-02 -2.96410761e-02
 -3.91274182e-02  5.73021965e-02  3.19749245e-02 -1.94461822e-01
 -2.18947371e+00  5.65142138e-03  9.90687855e-02  9.29094730e-02
 -9.12813744e-02 -3.48216440e-03  1.36674306e-01 -3.58484931e-02
  1.15070318e-01  5.04221824e-03  1.64579616e-02 -9.29945566e-02
  2.51858239e-03  5.35763629e-02 -5.83847752e-02  2.08168206e-02
  1.84579459e-02  1.56901560e-01  4.14426755e-02  4.13761522e-02
 -2.45523275e-01  4.02592421e-02  6.93834220e-02  5.20830107e-02
 -6.83235375e-03  1.93396164e-03  1.98540637e-02 -1.31080127e-01
  2.97262038e-02 -2.36789698e-02  3.19985478e-02  2.98730818e-03
 -2.58431053e-02  4.29845133e-02  6.50622723e-02  3.35048443e-02
 -5.91662642e-02 -5.25893893e-02  4.07645327e-02 -1.13822636e-01
  8.63675308e-02  1.66102057e-02  5.93189358e-02  2.58556995e-01
  3.50053088e-02 -1.75673000e-02  4.02986905e-02 -3.39627327e-02
 -3.22676016e-02  4.50188491e-02  4.09074679e-02 -1.44966535e-02
  1.74601874e-01 -4.14289667e-02  4.13253708e-02 -7.90463145e-03
  8.58159525e-02  4.26650314e-04 -3.28450126e-02  1.01176900e-02
  8.93832088e-02 -4.73176509e-02 -2.71016264e-02 -3.28690642e-02
 -1.39711824e-03  3.45121428e-02 -5.13091574e-02 -1.21372909e-01
 -5.06822176e-02 -2.55440270e-02 -4.57631188e-02  4.50774082e-02
  1.26774995e-01 -9.61615715e-02 -5.66730308e-02 -2.18645904e-02
 -2.52153522e-02 -6.67214610e-02 -3.18819314e-02 -9.59020138e-03
 -5.96784453e-02  6.82431447e-02 -8.29672744e-02 -6.59180566e-03
 -3.27008379e-02  2.21443903e-01  2.52878107e-02  3.74037019e-02
 -4.75133522e-02 -1.09281705e-01  2.60391440e-02 -4.36249824e-02
  8.55764182e-02  5.42434660e-02 -1.67539382e-02 -2.10280119e-02
 -1.01549346e-02  1.24213152e-02  1.17912176e-02 -6.10536709e-02
 -5.64677931e-02  8.59116855e-02  8.76696164e-03  7.72359856e-02
  1.20231601e-01 -1.64563522e-03 -2.68492560e-02  9.80352568e-02
 -1.24902743e-01  3.45170893e-02  6.27914190e-02 -9.81237981e-02
  1.39784616e-01 -1.77465723e-03 -1.08336843e-02 -4.62118489e-02
 -1.89529423e-02 -1.39162682e-01 -8.18494773e-02  3.13511403e-02
  2.56412409e-02 -8.89373780e-02 -5.32184288e-02 -2.73715698e-02
 -1.55847402e-01 -5.76796654e-02 -8.86369465e-02 -1.04026158e-01
 -1.86794743e+00  8.32692220e-02  3.16609623e-03  8.20727673e-04
 -2.16381730e-02 -8.07547799e-02  5.78709346e-02 -3.07452805e-02
 -8.35207107e-03 -6.14282780e-02 -6.45332038e-02  5.78607575e-02
  3.38294077e-02 -5.51979679e-02  1.70291499e-02  1.20015954e-01
 -1.57890189e-02  6.41716987e-02 -2.48818256e-01 -1.56155755e-02
 -3.04856950e-02  3.86501824e-02 -4.61744836e-02 -1.07679911e-01
 -2.61845817e-02 -5.32940768e-02 -1.53041962e-02  8.37963780e-02
  2.25559623e-02 -4.39660811e-02  9.47790354e-02  3.73514340e-03
  5.09105327e-02 -8.57238491e-03  1.08295736e-01 -1.04716472e-01
  1.36341390e-02  1.51617906e-02 -4.97316792e-02  6.15407585e-02
  4.87161428e-02 -9.32340282e-02 -2.93586629e-02 -7.28144391e-02
  2.28633730e-02  2.67138081e-02 -4.94147819e-02 -4.50359792e-02
 -6.22325654e-02  8.24484472e-02 -5.45907044e-03  8.93824252e-02
  8.40491943e-02 -5.68155912e-03 -3.09858412e-02  5.77578484e-02
  1.21763533e-01 -6.69676566e-02  2.80779528e-02  9.43715811e-02
  2.36647044e-03  9.13116069e-02  6.32097755e-02  2.18482697e-02
  4.25906000e-02  8.38058239e-03  2.56785302e-02 -7.29379906e-02
 -7.91222126e-02 -8.31263556e-02  7.83112748e-02  4.35933962e-02
 -3.16068994e-02  1.40417846e-01  1.39051219e-01  3.82918296e-02]
```

In [102]:

```
avg_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in t
his list
for sentence in tqdm(X_cv['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_cv.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████████████
█| 24155/24155 [00:05<00:00, 4344.33it/s]
```

In [103]:

```
avg_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in
this list
for sentence in tqdm(X_test['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████████████
█| 36052/36052 [00:08<00:00, 4101.25it/s]
```

In [104]:

```
# avg w2v for project_titles
avg_w2v_vectors_pj_title_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_pj_title_train.append(vector)

print(len(avg_w2v_vectors_pj_title_train))
print(len(avg_w2v_vectors_pj_title_train[0]))
print(avg_w2v_vectors_pj_title_train[0])
```

```
100%|██████████████████████████████████████████████████████████████████████████
█| 49041/49041 [00:00<00:00, 79824.86it/s]
```

```
49041
300
[ 1.47335000e-02  2.40459250e-01  1.26473750e-01  4.01700000e-02
  1.03930000e-01 -1.80428000e-01 -2.41657500e+00 -5.22285000e-01
 -1.10250000e-01 -4.03125000e-02  2.12650000e-01 -5.47485000e-02
  8.09607500e-02 -2.57711250e-01 -1.54774000e-01 -2.46225000e-02
  1.72018250e-01 -3.55000000e-03  1.10305000e-01 -8.21315000e-02
  1.01092500e-01  1.15109750e-01 -3.66892500e-02  3.23512500e-01
  1.44801750e-01 -1.64654800e-01  3.89472500e-01  2.19007500e-01
 -3.35650000e-02 -9.28422500e-02 -2.57100250e-01 -3.05049900e-01
 -2.54185000e-01 -1.25405625e-01 -2.85500000e-02  2.25362500e-01
 -1.45560250e-01  1.80248250e-01 -4.26897500e-01 -1.64338500e-01
 -1.03464650e-01 -1.32022500e-01  2.09199000e-01  2.88287500e-02
 -7.82105000e-02 -1.57390000e-01  9.63125000e-03 -5.57825000e-03
  2.03895000e-02 -7.38483750e-02  1.93367500e-01 -2.30542500e-02
  1.85250000e-01  2.83281500e-01  2.25269750e-01 -6.58400000e-02
  3.66870000e-02  3.97412500e-02 -1.36905000e-01  2.75802500e-01
 -3.54385000e-01  2.00625000e-01  1.97651500e-01 -1.71455500e-01
  6.76800000e-02  4.36478750e-01  2.55732000e-01 -2.63727250e-01
  6.41682500e-02 -1.49315000e-01 -4.32365000e-01 -1.67147500e-01
  1.44128500e-01  3.02200000e-02  2.12297500e-01 -3.36752500e-01
  8.13585000e-02 -5.71187500e-02 -1.73651500e-01 -3.22680000e-02
  2.66180750e-01 -3.08235000e-01 -2.63402500e-01  3.38357000e-01
 -1.87032500e-01  5.08667500e-02  3.01036000e-01 -4.79025000e-02
  2.54137750e-01 -1.20280000e-02 -1.74962500e-02 -1.20887500e-01
  1.23494000e-01  7.41015500e-02 -2.53655000e-02 -2.13410000e-01
 -2.41542500e+00 -4.49875000e-02  1.02022500e-01 -4.05227500e-02
 -1.89182250e-01 -8.78070000e-02  2.52952500e-02 -1.17757500e-01
  2.65185000e-01  1.51889750e-01 -9.40300000e-02 -3.41575000e-02
  1.65407500e-01  1.79537500e-02 -4.34859750e-01  1.99501000e-01
  2.03850000e-03  4.96900000e-01  2.30492500e-01 -6.06072500e-02
 -2.11482500e-01  2.60470250e-01  2.76070250e-01 -2.32423250e-02
 -1.68750175e-01  2.21042500e-01 -1.78940000e-01 -1.23238500e-01
 -6.31662500e-02  1.83395000e-01  2.24787500e-01  9.26465000e-02
  3.15652750e-01  3.82592500e-02  2.25925000e-02  2.28117000e-01
  1.46477500e-01  2.09340000e-01  3.44722750e-01  4.74600000e-02
  4.67667500e-02  1.94545250e-01 -8.90800000e-02  6.77262500e-01
 -2.30959250e-01 -7.77250000e-03 -2.05894750e-01  9.51975000e-02
  1.22500000e-04  2.19912500e-01 -7.41800000e-02  2.79072500e-02
 -1.22933000e-01  3.05350000e-02  7.48752500e-02  6.22680000e-02
  1.74877500e-01  1.51759750e-01 -2.27850000e-01 -1.42015000e-01
 -5.77710000e-02 -9.11875000e-02 -1.09800000e-01 -2.82337500e-01
 -3.97497500e-01  1.23285000e-01 -2.46589250e-01 -7.23222500e-02
 -3.82484000e-01 -1.53814250e-01  1.74249500e-01  2.33657000e-01
  2.01119500e-01 -1.18375500e-01  2.31988750e-01  1.90979275e-01
 -5.15125000e-02 -1.27795875e-01 -1.25434250e-01  7.31544000e-02
  7.95870000e-02  7.01977500e-02 -2.32355500e-01 -2.62362500e-02
 -3.36372500e-01  3.82415200e-01  8.30811000e-02  1.55697750e-01
 -2.08777500e-02 -6.18175000e-02 -1.90452500e-01 -2.66805000e-02
  2.28763750e-01 -2.18900000e-02  1.74781750e-01  1.58682500e-01
 -1.23325000e-02 -1.00427000e-01 -1.61797500e-01 -1.77280000e-01
 -2.41963250e-01  2.90457500e-01 -1.14069000e-01  2.06314000e-01
  3.09740000e-02 -1.73542500e-02 -1.34435000e-01  2.84464750e-01
 -2.24398000e-01 -1.60363250e-01  2.91040500e-01 -2.22521000e-01
 -3.40305000e-02 -3.87300000e-03 -9.30135000e-02  3.61477500e-02
 -1.03009500e-01 -7.75967500e-02  5.33767500e-02  5.49172250e-02
 -7.70654750e-02  6.62785000e-02 -3.69500000e-03  2.12587500e-01
 -1.60138250e-01  1.96535250e-01 -3.95415000e-02 -4.36595000e-01
 -2.59712500e+00 -5.64600000e-02 -1.15487750e-01  1.52062500e-01
  2.88521750e-02 -1.57232500e-01 -8.39975000e-02 -2.37791000e-01
  3.65020000e-02 -1.27460000e-01 -1.35877750e-01  2.05560000e-01
  6.98650000e-02  9.40900000e-03  4.10052500e-01  1.30801750e-01
 -2.08425000e-02  1.69155750e-01 -1.70547500e-01  3.47317500e-01
 -1.86952750e-01  3.92572500e-02  1.34214500e-01 -2.65344000e-01
 -9.84047500e-02 -1.58157500e-01 -6.63790000e-02  2.24256000e-01
  3.66520000e-01 -1.14130750e-01  1.62605500e-01 -1.19968000e-01
  1.62892250e-01 -4.27305000e-02 -5.32640250e-02 -3.93467500e-02
 -6.44917500e-02 -4.15675000e-03 -3.31775000e-02  1.56647500e-01
  1.34764750e-01  1.63930000e-02  3.30395000e-01  7.15417500e-02
 -5.13580000e-01 -1.02673000e-01  7.40517500e-02  1.84000000e-03
  1.11500000e-02  1.78500000e-01  2.44457500e-01  2.87249000e-01
  1.33258750e-01 -1.01585000e-02  9.69872500e-02  2.24903750e-01
  8.39730000e-02 -4.50850000e-02 -1.45256875e-01  1.50120000e-02
 -1.82596250e-01  2.05799250e-01 -9.97257500e-02  1.57202000e-01
 -1.50162650e-01 -2.64110000e-01  1.26671500e-01  6.37849000e-02
  3.89175000e-02 -1.87703750e-01 -6.92199500e-02 -5.37330000e-02
 -6.89025000e-02 -3.70555000e-02  2.26332500e-01  1.54360000e-01]
```

In [105]:

```python
avg_w2v_vectors_pj_title_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_pj_title_cv.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████
█| 24155/24155 [00:00<00:00, 80197.68it/s]
```

In [106]:

```python
avg_w2v_vectors_pj_title_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_pj_title_test.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████
█| 36052/36052 [00:00<00:00, 78926.61it/s]
```

In [107]:

```
X_tr = hstack((avg_w2v_vectors_train, X_train_state_ohe, X_train_teacher_ohe,
               X_train_grade_ohe, X_train_category_ohe,
               X_train_subcategory_ohe, X_train_price_norm,
               X_train_teach_prev_norm, avg_w2v_vectors_pj_title_train)).tocsr
()

X_cr = hstack((avg_w2v_vectors_cv, 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, avg_w2v_vectors_pj_title_cv)).tocsr()

X_te = hstack((avg_w2v_vectors_test, 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, avg_w2v_vectors_pj_title_test)).tocsr()

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

```
Final Data matrix
(49041, 701) (49041,)
(24155, 701) (24155,)
(36052, 701) (36052,)
==============================================================================
======================
```

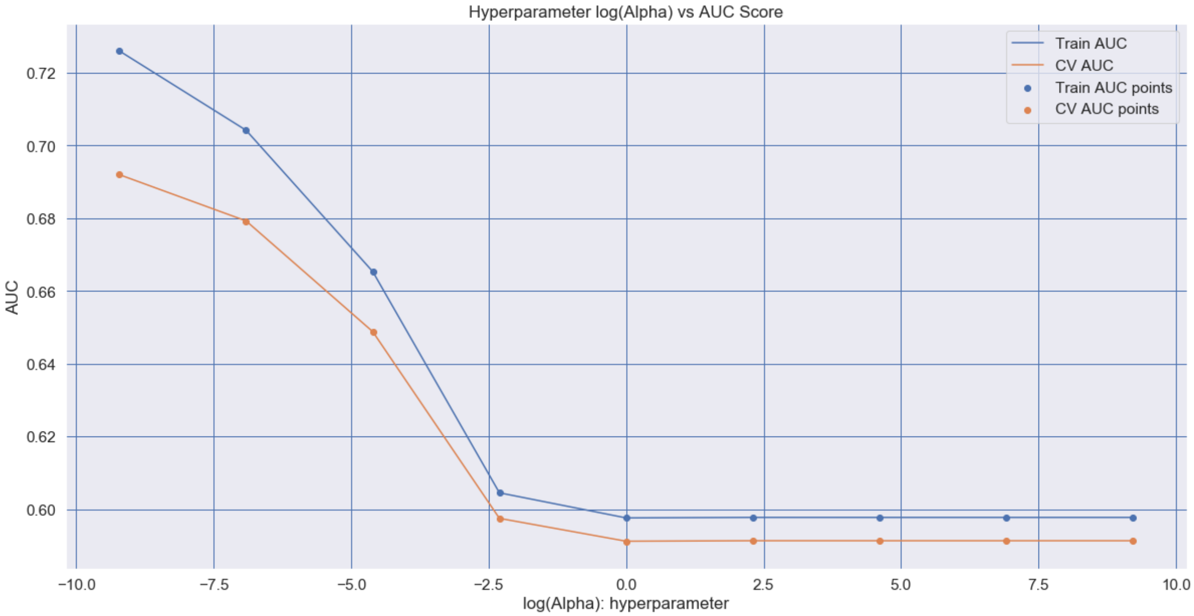
In [108]:

```
# SVM With L2 Penality
# L2 regulaization is default penality

# L1 Regularization - Lasso
# L2 Regularization - Ridge
train_auc = []
cv_auc = []
alpha = [10**-4, 10**-3,10**-2,10**-1,1,10,10**2,10**3,10**4]#alpha=1/C
for i in tqdm(alpha):
    base_estimator_svm_output_tfidf = SGDClassifier(loss="hinge", penalty='l2', class_weight='balanced', alpha=i)
    # Since SGDClassifier with Hinge loss doesn't have a predict_porba function lets consider this
    # as a base estimator and have a CalibrateClassiferCV on top of this
    svm_output_tfidf = CalibratedClassifierCV(base_estimator_svm_output_tfidf, cv=3)
    svm_output_tfidf.fit(X_tr, y_train)

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

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates 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%|██████████████████████████████████████████████████████████████
██████████| 9/9 [00:32<00:00,  3.63s/it]
```

In [109]:

```
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(b=True, color="b")
plt.show()
```

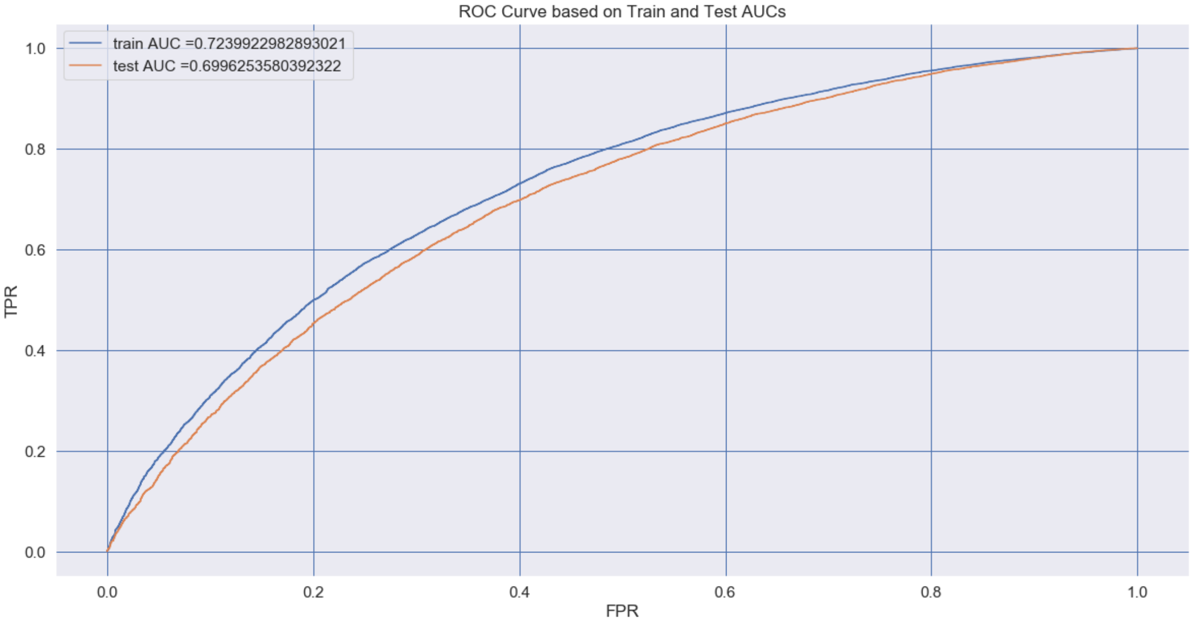


In [110]:

```
# best alpha appears to be at first point
best_alpha = 10**-4

from sklearn.metrics import roc_curve, auc


base_estimator_svm_output_bow = SGDClassifier(loss="hinge", penalty='l2', clas
s_weight='balanced', alpha=best_alpha)
svm_output_bow = CalibratedClassifierCV(base_estimator_svm_output_bow, cv=3)
svm_output_bow.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 = svm_output_bow.predict_proba(X_tr)[:,1]  # returning probabilit
y estimates of positive class
y_test_pred = svm_output_bow.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 [111]:

```
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(b=True, color='b')
plt.show()
```



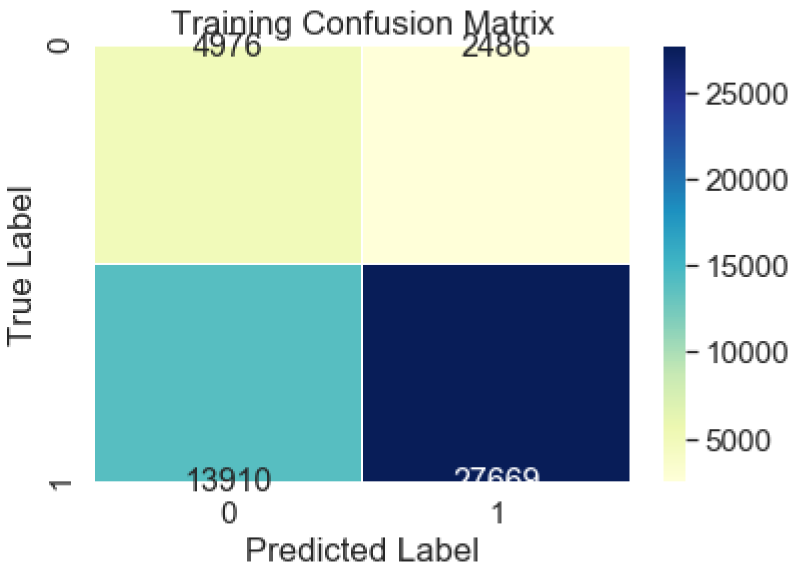
In [112]:

```
# 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) 0.44375631888324446 for threshold 0.839
Train confusion matrix
[[ 4976  2486]
 [13910 27669]]
Test confusion matrix
[[ 3478  1981]
 [10366 20227]]
```

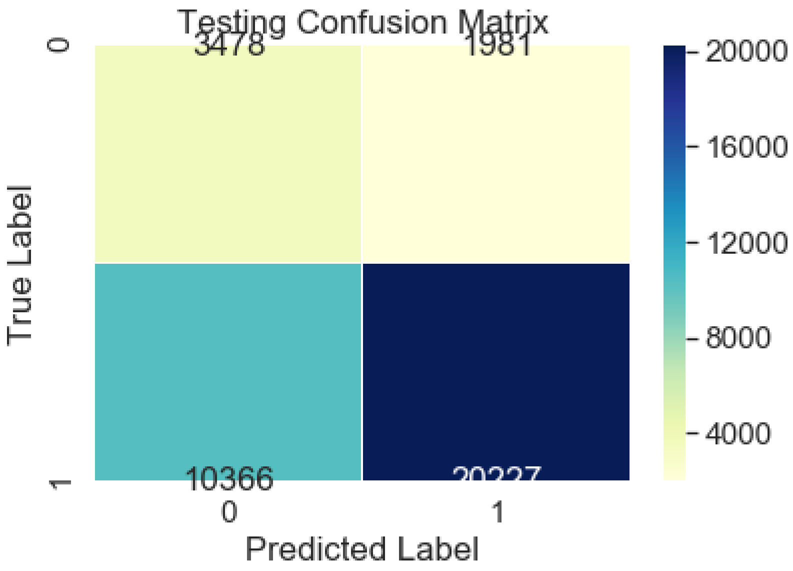
In [113]:
```
sns.set(font_scale=1.4)
sns.heatmap(Train_CM,annot=True,cbar=True,fmt="g", annot_kws = {"size":16},lin
ewidths=.5,cmap="YlGnBu")
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Training Confusion Matrix')
```

Out[113]:
```
Text(0.5, 1, 'Training Confusion Matrix')
```



In [114]:
```
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[114]:
```
Text(0.5, 1, 'Testing Confusion Matrix')
```



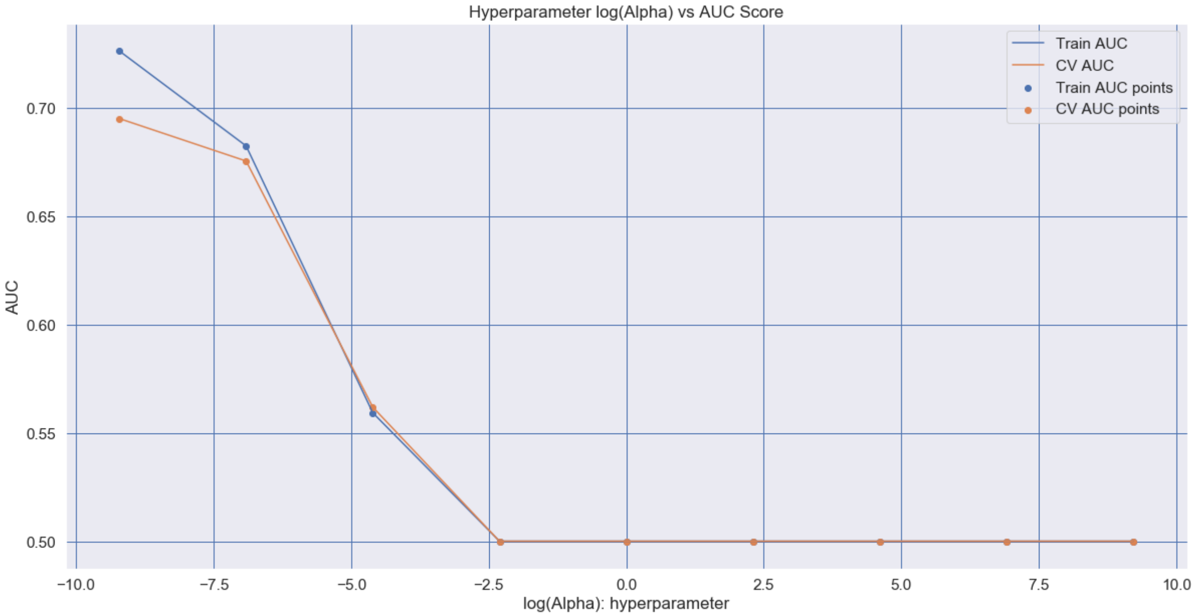
In [115]:

```
# SVM With L1 Penality
# L2 regulaization is default penality

# L1 Regularization - Lasso
# L2 Regularization - Ridge
train_auc = []
cv_auc = []
alpha = [10**-4, 10**-3,10**-2,10**-1,1,10,10**2,10**3,10**4]#alpha=1/C
for i in tqdm(alpha):
    base_estimator_svm_output_tfidf = SGDClassifier(loss="hinge", penalty='l1'
, class_weight='balanced', alpha=i)
    # Since SGDClassifier with Hinge loss doesn't have a predict_porba functio
n lets consider this
    # as a base estimator and have a CalibrateClassiferCV on top of this
    svm_output_tfidf = CalibratedClassifierCV(base_estimator_svm_output_tfidf,
cv=3)
    svm_output_tfidf.fit(X_tr, y_train)

    y_train_pred = svm_output_tfidf.predict_proba(X_tr)[:,1] # Returning the p
robablity score of greater class label
    y_cv_pred = svm_output_tfidf.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%|████████████████████████████████████████████████████████████████████████
████████████| 9/9 [01:21<00:00,  9.01s/it]
```

In [116]:

```
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(b=True, color="b")
plt.show()
```

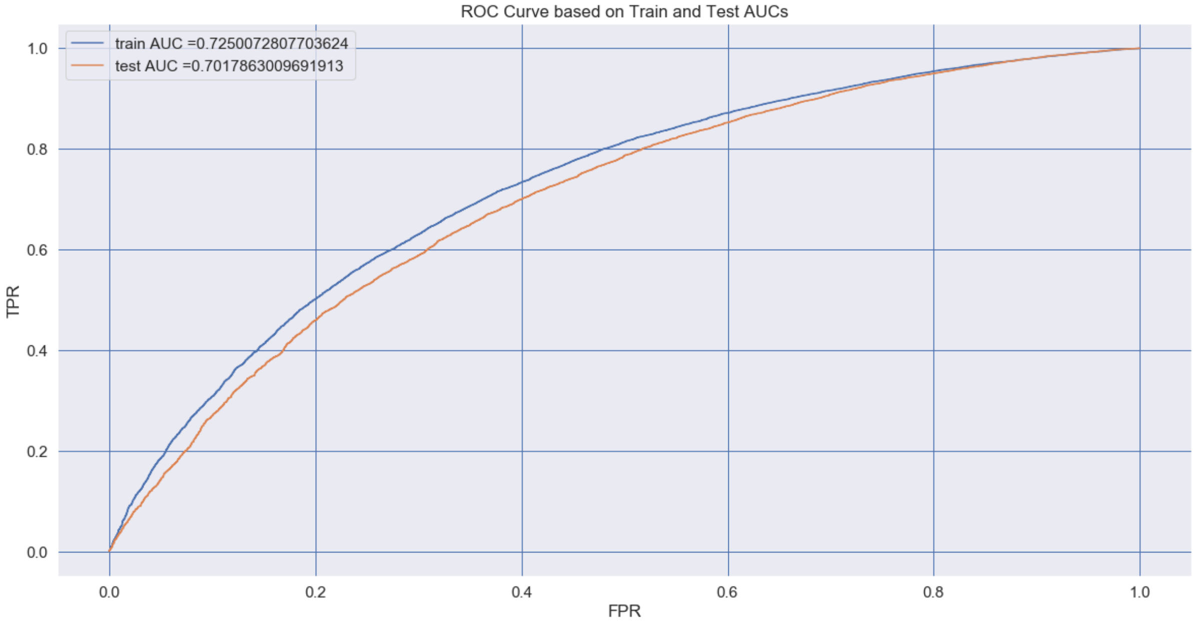


In [117]:

```
# best alpha appears to be at first point
best_alpha = 10**-4

from sklearn.metrics import roc_curve, auc


base_estimator_svm_output_tfidf = SGDClassifier(loss="hinge", penalty='l1', cl
ass_weight='balanced', alpha=best_alpha)
svm_output_tfidf = CalibratedClassifierCV(base_estimator_svm_output_tfidf, cv=
3)
svm_output_tfidf.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 = svm_output_tfidf.predict_proba(X_tr)[:,1]  # returning probabil
ity estimates of positive class
y_test_pred = svm_output_tfidf.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 [118]:

```
plt.figure(figsize=(20,10))
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve based on Train and Test AUCs")
plt.grid(b=True, color='b')
plt.show()
```



Set - 4 TFIDF W2V

In [119]:

```
# Please write all the code with proper documentation
# preprocessing project_title and essay with TFIDF W2V Vectorization
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['essay'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [120]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the 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_train.append(vector)

print(len(tfidf_w2v_vectors_train))
print(len(tfidf_w2v_vectors_train[0]))
```

```
100%|██████████████████████████████████████████████████████████| 49041/49041 [01:22<00:00, 593.65it/s]

49041
300
```

In [121]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the 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_cv.append(vector)

print(len(tfidf_w2v_vectors_cv))
print(len(tfidf_w2v_vectors_cv[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████
███| 24155/24155 [00:40<00:00, 601.45it/s]

24155
300
```

In [122]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored
 in this list
for sentence in tqdm(X_test['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/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_test.append(vector)

print(len(tfidf_w2v_vectors_test))
print(len(tfidf_w2v_vectors_test[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████
███| 36052/36052 [01:00<00:00, 599.27it/s]

36052
300
```

In [123]:

```
# preprocessing for Project_title with TFIDF Vectorization
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['project_title'])
# 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 [124]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_pj_title_train = []; # the avg-w2v for each sentence/review
 is stored in this list
for sentence in tqdm(X_train['project_title']): # 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_pj_title_train.append(vector)

print(len(tfidf_w2v_vectors_pj_title_train))
print(len(tfidf_w2v_vectors_pj_title_train[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
█| 49041/49041 [00:01<00:00, 38901.29it/s]

49041
300
```

In [125]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_pj_title_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the 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_pj_title_cv.append(vector)

print(len(tfidf_w2v_vectors_pj_title_cv))
print(len(tfidf_w2v_vectors_pj_title_cv[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
█| 24155/24155 [00:00<00:00, 39063.91it/s]

24155
300
```

In [126]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_pj_title_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the 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_pj_title_test.append(vector)

print(len(tfidf_w2v_vectors_pj_title_test))
print(len(tfidf_w2v_vectors_pj_title_test[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████
█| 36052/36052 [00:00<00:00, 38869.22it/s]

36052
300
```

In [127]:

```
# Concatinating all the features
X_tr = hstack((tfidf_w2v_vectors_train, X_train_state_ohe, X_train_teacher_ohe
,
               X_train_grade_ohe, X_train_category_ohe,
               X_train_subcategory_ohe, X_train_price_norm,
               X_train_teach_prev_norm, tfidf_w2v_vectors_pj_title_train)).toc
sr()

X_cr = hstack((tfidf_w2v_vectors_cv, 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, tfidf_w2v_vectors_pj_title_cv)).tocsr()

X_te = hstack((tfidf_w2v_vectors_test, 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, tfidf_w2v_vectors_pj_title_test)).tocsr
()

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

```
Final Data matrix
(49041, 701) (49041,)
(24155, 701) (24155,)
(36052, 701) (36052,)
=============================================================================
=======================
```

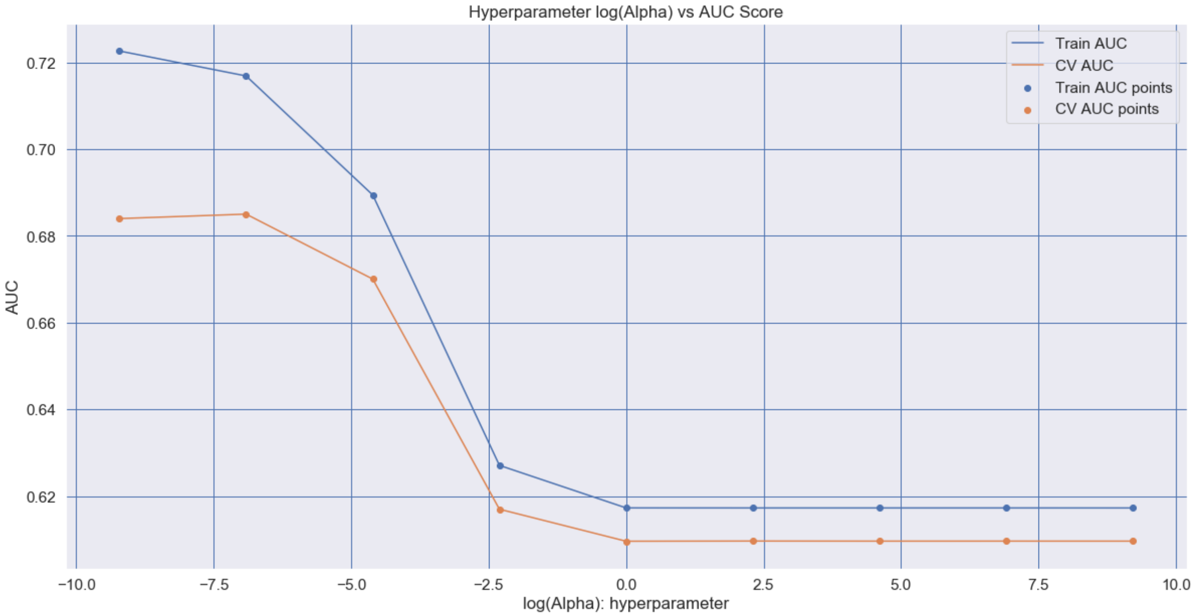
In [128]:

```
# SVM With L2 Penality
# L2 regulaization is default penality

# L1 Regularization - Lasso
# L2 Regularization - Ridge
train_auc = []
cv_auc = []
alpha = [10**-4, 10**-3,10**-2,10**-1,1,10,10**2,10**3,10**4]#alpha=1/C
for i in tqdm(alpha):
    base_estimator_svm_output_tfidf = SGDClassifier(loss="hinge", penalty='l2', class_weight='balanced', alpha=i)
    # Since SGDClassifier with Hinge loss doesn't have a predict_porba function lets consider this
    # as a base estimator and have a CalibrateClassiferCV on top of this
    svm_output_tfidf = CalibratedClassifierCV(base_estimator_svm_output_tfidf, cv=3)
    svm_output_tfidf.fit(X_tr, y_train)

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

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates 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%|████████████████████████████████████████████████████████████████████
███████████| 9/9 [00:32<00:00,  3.64s/it]
```

In [129]:

```
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(b=True, color="b")
plt.show()
```

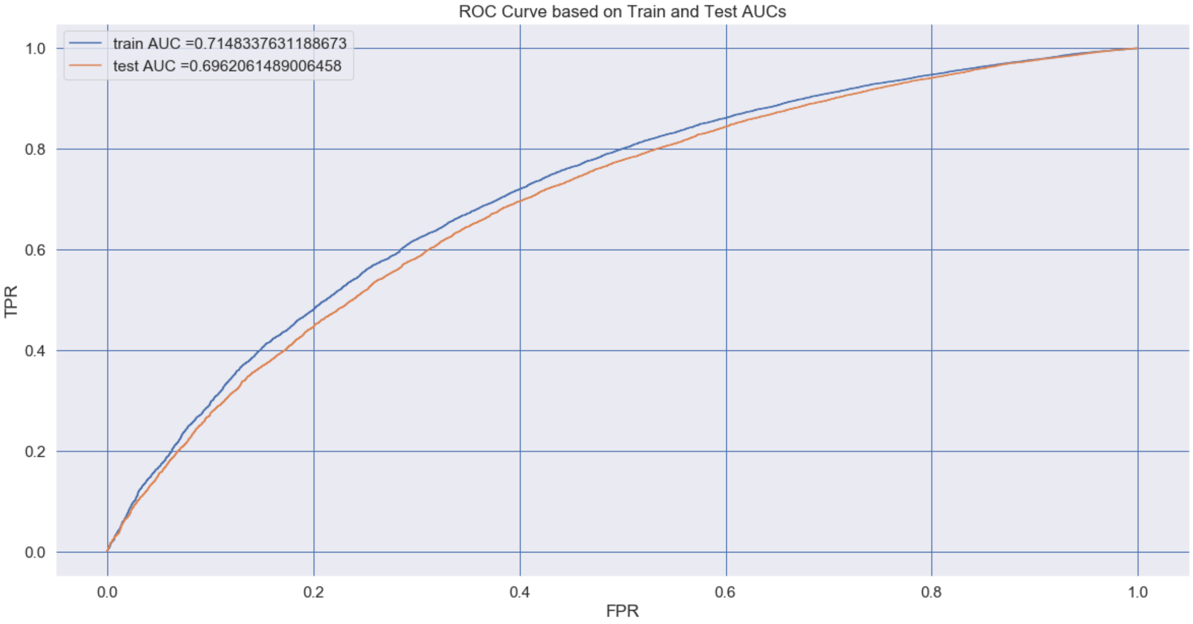


In [130]:

```
# best alpha appears to be at second point
best_alpha = 10**-3

from sklearn.metrics import roc_curve, auc


base_estimator_svm_output_bow = SGDClassifier(loss="hinge", penalty='l2', clas
s_weight='balanced', alpha=best_alpha)
svm_output_bow = CalibratedClassifierCV(base_estimator_svm_output_bow, cv=3)
svm_output_bow.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 = svm_output_bow.predict_proba(X_tr)[:,1]  # returning probabilit
y estimates of positive class
y_test_pred = svm_output_bow.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 [131]:

```
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(b=True, color='b')
plt.show()
```



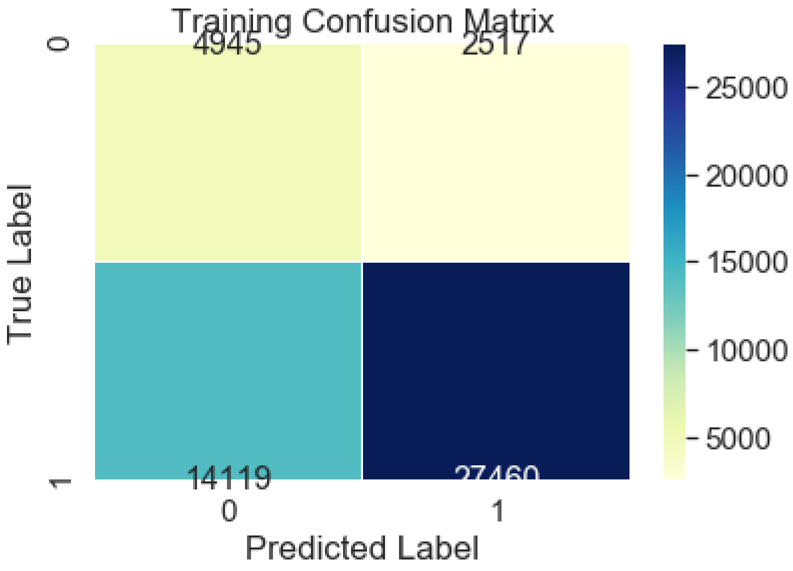
In [132]:

```
# 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) 0.43766069336552565 for threshold 0.838
Train confusion matrix
[[ 4945  2517]
 [14119 27460]]
Test confusion matrix
[[ 3499  1960]
 [10576 20017]]
```

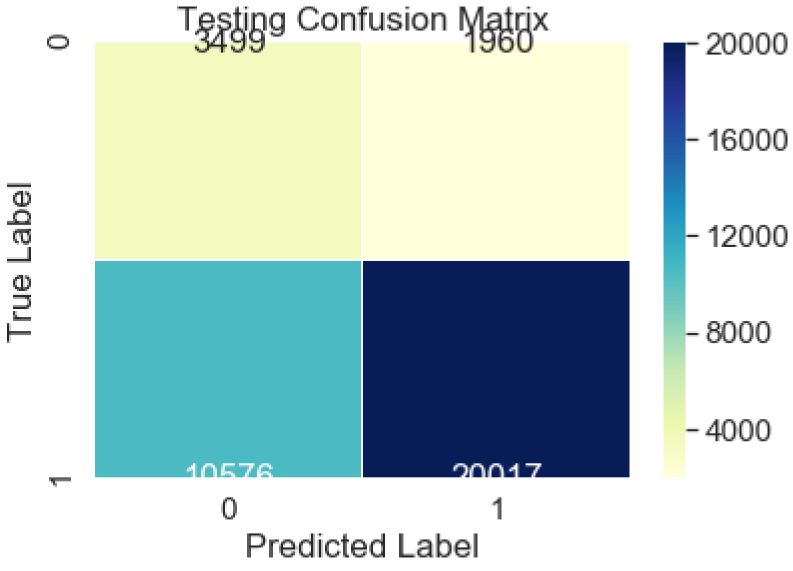
In [133]:
```
sns.set(font_scale=1.4)
sns.heatmap(Train_CM,annot=True,cbar=True,fmt="g", annot_kws = {"size":16},linewidths=.5,cmap="YlGnBu")
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Training Confusion Matrix')
```

Out[133]:
```
Text(0.5, 1, 'Training Confusion Matrix')
```



In [134]:
```
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[134]:
```
Text(0.5, 1, 'Testing Confusion Matrix')
```



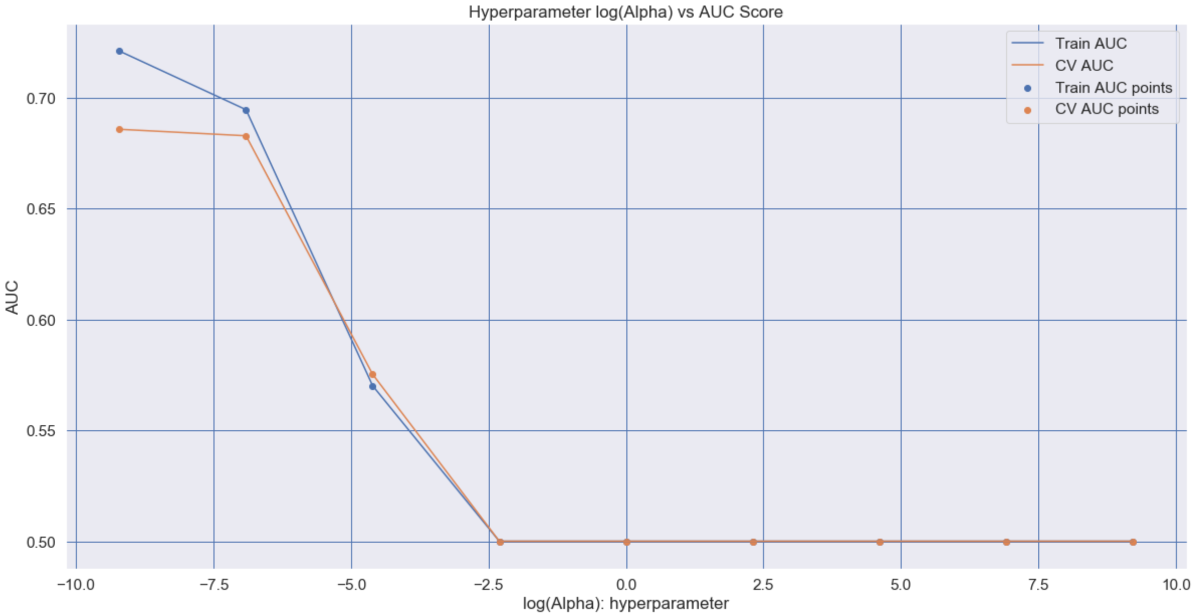
In [135]:

```
# SVM With L1 Penality
# L2 regulaization is default penality

# L1 Regularization - Lasso
# L2 Regularization - Ridge
train_auc = []
cv_auc = []
alpha = [10**-4, 10**-3,10**-2,10**-1,1,10,10**2,10**3,10**4]#alpha=1/C
for i in tqdm(alpha):
    base_estimator_svm_output_tfidf = SGDClassifier(loss="hinge", penalty='l1'
, class_weight='balanced', alpha=i)
    # Since SGDClassifier with Hinge loss doesn't have a predict_porba functio
n lets consider this
    # as a base estimator and have a CalibrateClassiferCV on top of this
    svm_output_tfidf = CalibratedClassifierCV(base_estimator_svm_output_tfidf,
cv=3)
    svm_output_tfidf.fit(X_tr, y_train)

    y_train_pred = svm_output_tfidf.predict_proba(X_tr)[:,1] # Returning the p
robablity score of greater class label
    y_cv_pred = svm_output_tfidf.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%|██████████████████████████████████████████████████████████████████████████
████████████| 9/9 [01:08<00:00,  7.57s/it]
```

In [136]:

```
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(b=True, color="b")
plt.show()
```

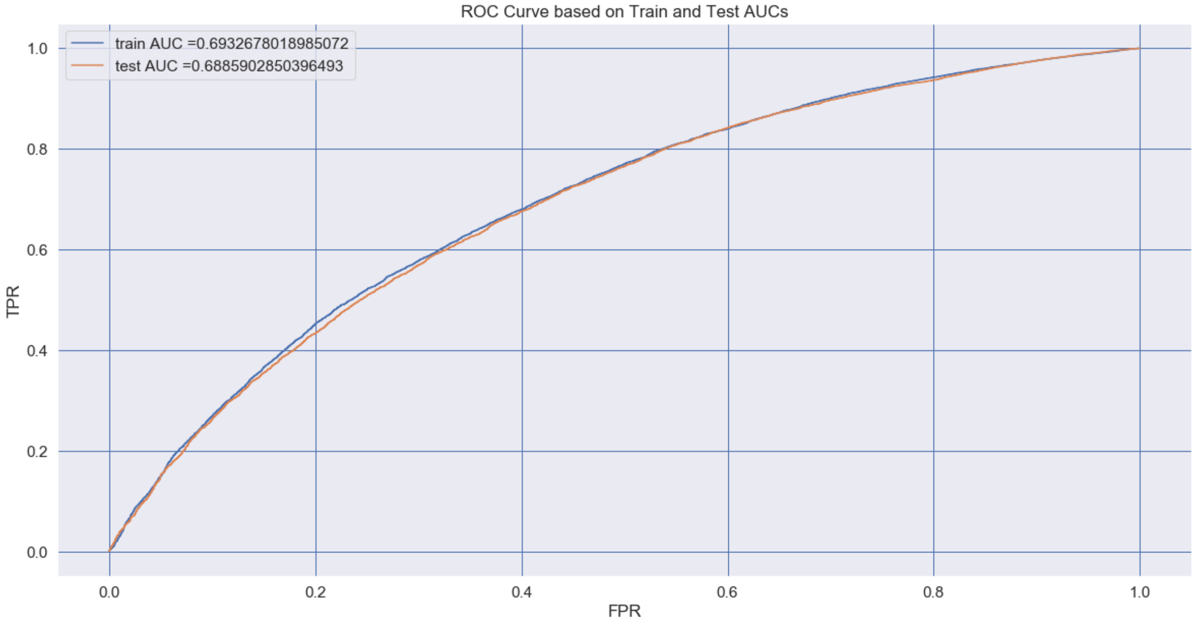


In [137]:

```
# best alpha appears to be at second point
best_alpha = 10**-3

from sklearn.metrics import roc_curve, auc


base_estimator_svm_output_tfidf = SGDClassifier(loss="hinge", penalty='l1', cl
ass_weight='balanced', alpha=best_alpha)
svm_output_tfidf = CalibratedClassifierCV(base_estimator_svm_output_tfidf, cv=
3)
svm_output_tfidf.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 = svm_output_tfidf.predict_proba(X_tr)[:,1]  # returning probabil
ity estimates of positive class
y_test_pred = svm_output_tfidf.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 [138]:

```
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(b=True, color='b')
plt.show()
```



In [139]:

```
# 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)
```

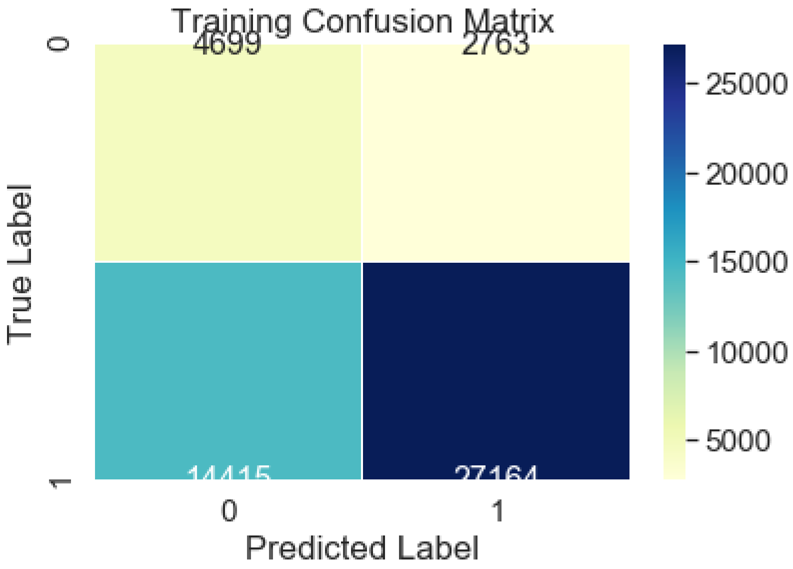
```
=============================================================================
=======================
the maximum value of tpr*(1-fpr) 0.41140529977941453 for threshold 0.837
Train confusion matrix
[[ 4699  2763]
 [14415 27164]]
Test confusion matrix
[[ 3431  2028]
 [10737 19856]]
```

In [140]:

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

Out[140]:

```
Text(0.5, 1, 'Training Confusion Matrix')
```

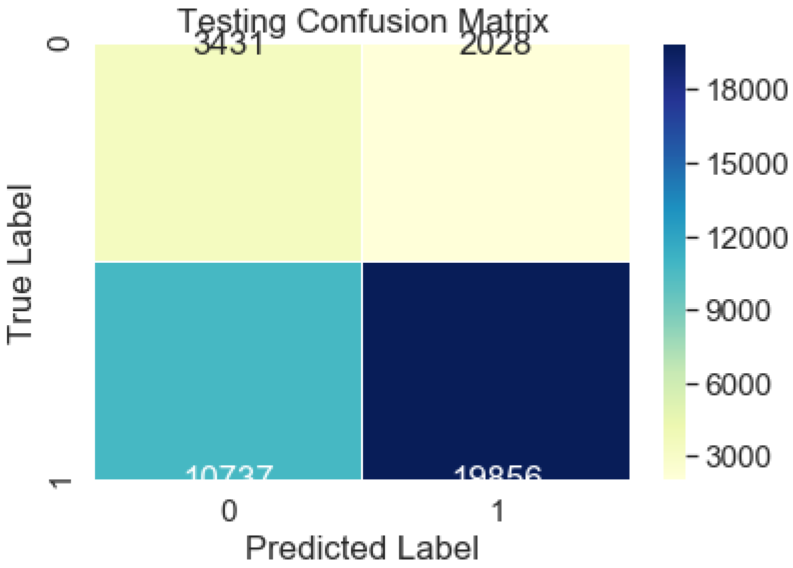


In [141]:

```
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[141]:

```
Text(0.5, 1, 'Testing Confusion Matrix')
```



2.5 Support Vector Machines with added Features `Set 5`

In [146]:
```
# Categorical data
print ("School State:", X_train_state_ohe.shape, X_cv_state_ohe.shape, X_test_state_ohe.shape)
print ("Clean Categories:", X_train_category_ohe.shape, X_cv_category_ohe.shape, X_test_category_ohe.shape)
print ("Clean Subcategories:", X_train_subcategory_ohe.shape, X_cv_subcategory_ohe.shape, X_test_subcategory_ohe.shape)
print ("Project Grade:", X_train_grade_ohe.shape, X_cv_grade_ohe.shape, X_test_grade_ohe.shape)
print ("Teacher Prefix:", X_train_teacher_ohe.shape, X_cv_teacher_ohe.shape, X_test_teacher_ohe.shape )
```

```
School State: (49041, 51) (24155, 51) (36052, 51)
Clean Categories: (49041, 9) (24155, 9) (36052, 9)
Clean Subcategories: (49041, 30) (24155, 30) (36052, 30)
Project Grade: (49041, 4) (24155, 4) (36052, 4)
Teacher Prefix: (49041, 5) (24155, 5) (36052, 5)
```

In [148]:
```
# Normalizing Quantity numerical features
normalizer = Normalizer()
normalizer.fit(X_train["quantity"].values.reshape(1,-1))
```

Out[148]:
```
Normalizer(copy=True, norm='l2')
```

In [149]:
```
X_train_quantity_norm = normalizer.transform(X_train['quantity'].values.reshape(1,-1))
X_cv_quantity_norm = normalizer.transform(X_cv['quantity'].values.reshape(1,-1))
X_test_quantity_norm = normalizer.transform(X_test['quantity'].values.reshape(1,-1))
```

In [151]:
```
X_train_quantity_norm = X_train_quantity_norm.reshape(-1,1)
X_cv_quantity_norm = X_cv_quantity_norm.reshape(-1,1)
X_test_quantity_norm = X_test_quantity_norm.reshape(-1,1)
```

In [152]:
```
X.head(1)
```

Out[152]:

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

In [154]:
```
X_train["project_title"].head(3)
```

Out[154]:
```
12073    boys college supplies success
74379              hooked reading nook
72392          everything better color
Name: project_title, dtype: object
```

In [158]:
```
def return_pj_title_word_count(_string):
    return len(_string.strip().split(" "))
```

In [160]:
```
[return_pj_title_word_count(_str) for _str in X_train["project_title"].head(3
)]
```

Out[160]:
```
[4, 3, 3]
```

In [161]:
```
X_train["pj_title_word_count"] = [return_pj_title_word_count(_str) for _str in
X_train["project_title"]]
X_cv["pj_title_word_count"] = [return_pj_title_word_count(_str) for _str in X_
cv["project_title"]]
X_test["pj_title_word_count"] = [return_pj_title_word_count(_str) for _str in
X_test["project_title"]]
```

In [162]:
```
print (X_train["project_title"].head(10),X_train["pj_title_word_count"].head(1
0))
```

```
12073                        boys college supplies success
74379                                  hooked reading nook
72392                              everything better color
14190                                techno tablet kinders
39698                    have seat take load off your feet
98687                                  creating future now
36950                            think outside the toy box
54947                                     class technology
21252                            a table full fun learning
53448    sensory opportunities young students center ba...
Name: project_title, dtype: object 12073    4
74379    3
72392    3
14190    3
39698    7
98687    3
36950    5
54947    2
21252    5
53448    7
Name: pj_title_word_count, dtype: int64
```

In [197]:
```
normalizer = Normalizer()
normalizer.fit(X_train["pj_title_word_count"].values.reshape(1,-1))
X_train_pj_words_norm = normalizer.transform(X_train['pj_title_word_count'].va
lues.reshape(1,-1))
X_cv_pj_words_norm = normalizer.transform(X_cv['pj_title_word_count'].values.r
eshape(1,-1))
X_test_pj_words_norm = normalizer.transform(X_test['pj_title_word_count'].valu
es.reshape(1,-1))
X_train_pj_words_norm = X_train_pj_words_norm.reshape(-1,1)
X_cv_pj_words_norm = X_cv_pj_words_norm.reshape(-1,1)
X_test_pj_words_norm = X_test_pj_words_norm.reshape(-1,1)
```

In [163]:
```
X_train["essay_word_count"] = [return_pj_title_word_count(_str) for _str in X_train["essay"]]
X_cv["essay_word_count"] = [return_pj_title_word_count(_str) for _str in X_cv["essay"]]
X_test["essay_word_count"] = [return_pj_title_word_count(_str) for _str in X_test["essay"]]
```

In [198]:
```
normalizer = Normalizer()
normalizer.fit(X_train["essay_word_count"].values.reshape(1,-1))
X_train_essay_words_norm = normalizer.transform(X_train['essay_word_count'].values.reshape(1,-1))
X_cv_essay_norm = normalizer.transform(X_cv['essay_word_count'].values.reshape(1,-1))
X_test_essay_norm = normalizer.transform(X_test['essay_word_count'].values.reshape(1,-1))
X_train_essay_words_norm = X_train_essay_words_norm.reshape(-1,1)
X_cv_essay_norm = X_cv_essay_norm.reshape(-1,1)
X_test_essay_norm = X_test_essay_norm.reshape(-1,1)
```

In [166]:
```
 sid = SentimentIntensityAnalyzer()
```

In [165]:
```
print(ss)
```

```
{'neg': 0.01, 'neu': 0.745, 'pos': 0.245, 'compound': 0.9975}
```

In [191]:
```
train_ss={"neg":[],"neu":[],"pos":[],"compound":[]}
cv_ss = {"neg":[],"neu":[],"pos":[],"compound":[]}
test_ss = {"neg":[],"neu":[],"pos":[],"compound":[]}
```

In [192]:
```
for _essay in tqdm(X_train["essay"].values):
    ss= sid.polarity_scores(_essay)
    for key, value in ss.items():
        train_ss[key].extend([str(value)])
X_train["essay_scores_neg"] = train_ss["neg"]
X_train["essay_scores_neu"] = train_ss["neu"]
X_train["essay_scores_pos"] = train_ss["pos"]
X_train["essay_scores_com"] = train_ss["compound"]
```

```
100%|████████████████████████████████████████████████████████████████████████
███| 49041/49041 [01:09<00:00, 700.97it/s]
```

3. Conclusion

In [194]:

```
for _essay in tqdm(X_cv["essay"].values):
    ss= sid.polarity_scores(_essay)
    for key, value in ss.items():
        cv_ss[key].extend([str(value)])
X_cv["essay_scores_neg"] = cv_ss["neg"]
X_cv["essay_scores_neu"] = cv_ss["neu"]
X_cv["essay_scores_pos"] = cv_ss["pos"]
X_cv["essay_scores_com"] = cv_ss["compound"]
```

```
100%|████████████████████████████████████████████████████████████████████
███| 24155/24155 [00:35<00:00, 674.30it/s]
```

In [195]:

```
for _essay in tqdm(X_test["essay"].values):
    ss= sid.polarity_scores(_essay)
    for key, value in ss.items():
        test_ss[key].extend([str(value)])
X_test["essay_scores_neg"] = test_ss["neg"]
X_test["essay_scores_neu"] = test_ss["neu"]
X_test["essay_scores_pos"] = test_ss["pos"]
X_test["essay_scores_com"] = test_ss["compound"]
```

```
100%|████████████████████████████████████████████████████████████████████
███| 36052/36052 [00:52<00:00, 689.16it/s]
```

In [199]:

```
normalizer = Normalizer()
normalizer.fit(X_train["essay_scores_neg"].values.reshape(1,-1))
X_train_ess_neg_norm = normalizer.transform(X_train['essay_scores_neg'].values
.reshape(1,-1))
X_cv_ess_neg_norm = normalizer.transform(X_cv['essay_scores_neg'].values.resha
pe(1,-1))
X_test_ess_neg_norm = normalizer.transform(X_test['essay_scores_neg'].values.r
eshape(1,-1))
X_train_ess_neg_norm = X_train_ess_neg_norm.reshape(-1,1)
X_cv_ess_neg_norm = X_cv_ess_neg_norm.reshape(-1,1)
X_test_ess_neg_norm = X_test_ess_neg_norm.reshape(-1,1)
```

In [200]:

```
normalizer = Normalizer()
normalizer.fit(X_train["essay_scores_neu"].values.reshape(1,-1))
X_train_ess_neu_norm = normalizer.transform(X_train['essay_scores_neu'].values
.reshape(1,-1))
X_cv_ess_neu_norm = normalizer.transform(X_cv['essay_scores_neu'].values.resha
pe(1,-1))
X_test_ess_neu_norm = normalizer.transform(X_test['essay_scores_neu'].values.r
eshape(1,-1))
X_train_ess_neu_norm = X_train_ess_neu_norm.reshape(-1,1)
X_cv_ess_neu_norm = X_cv_ess_neu_norm.reshape(-1,1)
X_test_ess_neu_norm = X_test_ess_neu_norm.reshape(-1,1)
```

In [201]:

```
normalizer = Normalizer()
normalizer.fit(X_train["essay_scores_pos"].values.reshape(1,-1))
X_train_ess_pos_norm = normalizer.transform(X_train['essay_scores_pos'].values.reshape(1,-1))
X_cv_ess_pos_norm = normalizer.transform(X_cv['essay_scores_pos'].values.reshape(1,-1))
X_test_ess_pos_norm = normalizer.transform(X_test['essay_scores_pos'].values.reshape(1,-1))
X_train_ess_pos_norm = X_train_ess_pos_norm.reshape(-1,1)
X_cv_ess_pos_norm = X_cv_ess_pos_norm.reshape(-1,1)
X_test_ess_pos_norm = X_test_ess_pos_norm.reshape(-1,1)
```

In [202]:

```
normalizer = Normalizer()
normalizer.fit(X_train["essay_scores_com"].values.reshape(1,-1))
X_train_ess_com_norm = normalizer.transform(X_train['essay_scores_com'].values.reshape(1,-1))
X_cv_ess_com_norm = normalizer.transform(X_cv['essay_scores_com'].values.reshape(1,-1))
X_test_ess_com_norm = normalizer.transform(X_test['essay_scores_com'].values.reshape(1,-1))
X_train_ess_com_norm = X_train_ess_com_norm.reshape(-1,1)
X_cv_ess_com_norm = X_cv_ess_com_norm.reshape(-1,1)
X_test_ess_com_norm = X_test_ess_com_norm.reshape(-1,1)
```

In [203]:

```
# numeric data
print ("quantity: ", X_train_quantity_norm.shape, X_cv_quantity_norm.shape, X_test_quantity_norm.shape)
print ("pj title word count: ", X_train_pj_words_norm.shape, X_cv_pj_words_norm.shape, X_test_pj_words_norm.shape)
print ("essay word count: ", X_train_essay_words_norm.shape, X_cv_essay_norm.shape, X_test_essay_norm.shape)
print ("neg: ", X_train_ess_neg_norm.shape, X_cv_ess_neg_norm.shape, X_test_ess_neg_norm.shape)
print ("neu: ", X_train_ess_neu_norm.shape, X_cv_ess_neu_norm.shape, X_test_ess_neu_norm.shape)
print ("pos: ", X_train_ess_pos_norm.shape, X_cv_ess_pos_norm.shape, X_test_ess_pos_norm.shape)
print ("com: ", X_train_ess_com_norm.shape, X_cv_ess_com_norm.shape, X_test_ess_com_norm.shape)
print ("teacher previous: ", X_train_teach_prev_norm.shape, X_cv_teach_prev_norm.shape, X_test_teach_prev_norm.shape)
print ("price: ", X_train_price_norm.shape, X_cv_price_norm.shape, X_test_price_norm.shape)
```

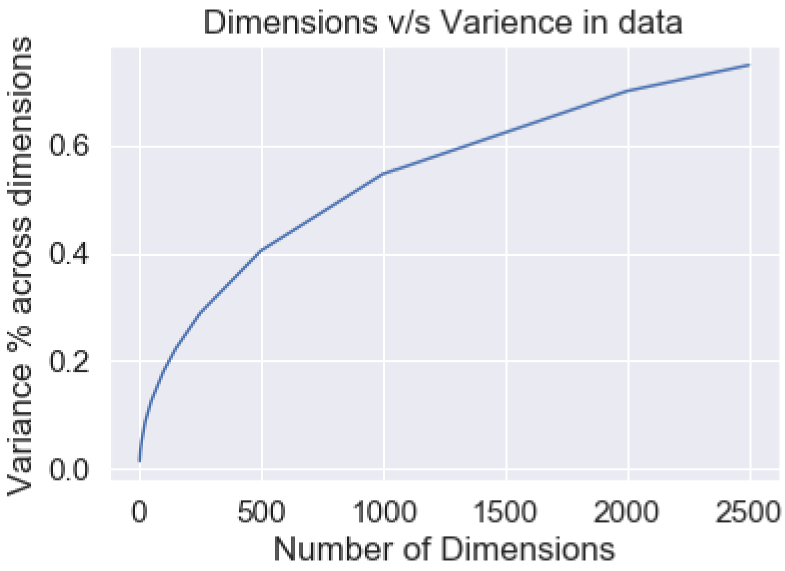
```
quantity:  (49041, 1) (24155, 1) (36052, 1)
pj title word count:  (49041, 1) (24155, 1) (36052, 1)
essay word count:  (49041, 1) (24155, 1) (36052, 1)
neg:  (49041, 1) (24155, 1) (36052, 1)
neu:  (49041, 1) (24155, 1) (36052, 1)
pos:  (49041, 1) (24155, 1) (36052, 1)
com:  (49041, 1) (24155, 1) (36052, 1)
teacher previous:  (49041, 1) (24155, 1) (36052, 1)
price:  (49041, 1) (24155, 1) (36052, 1)
```

In [207]:
```
from sklearn.decomposition import TruncatedSVD
from scipy.sparse import random as sparse_random
from sklearn.random_projection import sparse_random_matrix
```

In [209]:
```
# applying trucated SVD over n number of values to find best number of compone
nts
# reference - https://scikit-learn.org/stable/modules/generated/sklearn.decomp
osition.TruncatedSVD.html
_n_comps = [2,5,10,25,50,100,150,250,500,1000,2000,2500]
_varience_sum = []
for i in tqdm(_n_comps):
    svd = TruncatedSVD(n_components = i,random_state=42)
    svd.fit(X_train_essay_tfidf)
    _varience_sum.append(svd.explained_variance_ratio_.sum())
```

```
100%|████████████████████████████████████████████████████████████████████████
██████████| 12/12 [07:23<00:00, 36.93s/it]
```

In [210]:
```
plt.xlabel("Number of Dimensions")
plt.ylabel("Variance % across dimensions")
plt.title("Dimensions v/s Varience in data")
plt.plot(_n_comps, _varience_sum)
plt.show()
```

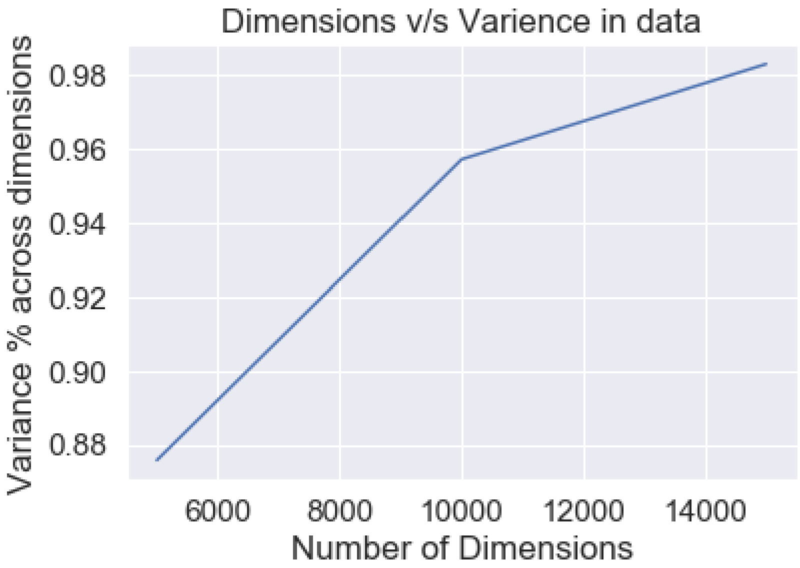


In [212]:
```
# Lets add 5000 also to number of components
_n_comps2 = [5000,10000,15000]
_varience_sum2 = []
for i in tqdm(_n_comps2):
    svd = TruncatedSVD(n_components = i,random_state=42)
    svd.fit(X_train_essay_tfidf)
    _varience_sum2.append(svd.explained_variance_ratio_.sum())
```

```
100%|████████████████████████████████████████████████████████████████████████
████████| 3/3 [1:34:24<00:00, 1888.01s/it]
```

In [214]:

```
plt.xlabel("Number of Dimensions")
plt.ylabel("Variance % across dimensions")
plt.title("Dimensions v/s Varience in data")
plt.plot(_n_comps2, _varience_sum2)
plt.show()
```



I observe that out of 40 thousand features 98% varience is maintained by 14000 components and over 90% varience by 5000 components

In [215]:

```
# concatenating all data
X_tr = hstack((X_train_quantity_norm, X_train_pj_words_norm, X_train_essay_wor
ds_norm,
             X_train_ess_neg_norm, X_train_ess_neu_norm, X_train_ess_pos_norm,
             X_train_ess_com_norm, X_train_teach_prev_norm, X_train_price_norm
,
             X_train_state_ohe, X_train_category_ohe, X_train_subcategory_ohe,
             X_train_grade_ohe, X_train_teacher_ohe)).tocsr()


X_cr = hstack((X_cv_quantity_norm, X_cv_pj_words_norm, X_cv_essay_norm,
             X_cv_ess_neg_norm, X_cv_ess_neu_norm, X_cv_ess_pos_norm,
             X_cv_ess_com_norm, X_cv_teach_prev_norm, X_cv_price_norm,
             X_cv_state_ohe, X_cv_category_ohe, X_cv_subcategory_ohe,
             X_cv_grade_ohe, X_cv_teacher_ohe)).tocsr()


X_te = hstack((X_test_quantity_norm, X_test_pj_words_norm, X_test_essay_norm,
             X_test_ess_neg_norm, X_test_ess_neu_norm, X_test_ess_pos_norm,
             X_test_ess_com_norm, X_test_teach_prev_norm, X_test_price_norm,
             X_test_state_ohe, X_test_category_ohe, X_test_subcategory_ohe,
             X_test_grade_ohe, X_test_teacher_ohe)).tocsr()
```

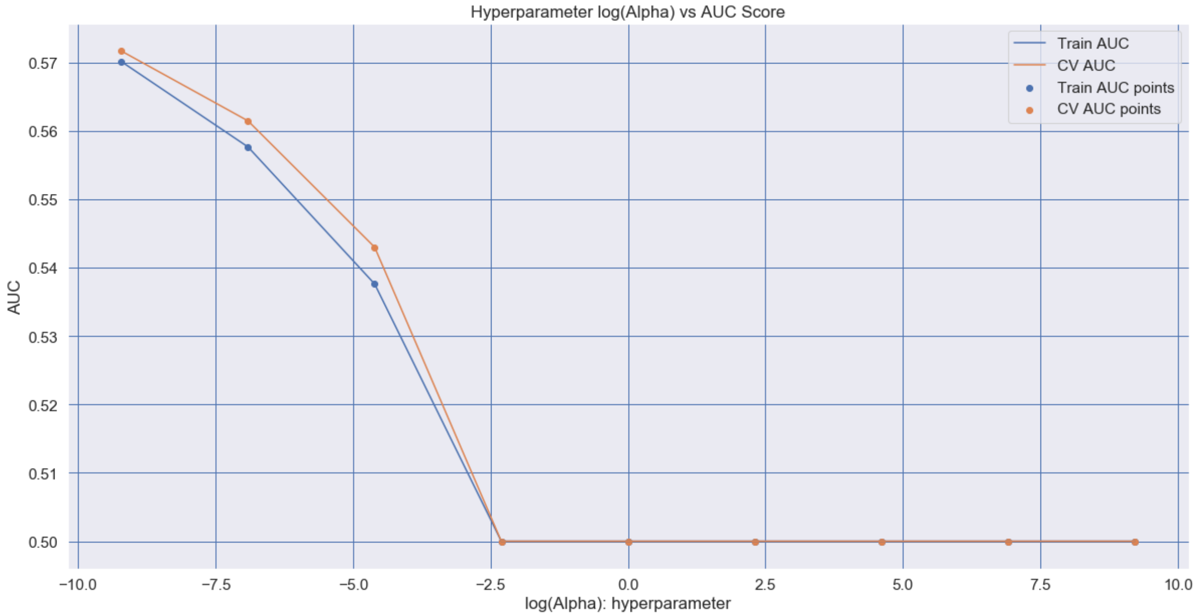
In [216]:

```
# set - 5
# SVM With L1 Penality
# L2 regulaization is default penality

# L1 Regularization - Lasso
# L2 Regularization - Ridge
train_auc = []
cv_auc = []
alpha = [10**-4, 10**-3,10**-2,10**-1,1,10,10**2,10**3,10**4]#alpha=1/C
for i in tqdm(alpha):
    base_estimator_svm_output_tfidf = SGDClassifier(loss="hinge", penalty='l1', class_weight='balanced', alpha=i)
    # Since SGDClassifier with Hinge loss doesn't have a predict_porba function lets consider this
    # as a base estimator and have a CalibrateClassiferCV on top of this
    svm_output_tfidf = CalibratedClassifierCV(base_estimator_svm_output_tfidf, cv=3)
    svm_output_tfidf.fit(X_tr, y_train)

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

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates 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%|██████████████████████████████████████████████████████████████
██████████| 9/9 [00:02<00:00,  3.45it/s]
```

In [217]:

```
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(b=True, color="b")
plt.show()
```

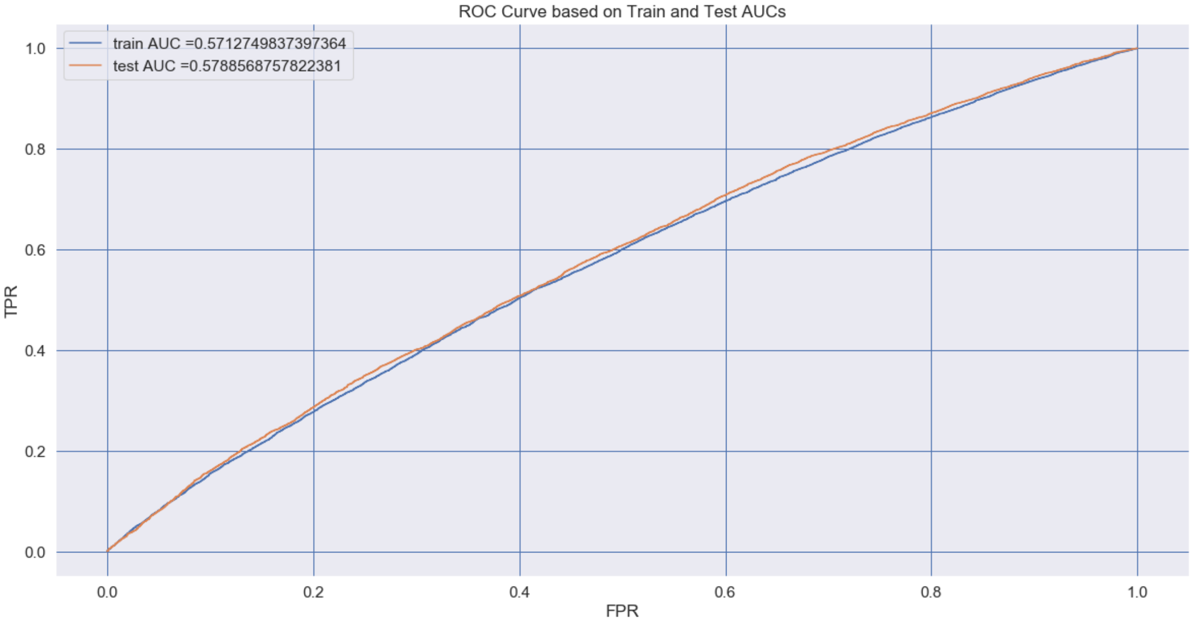


In [218]:

```
# best alpha appears to be at first point
best_alpha = 10**-4

from sklearn.metrics import roc_curve, auc


base_estimator_svm_output_tfidf = SGDClassifier(loss="hinge", penalty='l1', cl
ass_weight='balanced', alpha=best_alpha)
svm_output_tfidf = CalibratedClassifierCV(base_estimator_svm_output_tfidf, cv=
3)
svm_output_tfidf.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 = svm_output_tfidf.predict_proba(X_tr)[:,1]  # returning probabil
ity estimates of positive class
y_test_pred = svm_output_tfidf.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 [219]:

```
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(b=True, color='b')
plt.show()
```



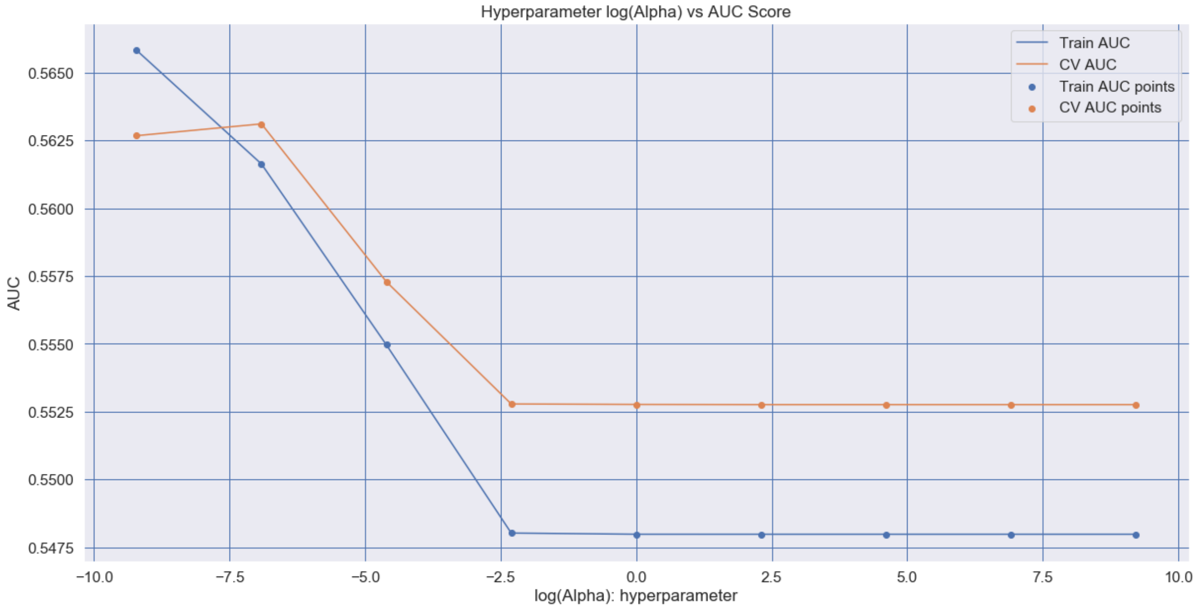
In [220]:

```
# with L2 Regularization for set 5
train_auc = []
cv_auc = []
alpha = [10**-4, 10**-3,10**-2,10**-1,1,10,10**2,10**3,10**4]#alpha=1/C
for i in tqdm(alpha):
    base_estimator_svm_output_tfidf = SGDClassifier(loss="hinge", penalty='l2', class_weight='balanced', alpha=i)
    # Since SGDClassifier with Hinge loss doesn't have a predict_porba function lets consider this
    # as a base estimator and have a CalibrateClassiferCV on top of this
    svm_output_tfidf = CalibratedClassifierCV(base_estimator_svm_output_tfidf, cv=3)
    svm_output_tfidf.fit(X_tr, y_train)

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

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates 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%|████████████████████████████████████████████████████████████████████
██████████| 9/9 [00:02<00:00,  3.59it/s]
```

In [221]:

```
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(b=True, color="b")
plt.show()
```

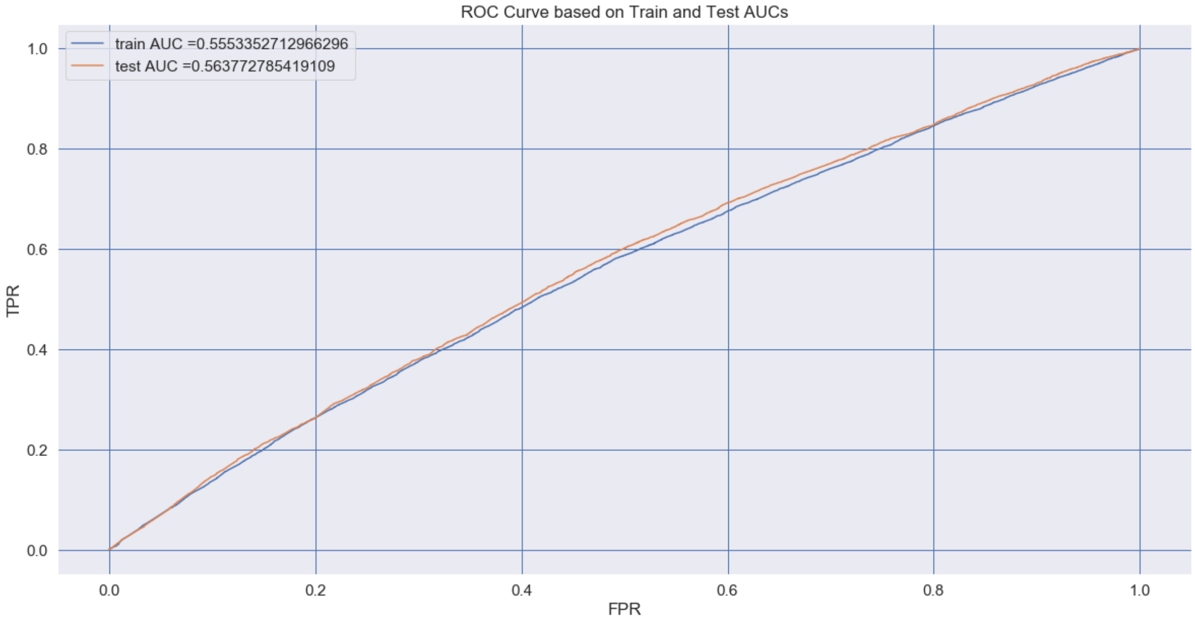


In [222]:

```
# best alpha appears to be at second point
best_alpha = 10**-3

from sklearn.metrics import roc_curve, auc


base_estimator_svm_output_tfidf = SGDClassifier(loss="hinge", penalty='l1', cl
ass_weight='balanced', alpha=best_alpha)
svm_output_tfidf = CalibratedClassifierCV(base_estimator_svm_output_tfidf, cv=
3)
svm_output_tfidf.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 = svm_output_tfidf.predict_proba(X_tr)[:,1]  # returning probabil
ity estimates of positive class
y_test_pred = svm_output_tfidf.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 [223]:

```
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(b=True, color='b')
plt.show()
```



In [224]:

```
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)
```

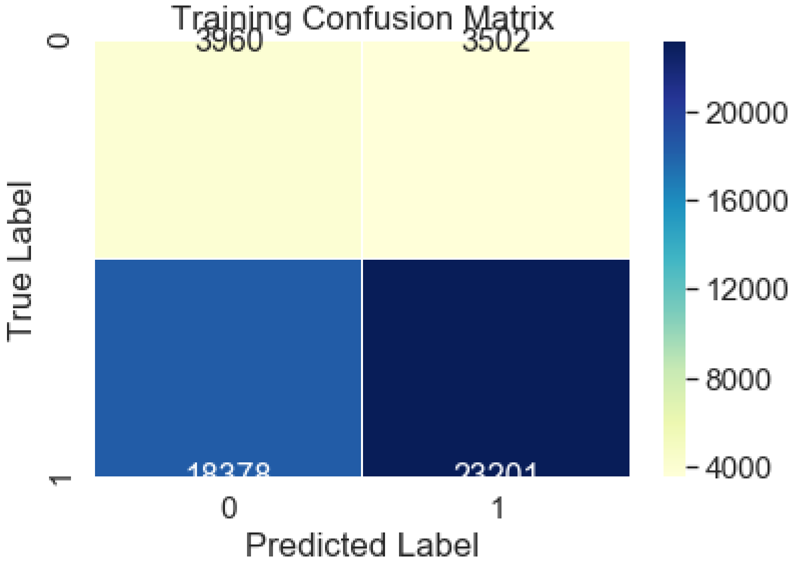
```
=============================================================================
=======================
the maximum value of tpr*(1-fpr) 0.2961233168437908 for threshold 0.848
Train confusion matrix
[[ 3960  3502]
 [18378 23201]]
Test confusion matrix
[[ 2920  2539]
 [13268 17325]]
```

In [225]:

```
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[225]:

```
Text(0.5, 1, 'Training Confusion Matrix')
```

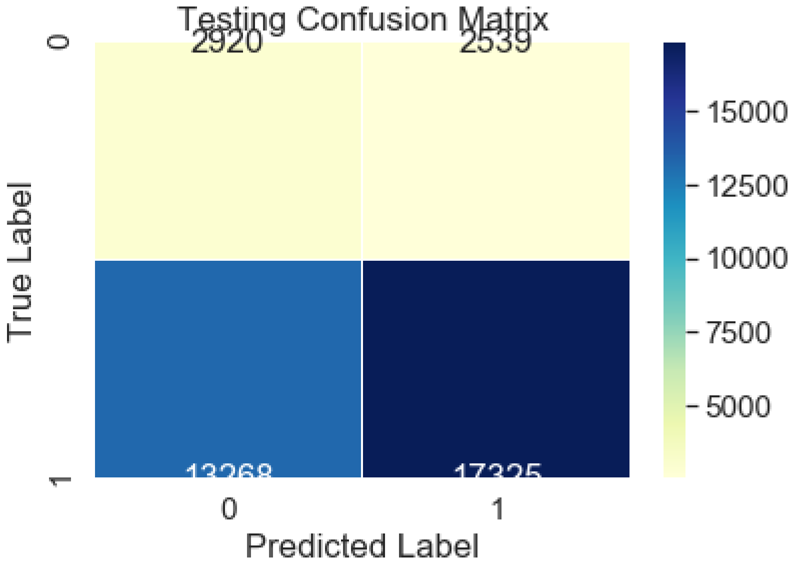


In [226]:

```
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[226]:

```
Text(0.5, 1, 'Testing Confusion Matrix')
```



The AUC of set 5 clearly shows the model is dumb without any text features such as pj title or essay.

Summary

In [227]:
```python
# Please compare all your models using Prettytable library
from prettytable import PrettyTable

#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable

x = PrettyTable()
x.field_names = ["Vectorizer", "Regularization", "Hyperparameter (alpha)", "Train AUC", "Test AUC"]

x.add_row(["BOW","L2",0.01,0.767,0.678])
x.add_row(["BOW","L1",0.001,0.682,0.64])
x.add_row(["TFIDF","L1",0.0001,0.781,0.677])
x.add_row(["TFIDF","L2",0.001,0.737,0.676])
x.add_row(["W2V","L1",0.0001,0.725,0.701])
x.add_row(["W2V","L2",0.0001,0.723,0.699])
x.add_row(["TFIDF W2V","L1",0.001,0.693,0.688])
x.add_row(["TFIDF W2V","L2",0.001,0.714,0.696])
x.add_row(["Dumb Model", "L1", 0.0001, 0.571, 0.578])
x.add_row(["Dumb Model", "L2", 0.001, 0.555, 0.563])
print(x)
```

```
+------------+----------------+------------------------+-----------+----------+
| Vectorizer | Regularization | Hyperparameter (alpha) | Train AUC | Test AUC |
+------------+----------------+------------------------+-----------+----------+
|    BOW     |       L2       |          0.01          |   0.767   |  0.678   |
|    BOW     |       L1       |         0.001          |   0.682   |   0.64   |
|   TFIDF    |       L1       |         0.0001         |   0.781   |  0.677   |
|   TFIDF    |       L2       |         0.001          |   0.737   |  0.676   |
|    W2V     |       L1       |         0.0001         |   0.725   |  0.701   |
|    W2V     |       L2       |         0.0001         |   0.723   |  0.699   |
| TFIDF W2V  |       L1       |         0.001          |   0.693   |  0.688   |
| TFIDF W2V  |       L2       |         0.001          |   0.714   |  0.696   |
| Dumb Model |       L1       |         0.0001         |   0.571   |  0.578   |
| Dumb Model |       L2       |         0.001          |   0.555   |  0.563   |
+------------+----------------+------------------------+-----------+----------+
```

In [ ]: