DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located (Two-letter U.S. postal code). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy` • `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • `My students need hands on literacy materials to manage sensory needs!`
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*

Feature	Description
`project_essay_4`	Fourth application essay
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
`teacher_id`	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same teacher. **Example:** `2`

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [1]:

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

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

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

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

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

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

from tqdm import tqdm
import os

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

1.1 Reading Data

In [2]:

```
project_data = pd.read_csv('train_data.csv', nrows = 15000)
resource_data = pd.read_csv('resources.csv')
resource_data.head()
```

Out[2]:

	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
2	p069063	Cory Stories: A Kid's Book About Living With Adhd	1	8.45
3	p069063	Dixon Ticonderoga Wood-Cased #2 HB Pencils, Bo...	2	13.59
4	p069063	EDUCATIONAL INSIGHTS FLUORESCENT LIGHT FILTERS...	3	24.95

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 (15000, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [4]:

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

```
resource_data.head(2)
```

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

Out[4]:

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

1.2 preprocessing of `project_subject_categories`

In [5]:

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

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

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [6]:

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

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

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
```

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

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

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

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

1.3 Text preprocessing

In [7]:

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

In [8]:

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

Out[8]:

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

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


---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-10-009ab9740f10> in <module>
      6 print(project_data['essay'].values[1000])
      7 print("="*50)
----> 8 print(project_data['essay'].values[20000])
      9 print("="*50)
     10 print(project_data['essay'].values[99999])

IndexError: index 20000 is out of bounds for axis 0 with size 15000
```

In [11]:

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

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

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

In [12]:

```
sent = decontracted(project_data['essay'].values[14000])
print(sent)
print("="*50)
```

```
Our school is located in a northern suburb of Atlanta in a diverse community. Our students come fr
om various backgrounds, cultures, and countries. \r\n\r\nOur school is a Title I public school and
my classroom alone serves around 120 students. \r\n\r\nMany of the students in my classroom are be
ing served in one or more of the following programs: ESOL (English as a second language), Special
Education, and TAG (our district is Gifted Program). The students that I have the privilege to tea
ch are so sweet and work hard every day.My students need a class set of VR Pasonomi 3D VR Glasses
to give them the opportunity to explore the world without leaving the classroom. \r\n\r\nWith the
3D VR classes, students will not just read about a location, they would experience it firsthand wi
th a 360 degree panoramic view. Imagine virtual field-trips in Australia at Ayers Rocker,
exploring the ancient Incan city of Machu Picchu or a walking though the Amazon Rainforest. When w
e are studying about Europe, I will have my students gaze up at and walk around cities and histori
cal landmarks. While they are 'visiting' these wonders, students can hear commentary about what th
ey are seeing (with certain applications) or even create their own commentary, without the
thousands of dollars it would take to fly to these areas and be on time to math the next
period.\r\n\r\nHaving this class set available to my students would definitely take my class to
the next level and open students' eyes to the world around them without having to leave the
classroom. Students will be able to dig deeper into any subject or location that we are studying i
n social studies.\r\n\r\nI can imagine conversations about the locations we visit- "is that really
where the Aztecs would perform human sacrifice!?" I am excited at the uses of this technology and
know that my students would thrive on our virtual trips!\r\nnannan
==================================================
```

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)
```

```
Our school is located in a northern suburb of Atlanta in a diverse community. Our students come fr
om various backgrounds, cultures, and countries.     Our school is a Title I public school and my
classroom alone serves around 120 students.     Many of the students in my classroom are being ser
ved in one or more of the following programs: ESOL (English as a second language), Special
Education, and TAG (our district is Gifted Program). The students that I have the privilege to tea
ch are so sweet and work hard every day.My students need a class set of VR Pasonomi 3D VR Glasses
to give them the opportunity to explore the world without leaving the classroom.     With the 3D V
R classes, students will not just read about a location, they would experience it firsthand with a
360 degree panoramic view. Imagine virtual field-trips in Australia at Ayers Rocker, exploring the
ancient Incan city of Machu Picchu or a walking though the Amazon Rainforest. When we are studying
about Europe, I will have my students gaze up at and walk around cities and historical landmarks.
While they are 'visiting' these wonders, students can hear commentary about what they are seeing (
with certain applications) or even create their own commentary, without the thousands of dollars i
t would take to fly to these areas and be on time to math the next period.    Having this class
set available to my students would definitely take my class to the next level and open students' e
yes to the world around them without having to leave the classroom. Students will be able to dig d
```

eeper into any subject or location that we are studying in social studies. I can imagine conver
sations about the locations we visit- "is that really where the Aztecs would perform human
sacrifice!?" I am excited at the uses of this technology and know that my students would thrive on
our virtual trips! nannan

In [14]:

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

```
Our school is located in a northern suburb of Atlanta in a diverse community Our students come fro
m various backgrounds cultures and countries Our school is a Title I public school and my classroo
m alone serves around 120 students Many of the students in my classroom are being served in one or
more of the following programs ESOL English as a second language Special Education and TAG our dis
trict is Gifted Program The students that I have the privilege to teach are so sweet and work hard
every day My students need a class set of VR Pasonomi 3D VR Glasses to give them the opportunity t
o explore the world without leaving the classroom With the 3D VR classes students will not just re
ad about a location they would experience it firsthand with a 360 degree panoramic view Imagine vi
rtual field trips in Australia at Ayers Rocker exploring the ancient Incan city of Machu Picchu or
a walking though the Amazon Rainforest When we are studying about Europe I will have my students g
aze up at and walk around cities and historical landmarks While they are visiting these wonders st
udents can hear commentary about what they are seeing with certain applications or even create the
ir own commentary without the thousands of dollars it would take to fly to these areas and be on t
ime to math the next period Having this class set available to my students would definitely take m
y class to the next level and open students eyes to the world around them without having to leave
the classroom Students will be able to dig deeper into any subject or location that we are studyin
g in social studies I can imagine conversations about the locations we visit is that really where
the Aztecs would perform human sacrifice I am excited at the uses of this technology and know that
my students would thrive on our virtual trips 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', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "d
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [16]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
```

```
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████| 15000/15000
[00:12<00:00, 1196.50it/s]
```

In [17]:

```
# after preprocesing
preprocessed_essays[14000]
```

Out[17]:

```
'our school located northern suburb atlanta diverse community our students come various
backgrounds cultures countries our school title i public school classroom alone serves around 120
students many students classroom served one following programs esol english second language specia
l education tag district gifted program the students i privilege teach sweet work hard every day m
y students need class set vr pasonomi 3d vr glasses give opportunity explore world without leaving
classroom with 3d vr classes students not read location would experience firsthand 360 degree pano
ramic view imagine virtual field trips australia ayers rocker exploring ancient incan city machu
picchu walking though amazon rainforest when studying europe i students gaze walk around cities hi
storical landmarks while visiting wonders students hear commentary seeing certain applications eve
n create commentary without thousands dollars would take fly areas time math next period having cl
ass set available students would definitely take class next level open students eyes world around
without leave classroom students able dig deeper subject location studying social studies i imagin
e conversations locations visit really aztecs would perform human sacrifice i excited uses
technology know students would thrive virtual trips nannan'
```

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%|████████████████████████████████████████████████████████████████| 15000/15000
[00:00<00:00, 26753.88it/s]
```

1.5 Preparing data for models

In [20]:

```
project_data.columns
```

Out[20]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', 'project_title',
       'project_essay_1', 'project_essay_2', 'project_essay_3',
```

```
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay'],
      dtype='object')
```

we are going to consider

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

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

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

1.5.1 Vectorizing Categorical data

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

In [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  (15000, 9)
```

In [22]:

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

```
['Economics', 'FinancialLiteracy', 'CommunityService', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig  (15000, 30)
```

In [23]:

```
# you can do the similar thing with state, teacher_prefix and project_grade_category 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).values)
```

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', 'IA', 'ID', 'IL', 'IN', 'K
S', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM',
'NV', 'NY', 'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA', 'WI', 'WV
', 'WY']
==================================================
Shape of matrix after one hot encodig  (15000, 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' :
rades_6_8',
                                                                                     'Grades 3-5' :
rades_3_5',
                                                                                     'Grades 9-12'
Grades_9_12'})
project_data['teacher_prefix'] = project_data['teacher_prefix'].map({'Mrs.': 'Mrs', 'Ms.': 'Ms', 'M
r.' : 'Mr',
                                                                     'Teacher': 'Teacher', 'Dr.' :
'Dr'})
```

In [26]:

```
# 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)
```

```
['Mr', 'Mrs', 'Ms', 'Teacher']
['Mr', 'Mrs', 'Ms', 'Teacher']
==================================================
Shape of matrix after one hot encodig  (15000, 4)
```

In [27]:

```
# 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  (15000, 4)
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [28]:

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

```
Shape of matrix after one hot encodig  (15000, 7465)
```

In [29]:

```
# 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  (15000, 912)
```

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  (15000, 7465)
```

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  (15000, 912)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [32]:

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

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

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

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

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

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

```
print("the unique words in the coupus", len(words))

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

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


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

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


'''
```

Out[32]:

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

In [33]:

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

In [34]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors.append(vector)

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

```
100%|██████████| 15000/15000 [00:07<00:00, 2127.66it/s]
```

```
15000
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [35]:

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

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

```
100%|██████████████████████████████████████████████████████████████████████████████| 15000/15000 [00:
49<00:00, 305.42it/s]
```

```
15000
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/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
```

```
    tfidf_w2v_vectors_titles.append(vector)

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

```
100%|██████████████████████████████████████████████████████████████████████████| 15000/15000
[00:00<00:00, 21913.88it/s]
```

```
15000
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/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler

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

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

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

```
Mean : 297.8444793333333, Standard deviation : 383.6922825999444
```

In [41]:

```
price_standardized
```

Out[41]:

```
array([[-0.37333167],
       [ 0.00301158],
       [ 0.57078427],
       ...,
       [-0.31906943],
       [-0.26024625],
       [-0.48414964]])
```

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_
osted_projects'].values.reshape(-1,1)) # finding the mean and standard deviation of this data
print(f"Mean : {teacher_number_of_previously_posted_projects_scalar.mean_[0]}, Standard deviation
: {np.sqrt(teacher_number_of_previously_posted_projects_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
teacher_number_of_previously_posted_projects_standardized =
teacher_number_of_previously_posted_projects_scalar.transform(project_data['teacher_number_of_previ
ously_posted_projects'].values.reshape(-1, 1))
```

```
Mean : 11.1954, Standard deviation : 27.548624022020892
```

1.5.4 Merging all the above features

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

In [43]:

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

```
(15000, 9)
(15000, 30)
(15000, 7465)
(15000, 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 dense matirx :)
X = hstack((school_state_one_hot,categories_one_hot, sub_categories_one_hot,
teacher_prefix_one_hot,
            grade_one_hot, text_bow_titles, text_bow, price_standardized,
            teacher_number_of_previously_posted_projects_standardized))
X.shape
```

Out[44]:

```
(15000, 8477)
```

In [45]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# 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
```

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 w
ith the biggest enthusiasm \
for learning my students learn in many different ways using all of our senses and multiple intelli
gences i use a wide range\
of techniques to help all my students succeed students in my class come from a variety of differen
t backgrounds which makes\
for wonderful sharing of experiences and cultures including native americans our school is a carin
g community of successful \
learners which can be seen through collaborative student project based learning in and out of the
classroom kindergarteners \
in my class love to work with hands on materials and have many different opportunities to practice
a skill before it is\
mastered having the social skills to work cooperatively with friends is a crucial aspect of the ki
ndergarten curriculum\
```

```
montana is the perfect place to learn about agriculture and nutrition my students love to role pla
y in our pretend kitchen\
in the early childhood classroom i have had several kids ask me can we try cooking with real food
i will take their idea \
and create common core cooking lessons where we learn important math and writing concepts while co
oking delicious healthy \
food for snack time my students will have a grounded appreciation for the work that went into maki
ng the food and knowledge \
of where the ingredients came from as well as how it is healthy for their bodies this project woul
d expand our learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade apple
sauce make our own bread \
and mix up healthy plants from our classroom garden in the spring we will also create our own cook
books to be printed and \
shared with families students will gain math and literature skills as well as a life long enjoymen
t for healthy cooking \
nannan'
ss = sid.polarity_scores(for_sentiment)

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

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

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

Assignment 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 value
   - Find the best hyper paramter using k-fold cross validation or simple cross validation data
   - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning
3. **Representation of results**
   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.
   - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
   - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.
4. **[Task-2] Apply the Support Vector Machines on these features by finding the best hyper paramter as suggested in step 2 and step 3**
   - Consider these set of features Set 5 :
     - **school_state** : categorical data
     - **clean_categories** : categorical data
     - **clean_subcategories** : categorical data
     - **project_grade_category** :categorical data
     - **teacher_prefix** : categorical data
     - **quantity** : numerical data
     - **teacher_number_of_previously_posted_projects** : numerical data
     - **price** : numerical data
     - **sentiment score's of each of the essay** : numerical data
     - **number of words in the title** : numerical data
     - **number of words in the combine essays** : numerical data
     - **Apply TruncatedSVD on TfidfVectorizer of essay text, choose the number of components (`n_components`)**

using elbow method : 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

---

Note: Data Leakage

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

2. Support Vector Machines

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

In [47]:

```
# 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]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_categ
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs	IN	2016-12-05 13:43:57	Grades_Prel

In [48]:

```
# 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)
```

```
(6733, 19) (6733,)
(3317, 19) (3317,)
(4950, 19) (4950,)
```

In [49]:

```
# Encoding School State - OHE
# School State
vectorizer = CountVectorizer()
```

```
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())
```

```
After vectorizations
(6733, 51) (6733,)
(3317, 51) (3317,)
(4950, 51) (4950,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'k
s', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm',
'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv
', 'wy']
```

In [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())
```

```
After vectorizations
(6733, 4) (6733,)
(3317, 4) (3317,)
(4950, 4) (4950,)
['mr', 'mrs', 'ms', 'teacher']
```

In [51]:

```
# 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())
```

```
After vectorizations
(6733, 4) (6733,)
(3317, 4) (3317,)
(4950, 4) (4950,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
```

In [52]:

```
# Encoding Categories
```

```
# 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())
```

```
After vectorizations
(6733, 9) (6733,)
(3317, 9) (3317,)
(4950, 9) (4950,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
```

In [53]:

```
# 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())
```

```
After vectorizations
(6733, 30) (6733,)
(3317, 30) (3317,)
(4950, 30) (4950,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience',
'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness',
'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'm
athematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socia
lsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
```

In [54]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
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)
```

In [55]:

```
# teacher previously posted projects
normalizer = Normalizer()
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'].v
alues.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)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [56]:

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

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

In [57]:

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

Out[57]:

```
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 [58]:

```
# 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 [59]:

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

Out[59]:

```
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=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)
```

2.3 Make Data Model Ready: encoding eassay, and project_title

In [61]:

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

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

In [62]:

```
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['essay'].values)
```

Out[62]:

```
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=(1, 4), preprocessor=None, stop_words=None,
                strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, vocabulary=None)
```

In [63]:

```
# 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 [64]:

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

Out[64]:

```
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=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 [65]:

```
# 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)
```

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

In [66]:

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

In [67]:

```
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)).tocsr()

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 [68]:

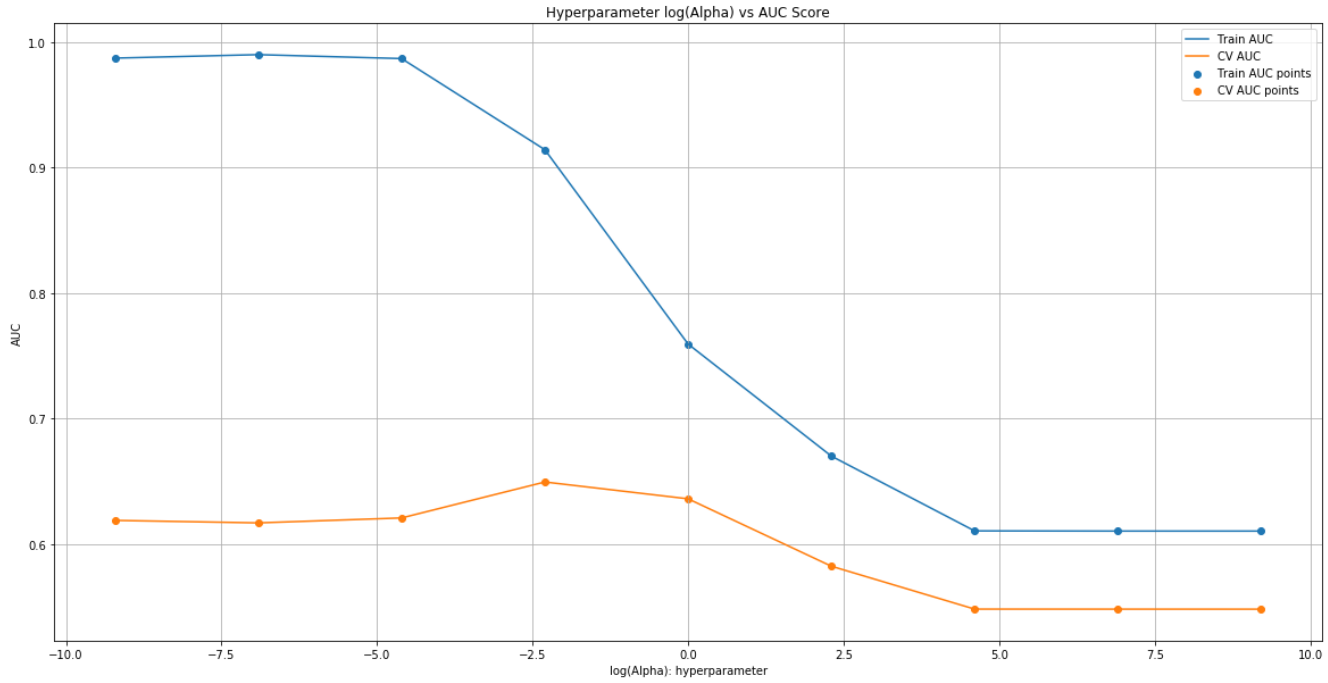
```
# Necessary Package imports
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn.calibration import CalibratedClassifierCV

# 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='balance
d', 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 grea
ter 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 posi
tive 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
:04<00:00,  1.98it/s]
```

In [69]:

```
# 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()
```

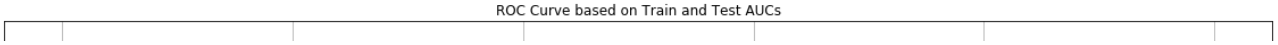


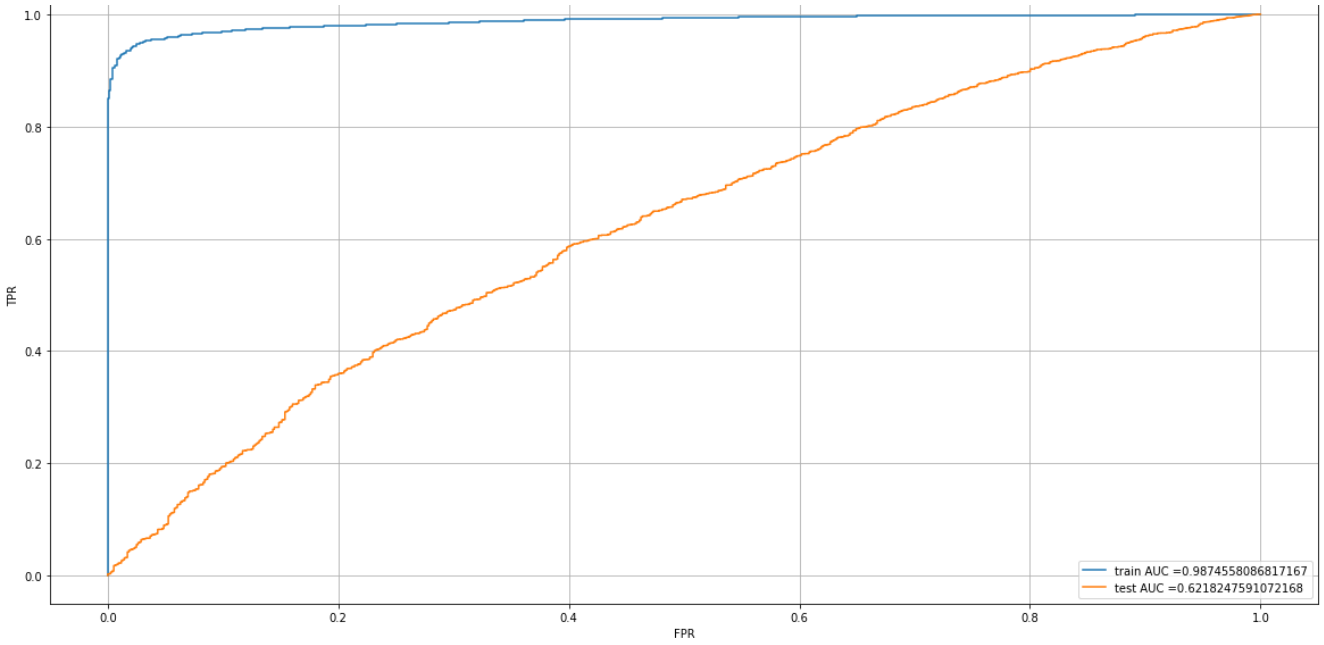
In [70]:

```
# 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', 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 [71]:

```
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()
plt.show()
```





In [72]:

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

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

In [73]:

```
# 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) 0.9231209449803321 for threshold 0.799
Train confusion matrix
[[1015   26]
 [ 303 5389]]
Test confusion matrix
[[ 219  542]
 [ 662 3527]]
```

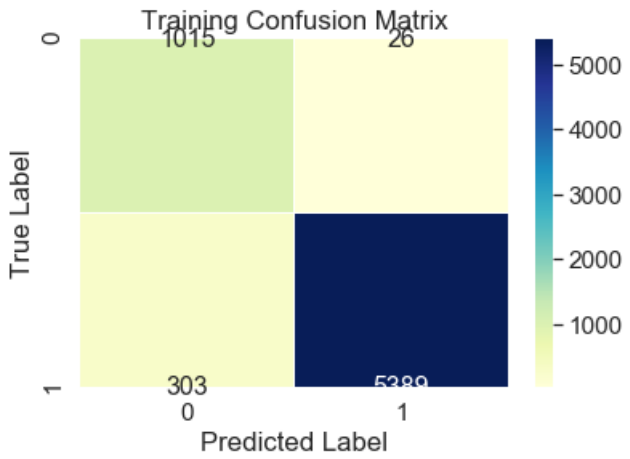
In [74]:

```
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[74]:

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

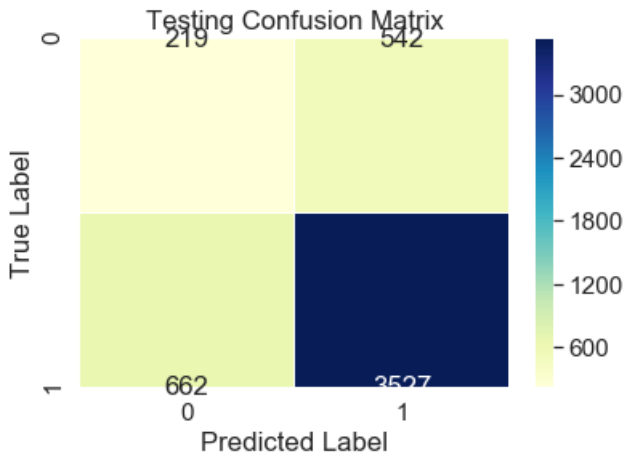


In [75]:

```
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[75]:

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



In [76]:

```
#Set -1 BOW with L1 Penality
```

In [77]:

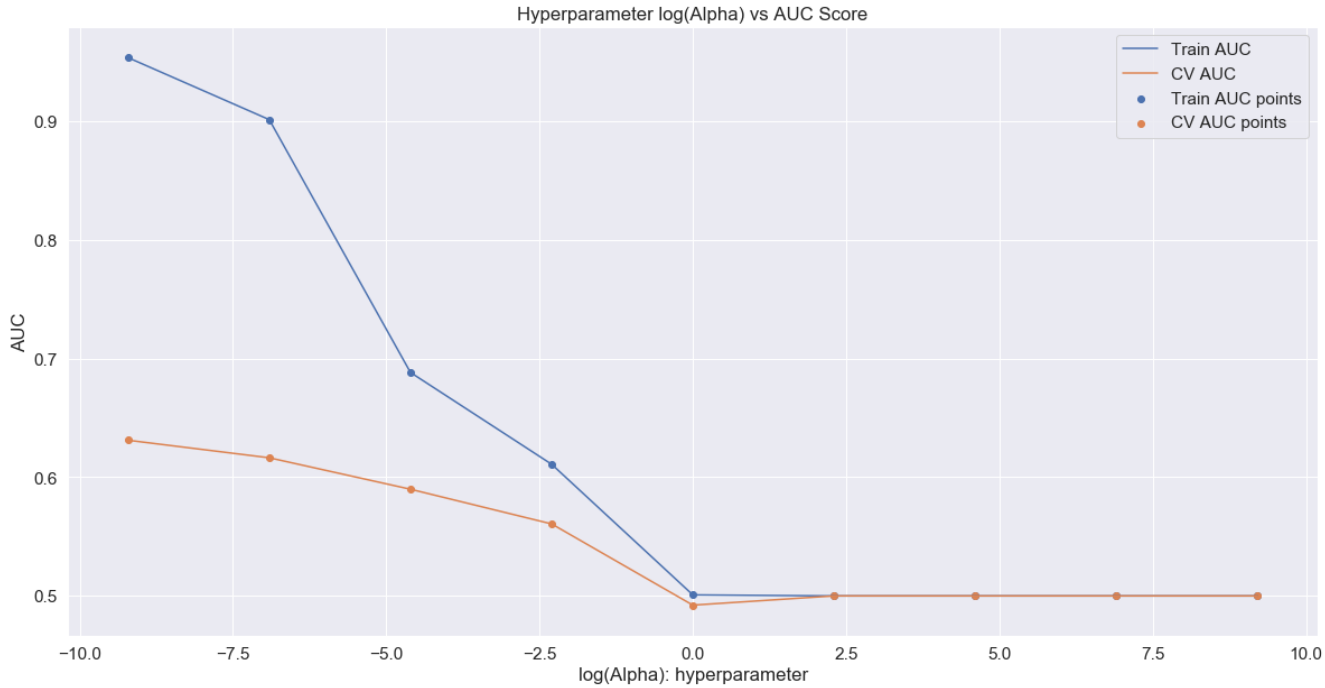
```
# SVM With L2 Penality
# L2 regulaization is default penality

# 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='balance
d', 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 grea
ter 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 posi
tive 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,  9.03s/it]
```

In [78]:

```
# 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()
```



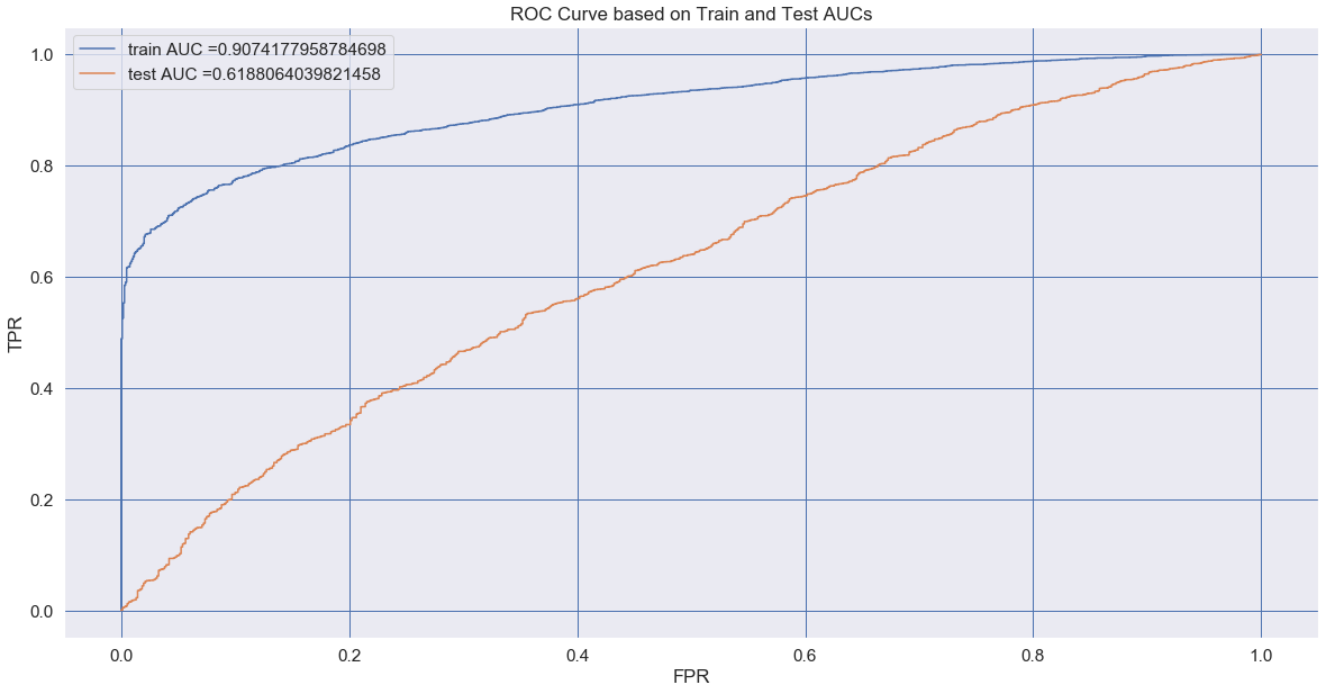
In [79]:

```
# 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', 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 [80]:

```
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()
```



In [81]:

```
# 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) 0.6992141590435166 for threshold 0.829
Train confusion matrix
[[ 952   89]
 [1340 4352]]
Test confusion matrix
[[ 348  413]
 [1304 2885]]
```

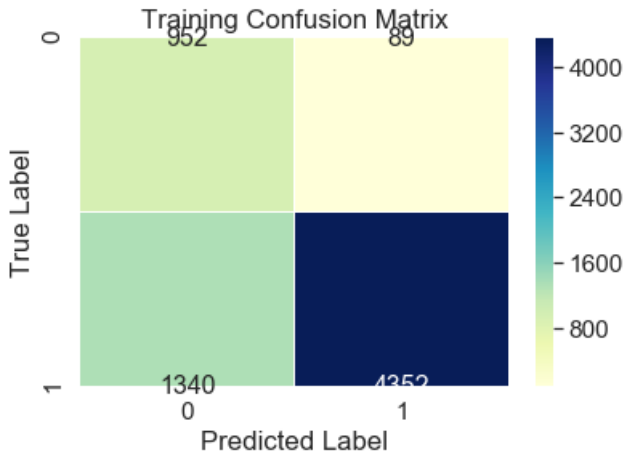
[1304 2885]]

In [82]:

```
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[82]:

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

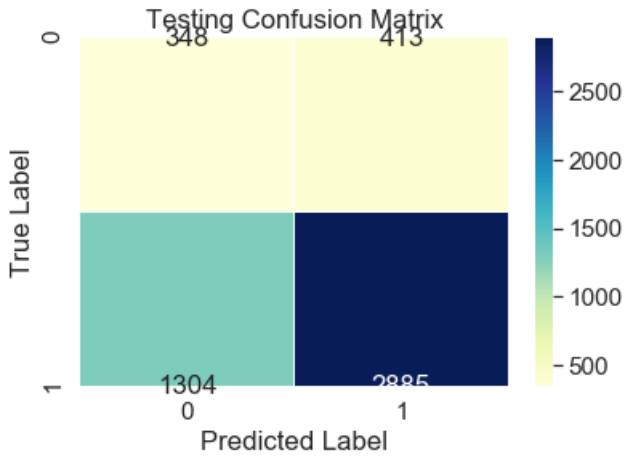


In [83]:

```
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[83]:

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



In [84]:

```
# 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_train_score=True)
clf.fit(X_tr, y_train)
```

Out[84]:

```
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=0.001,
                                     validation_fraction=0.1, verbose=0,
                                     warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'alpha': array([1.0000000e-04, 5.0001000e+00, 1.0000100e+01, ..., 9.985000
1e+03,
       9.9900001e+03, 9.9950001e+03])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [85]:

```
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:  0.636653806648036
k value with best score:  {'alpha': 5.0001}
============================================================================
Train AUC scores
[0.99974269 0.71126286 0.67707306 ... 0.61446246 0.61447486 0.61445409]
CV AUC scores
[0.62577707 0.63665381 0.62463318 ... 0.5926726  0.59267095 0.59265994]
```

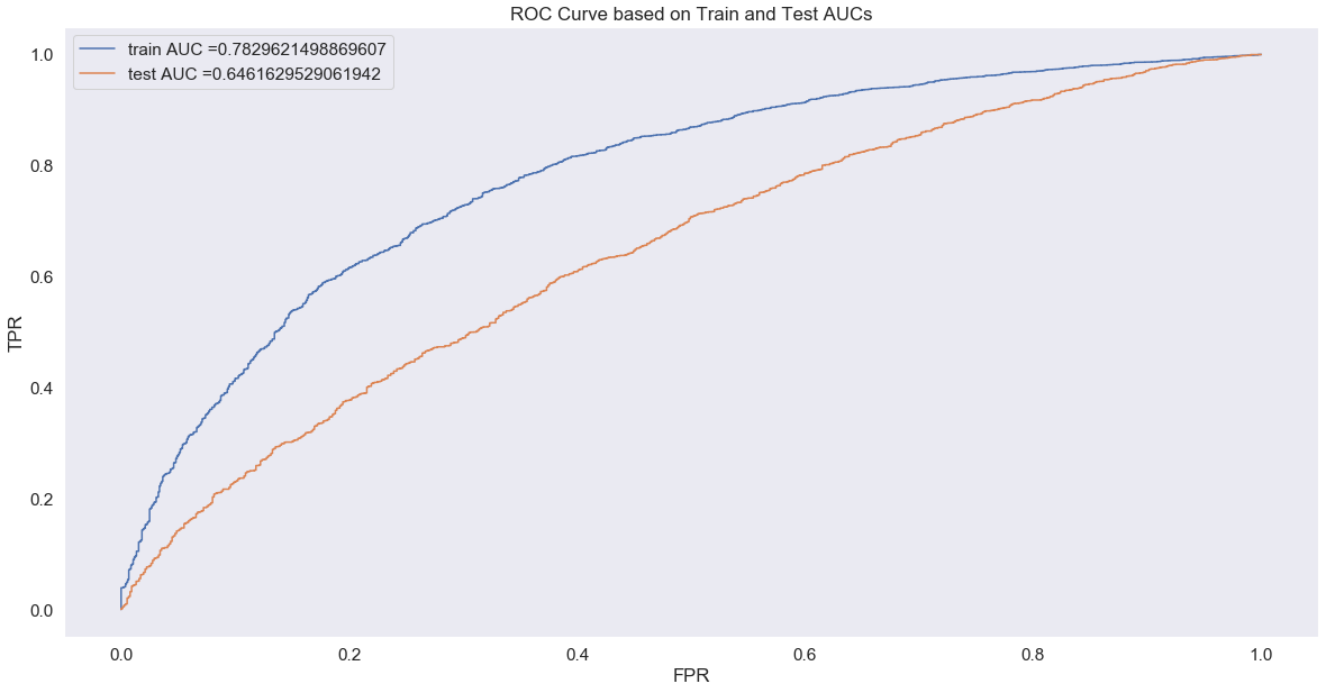
In [86]:

```
# 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 [87]:

```
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()
plt.show()
```



In [88]:

```
### Set 2 TFIDF
```

In [89]:

```
# 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
(6733, 20204) (6733,)
(3317, 20204) (3317,)
(4950, 20204) (4950,)
====================================================================================================
```

In [90]:

```
# 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)
```

In [91]:

```
# 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)).tocsr()

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()
```

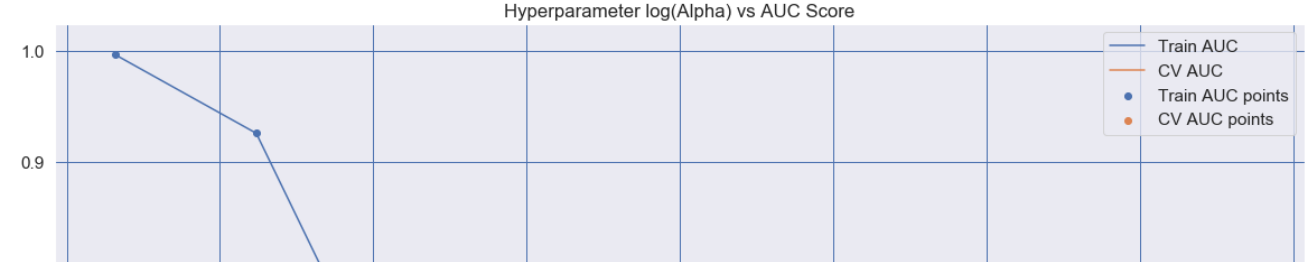
In [92]:

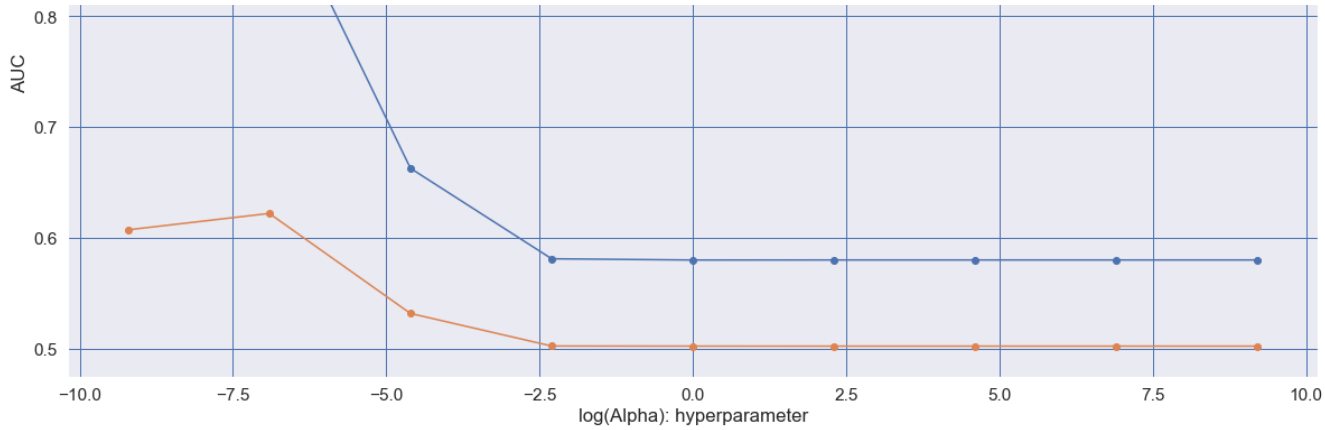
```
# SVM With L2 Penality
# L2 regulaization is default penality

# 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='balan
ced', 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 gr
eater 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 posi
tive 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,  3.17it/s]
```

In [93]:

```
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()
```



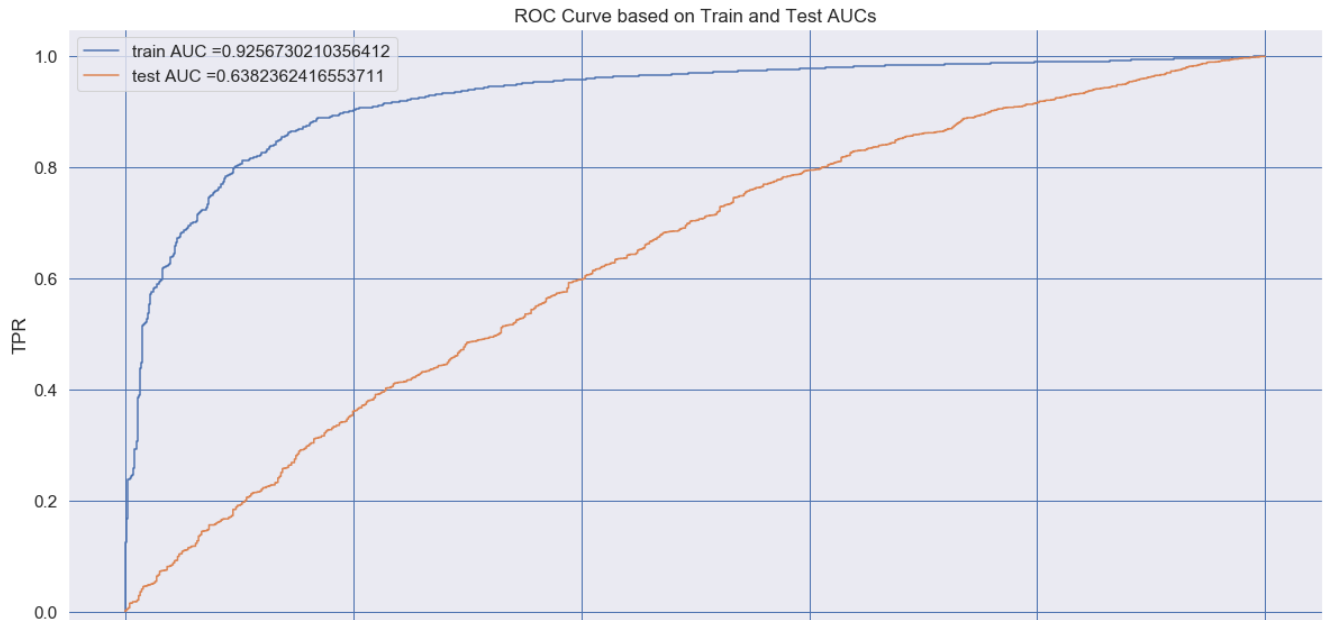


In [94]:

```
# 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', 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 [95]:

```
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()
```



In [96]:

```
# 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)
```

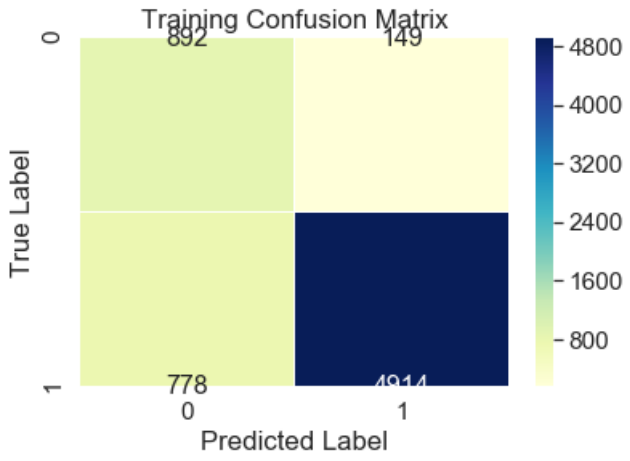
```
====================================================================================================

the maximum value of tpr*(1-fpr) 0.7397489980375915 for threshold 0.812
Train confusion matrix
[[ 892  149]
 [ 778 4914]]
Test confusion matrix
[[ 326  435]
 [ 936 3253]]
```

In [97]:

```
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[97]:

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

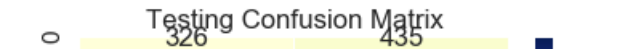


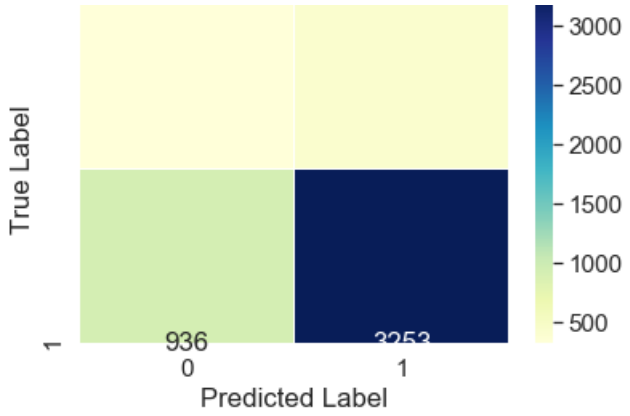
In [98]:

```
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[98]:

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





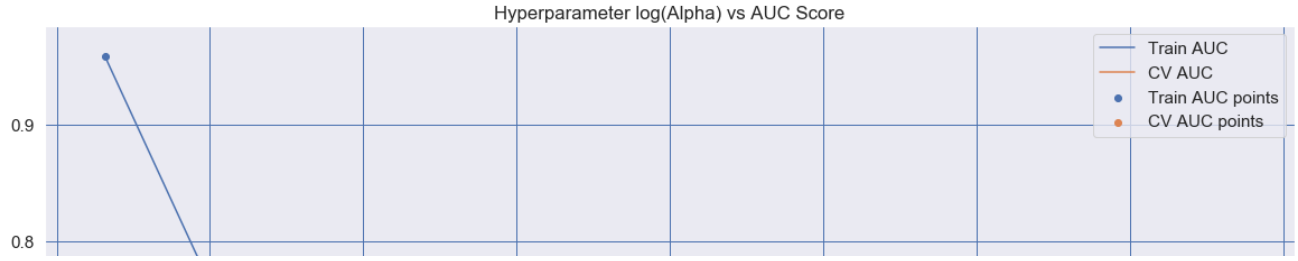
In [99]:

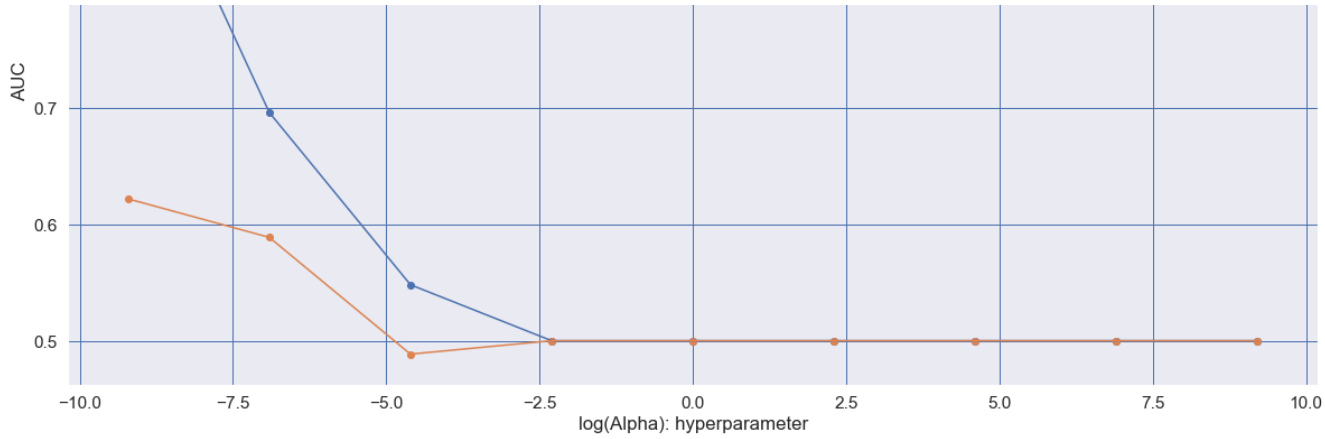
```
# SVM With L1 Penality
# L2 regulaization is default penality

# 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='balan
ced', 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 gr
eater 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 posi
tive 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
:05<00:00,  1.73it/s]
```

In [100]:

```
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()
```



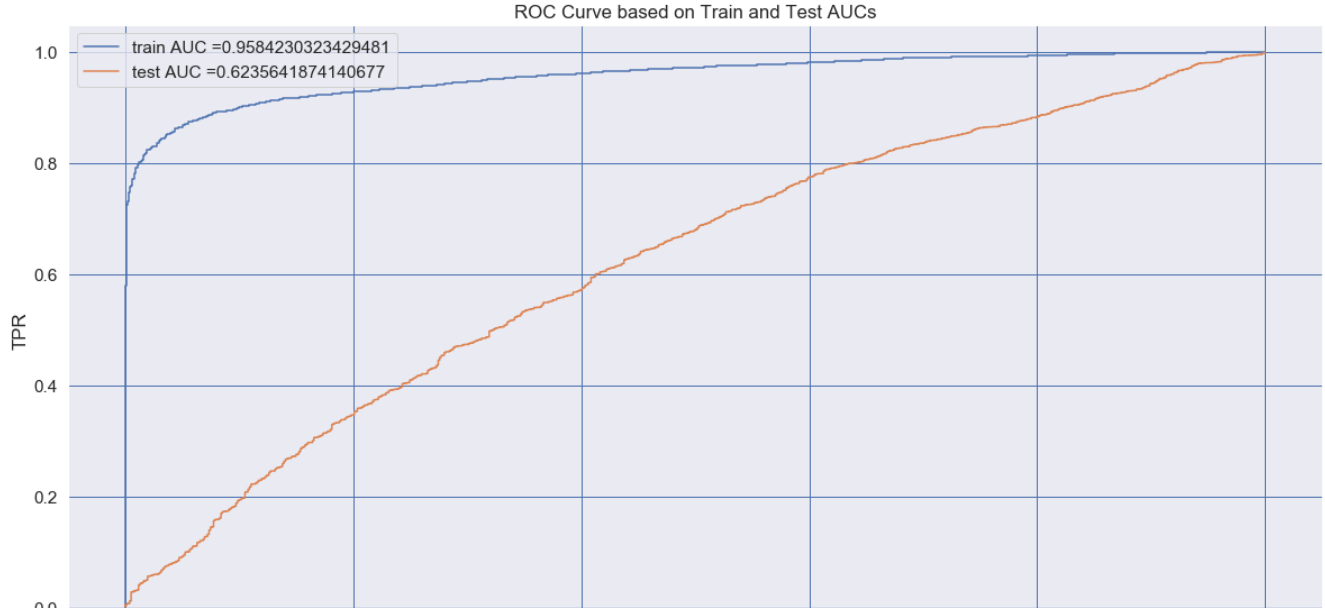


In [101]:

```
# 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',
class_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 estimates of the positive
class
# not the predicted outputs

y_train_pred = svm_output_tfidf.predict_proba(X_tr)[:,1]  # returning probability 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 [102]:

```
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()
```



In [103]:

```
# 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)
```

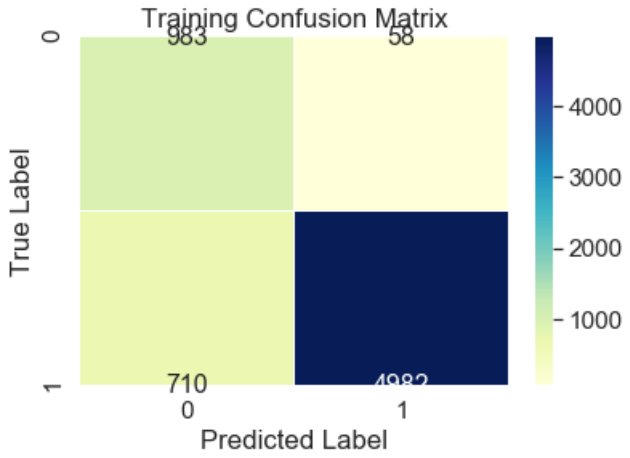
```
====================================================================================================

the maximum value of tpr*(1-fpr) 0.8264976443673072 for threshold 0.818
Train confusion matrix
[[ 983   58]
 [ 710 4982]]
Test confusion matrix
[[ 333  428]
 [1100 3089]]
```

In [104]:

```
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[104]:

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

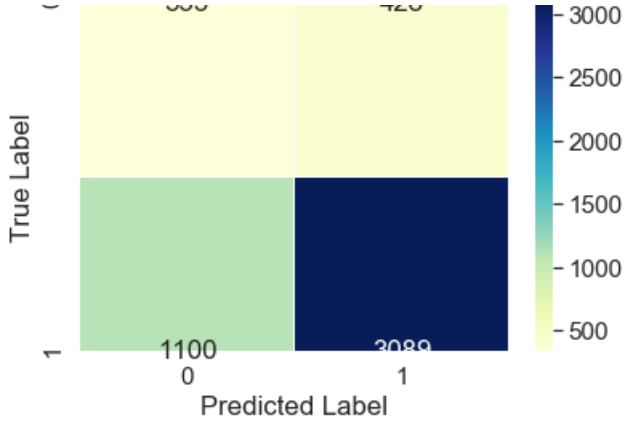


In [105]:

```
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[105]:

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



In [106]:

```
##Set - 3 AvgW2V
```

In [107]:

```
# 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 [108]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in 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%|██████████████████████████████████████████████████████████████████████████| 6733/6733
[00:04<00:00, 1350.42it/s]
```

```
6733
300
[-2.08909341e-02 -3.62653333e-02 -4.40961568e-02 -1.67855501e-01
  1.49990850e-02 -6.56487575e-02 -3.64037936e+00  2.49549548e-01
 -9.16519207e-03 -1.59060278e-01  1.23727800e-01  5.85454441e-03
 -2.72114476e-02 -1.31655211e-01 -6.14651991e-02 -1.26355792e-01
 -2.87874956e-02 -1.15285669e-01  1.06628635e-01  6.22245075e-02
  1.01554050e-02 -2.77927334e-02 -2.15055668e-02  3.45597396e-02
 -4.11972341e-02 -5.56673454e-02  7.63059079e-02 -1.44827517e-01
 -1.04791703e-01 -1.17047642e-01 -2.83189561e-01 -5.21464172e-02
  3.69465885e-02  1.91604903e-02 -8.88255097e-02  2.37741278e-03
 -4.68979925e-02 -8.64613335e-02  5.94105627e-02 -6.76618020e-02
 -4.54885524e-02  1.21511263e-01 -7.37835198e-03 -2.11553002e-01
 -6.60269863e-02 -8.91379357e-02  8.09292621e-02 -1.50878313e-01
 -6.51960652e-02  3.93465916e-02  1.17583802e-02  5.49358260e-02
 -4.33311366e-03 -2.03681322e-02  5.52851705e-02 -7.21314568e-02
  1.22069158e-01 -3.60356132e-02 -8.57569507e-02  1.03882398e-01
 -4.34531313e-02  1.13422899e-02  6.02778897e-02 -7.43509859e-02
 -2.56904515e-02  1.89435922e-01  7.38981626e-02  1.31605339e-02
  1.90215552e-01 -1.54437887e-01 -1.30988088e-01  3.82344101e-02
  2.38450435e-02 -8.71223604e-02  8.02854185e-03 -2.36042819e-01
```

```
 1.24016604e-01  4.11229383e-03  6.17987511e-02 -7.28504903e-02
 5.55341894e-02 -5.05178144e-01 -7.84429194e-02 -1.59193944e-01
-5.03591330e-02  1.45055405e-02  1.64003427e-02 -1.33833648e-01
 1.26687980e-01 -3.87802751e-02  3.35720907e-02 -4.41159414e-02
-2.47449374e-02  7.64186651e-02 -1.13925225e-02 -2.83231034e-01
-2.55143432e+00 -6.97588659e-02  1.47948733e-01  1.31513000e-01
-7.37454150e-02  5.74227736e-02  2.28799824e-01 -1.09165806e-02
-1.51772247e-04 -3.80991238e-03  1.04257208e-01 -2.02409656e-01
-7.07001789e-02  3.39719207e-02 -1.94369824e-02  8.02760762e-02
 3.32079639e-02  2.07886801e-01 -4.24974445e-02  5.36315359e-02
-5.98486219e-02  4.22183881e-03  1.12363364e-01  4.48350996e-02
-1.20117086e-01  3.22088855e-02  5.70497657e-02 -2.09652080e-01
 1.57570203e-02 -1.88591921e-02  3.66085881e-02 -5.32514304e-02
-9.97618634e-04  1.49022177e-01  6.47502959e-02  4.04258758e-02
-5.24677577e-02 -9.28816705e-02  7.97130019e-02 -1.83373330e-02
 8.90690396e-02 -4.88527129e-02  2.30817045e-01  3.31823339e-01
 1.21480361e-01  4.78014401e-02  3.02123877e-03 -6.05603630e-03
-2.29921553e-02 -6.65427540e-02  7.73519229e-02 -8.29402705e-02
 2.75559814e-01  1.05116559e-01 -3.62803740e-02 -6.07743295e-02
 7.52770540e-02 -7.43504861e-02  9.21603568e-03 -8.30203361e-02
 7.43202511e-02 -7.86099568e-03 -1.25060570e-01 -5.57166370e-02
 1.07098680e-01 -3.73130661e-02 -3.34023361e-02 -2.92256454e-02
-7.70494630e-02  3.26645229e-02 -2.67748969e-02  1.17630136e-01
 9.50065590e-02 -7.90638585e-02  1.78515665e-02 -9.08387093e-03
-8.00852244e-02 -7.45067009e-02 -1.13120352e-01  2.20451704e-01
-7.10207481e-02 -5.26146767e-02 -4.75683217e-02 -4.71844908e-02
 1.59476638e-01  1.72085514e-01 -2.46184004e-02 -7.67620432e-02
-1.85044899e-02 -1.67344996e-01 -3.45437374e-02  1.22982167e-02
 1.59586272e-01 -2.87172749e-02 -2.88939721e-02 -1.06880767e-01
-2.04094575e-02 -4.78146805e-02  2.71326167e-03 -1.32523804e-01
 6.33392894e-02  1.27754926e-01  8.87743604e-02 -3.20736600e-02
 1.26580796e-01 -2.79621930e-02  3.61642026e-03  4.40287656e-02
-1.72538885e-02  1.37026747e-01  5.08144379e-02 -4.80333740e-02
 1.55943904e-01 -3.48042101e-02  7.42036586e-02 -3.48288458e-03
-6.79341520e-02 -1.09105238e-01 -3.78702667e-02 -2.91654053e-03
-9.01300678e-02 -1.05322070e-01 -5.20481176e-02  2.67454458e-02
-1.32490599e-01 -6.69233207e-02 -5.66246313e-02 -2.40108811e-03
-2.54895348e+00  1.24997326e-01 -8.38635366e-02 -4.07649507e-02
 1.50287722e-02 -9.30988502e-02  9.46978189e-02  4.05110938e-02
-1.40330480e-02 -5.96189546e-02 -8.60456652e-02  1.33197600e-01
 1.16551000e-02 -4.86569520e-02 -6.60758943e-02  2.71332471e-02
-1.57194377e-01  4.83092700e-02 -1.91174079e-01  5.22561295e-02
-2.03521784e-02 -1.63770194e-02 -4.95770797e-02 -8.02897650e-02
-1.12718138e-01  5.76994070e-02 -3.94241907e-02 -2.20773642e-02
 3.40348586e-02 -3.05507328e-02  6.34580427e-02 -2.57880863e-02
 6.52916770e-02 -8.48456419e-02  7.31689093e-02 -3.15741048e-02
 7.57895597e-02 -5.36126546e-02 -2.47804833e-02 -1.80766132e-02
 6.31772590e-02 -1.02377659e-01 -2.38019686e-01 -4.39680000e-02
 1.15006784e-01  4.99308255e-02  2.58643749e-02 -9.99853634e-02
-1.78938802e-01  6.72197593e-02  1.88873700e-03  8.48855361e-02
 1.11123383e-01  6.27836203e-02 -1.71334894e-02 -2.67726225e-02
 3.00819127e-01 -1.66613656e-03  2.58666300e-02  1.31811101e-01
 5.11970744e-02  1.18666762e-01  3.24481264e-02  6.05228868e-02
 1.02800700e-02 -2.60828123e-02 -5.73169780e-03 -1.08446826e-01
-8.66416185e-02 -2.47103890e-02 -1.21762150e-02  4.17373423e-02
-4.71182938e-02 -2.56189604e-03  1.01719160e-01  2.12146537e-02]
```

In [109]:

```
avg_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this 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%|████████████████████████████████████████████████████████████████████████████| 3317/3317
[00:01<00:00, 1661.09it/s]
```

In [110]:

```
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%|████████████████████████████████████████████████████████████████| 4950/4950
[00:03<00:00, 1621.04it/s]
```

In [111]:

```
# 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%|████████████████████████████████████████████████████████████████| 6733/6733
[00:00<00:00, 39862.92it/s]
```

```
6733
300
[-2.29374800e-01 -1.73390000e-01  1.49640000e-02 -1.69958000e-01
 -6.70598000e-02  2.34861600e-01 -2.95358000e+00  7.53166000e-02
  4.07878000e-01  3.28878000e-01  1.12256600e-01  5.22040000e-02
 -1.57486000e-01 -1.02534400e-01 -3.88868400e-01  1.92471400e-01
  9.55344000e-02 -3.00078000e-01  6.32786000e-02 -7.03348000e-02
  1.38425000e-01  1.77280000e-02  2.57912000e-02 -5.66920000e-02
 -1.12539400e-01 -1.06442600e-01 -1.90940000e-03  1.00672800e-01
  1.60850000e-01 -1.92308200e-01 -2.07721800e-01 -1.62560580e-01
  6.01689400e-02  3.41140000e-02  1.80916000e-01  8.83764000e-02
  7.52620000e-02  8.72801600e-02 -4.96130000e-02 -2.93724120e-01
 -1.14031800e-01 -1.84920000e-02  1.68043400e-01 -5.61726000e-02
 -1.33271660e-01  7.87280000e-03 -1.44443400e-02  7.01510000e-02
 -5.81788000e-02 -7.39224000e-02  2.87471060e-02  7.03054000e-02
  2.25583400e-01  1.17396000e-01  2.30717800e-01  9.80632000e-02
  3.30088000e-01 -6.76340000e-02 -3.18952600e-01  3.42676000e-01
 -1.67716920e-01  1.68256783e-02 -1.92532000e-01  8.55469800e-02
  1.18978000e-01  4.09258600e-01  6.48536000e-02 -2.03975180e-01
  4.89642600e-02 -6.54312000e-02 -1.97928000e-01 -9.61600000e-02
  8.26792000e-02 -1.35530000e-01 -3.44716200e-02 -3.17211200e-01
  2.02870600e-01 -2.15452800e-02 -3.48284000e-02 -1.37214000e-02
  1.32642400e-01 -7.12799800e-01  9.15760000e-02  2.39961400e-01
 -2.91567400e-01 -1.10328600e-01  2.66734000e-02 -1.70576380e-01
 -6.39772000e-02  2.73204000e-02  2.51159000e-01 -1.76048800e-01
 -1.62974000e-01  2.97220000e-02  1.08399600e-01 -6.26094000e-01
 -2.66100000e+00 -1.09401800e-01 -6.27242000e-02  1.66420000e-02
 -2.74568000e-01  9.44494000e-02  2.73643200e-01  3.64006000e-02
  3.11988200e-01 -1.18150800e-01  2.63011400e-01 -3.34881200e-01
  2.81840000e-02  1.00373600e-01 -1.34625400e-01  1.85108000e-02
  2.51560000e-02  1.22270000e-01 -2.31822600e-01 -6.75702000e-02
 -2.43460000e-02  5.95520000e-03 -2.41226940e-01  8.13178000e-02
 -1.31292000e-02  1.58424000e-01 -6.47300000e-03 -2.02483200e-01
  1.03940800e-01  5.08880000e-02  1.36835600e-01  1.05761000e-01
 -9.63692000e-02 -9.48980000e-02 -5.33240000e-02 -1.82972000e-02
```

```
 8.37604000e-02 -6.56675000e-02  1.30457800e-01 -4.07382000e-01
 4.33480000e-02  9.14378800e-02  9.35890000e-02  4.42244000e-01
-2.71596000e-01  2.36677600e-01 -1.39914000e-02  8.68416000e-02
 1.20794000e-01  4.44060000e-02 -1.07914600e-01 -1.79276000e-01
 3.55862000e-01  1.70634000e-01 -1.47139910e-01  1.50091800e-01
 8.32922000e-02 -4.70488000e-02 -2.91525400e-01  2.42040800e-01
 7.02869400e-02 -7.21069400e-02  3.76487760e-01  2.96614000e-01
-3.59857020e-01  6.62460000e-03  7.38886000e-02 -1.40533280e-01
-1.24669000e-01  2.69800000e-02  1.51574000e-01  1.62894840e-01
 1.19574345e-01 -5.65338000e-02 -1.02112000e-01  1.30480600e-01
 1.11989400e-01  6.11024000e-02 -1.31824400e-01 -1.02310000e-02
-4.11500000e-02  3.25092200e-01 -2.38131800e-01 -1.30200000e-04
-2.87208140e-01  2.89748800e-01  8.16754000e-02  1.75153800e-01
 3.13454000e-02  5.76656000e-02 -1.15660000e-02 -1.78278000e-01
 4.26820000e-01  2.85819340e-01  8.18108600e-02  2.77612000e-02
-1.50020000e-02 -1.57964800e-01 -7.94550000e-02 -1.14097200e-01
-1.34032000e-01  5.04320000e-02 -9.46994000e-02  1.88982000e-01
 1.15663200e-01 -1.60090200e-01 -4.75864000e-01  6.81460000e-02
-1.42932800e-01 -1.39878000e-01  2.62616000e-01  1.58035600e-01
 1.09396000e-02 -9.32394000e-02 -4.20742000e-01 -1.71170000e-02
 6.17840000e-03 -2.70445400e-01 -1.81683200e-01  3.63200000e-02
-3.24684000e-02 -4.43272000e-01 -2.10112000e-01 -1.34554000e-01
-2.08133800e-01  4.92986000e-02 -4.67012000e-02  8.58892000e-02
-1.91182000e+00  4.09584000e-01  1.63580400e-01  1.32700000e-01
 6.40786000e-02 -1.23938200e-01  2.13654000e-01 -3.21308600e-01
 6.27854000e-02  1.85100400e-01 -5.60280000e-02  1.18194400e-01
 2.49050000e-01 -1.91936000e-01 -5.89228000e-02  9.83124000e-02
 1.30711400e-01  4.77120000e-02 -2.80574000e-01 -3.31434600e-01
 7.01298000e-02  4.40938000e-02  6.64728000e-02 -1.55752000e-02
 1.37310000e-01 -1.17864200e-01 -2.40659000e-01  6.07140000e-02
 1.13441100e-01  1.62245400e-01  1.89956000e-01  7.46744000e-02
 2.11846000e-01 -5.53234000e-02  3.06294200e-01  7.34666000e-02
 4.88742800e-02  1.25710600e-01 -1.81557800e-01  1.19910000e-01
 8.23000000e-05 -1.93403800e-01  1.24430600e-01  1.01398000e-02
 2.30205400e-02  1.02994200e-01 -2.62538000e-02 -2.66064300e-01
 1.98282800e-01 -1.14002000e-01  5.92568000e-02 -9.54880000e-03
 1.49004000e-01 -1.58280200e-01 -3.51182000e-02  1.17482800e-01
 1.19810380e-01  4.93560000e-02 -1.52741400e-01 -5.67520000e-03
-2.99808000e-01  1.14299400e-01  3.17157400e-01 -5.88148000e-02
 5.97684000e-02 -8.64460000e-02 -4.00260000e-03 -2.75662800e-02
-6.40280000e-02  9.05262000e-02  1.54573000e-01 -5.45702000e-02
 1.52862000e-02  9.28382000e-02  2.77801800e-01  6.14936000e-02]
```

In [112]:

```
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%|████████████████████████████████████████████████████████████████████████████| 3317/3317
[00:00<00:00, 46744.46it/s]
```

In [113]:

```
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%|████████████████████████████████████████████████████████████████████████| 4950/4950
[00:00<00:00, 40266.65it/s]
```

In [114]:

```
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()
```

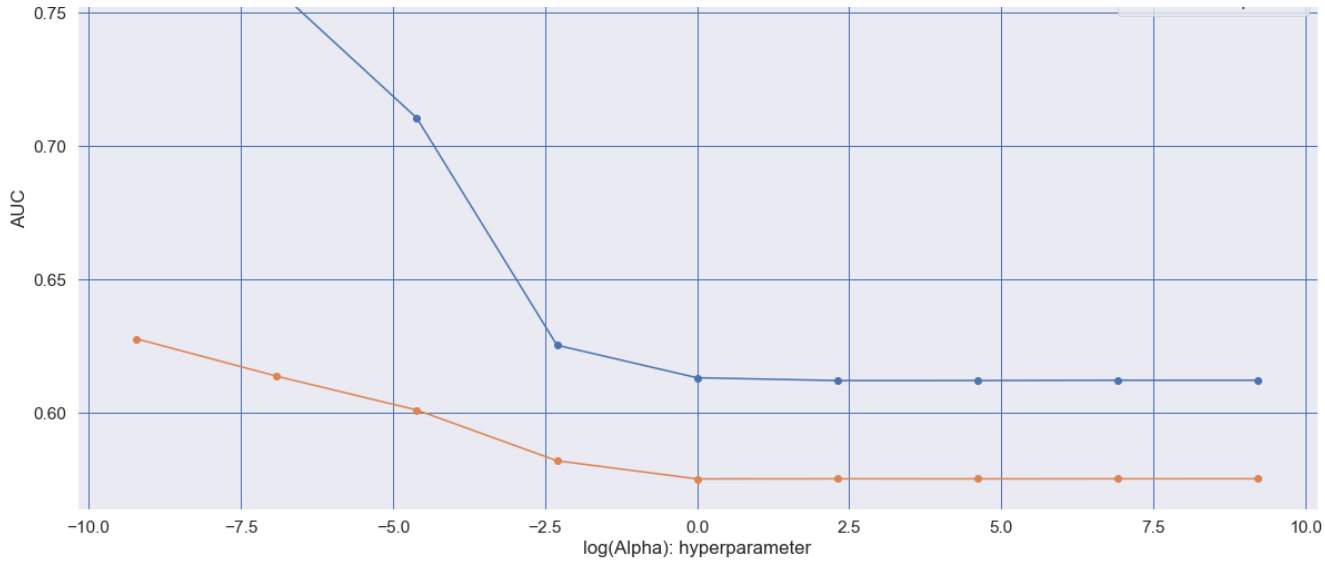
In [115]:

```
# 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='balan
ced', 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 gr
eater 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 posi
tive 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,  1.32it/s]
```

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()
```

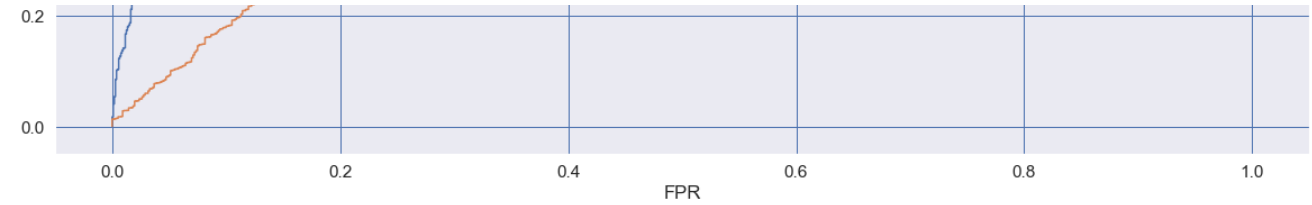


In [117]:

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

from sklearn.metrics import roc_curve, auc


base_estimator_svm_output_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 [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()
```



In [119]:

```
# 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)
```

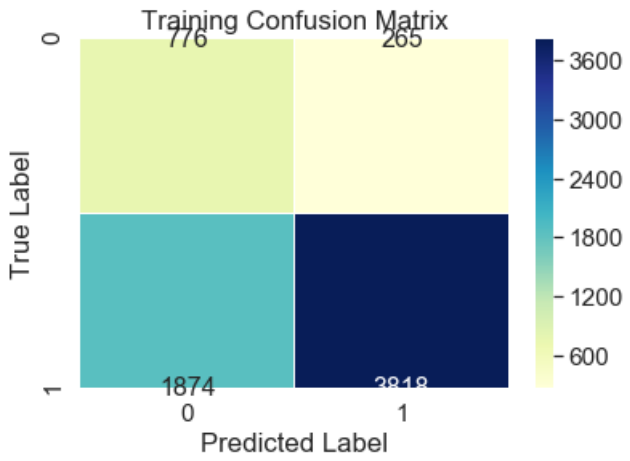
```
====================================================================================================

the maximum value of tpr*(1-fpr) 0.5000138387935813 for threshold 0.838
Train confusion matrix
[[ 776  265]
 [1874 3818]]
Test confusion matrix
[[ 439  322]
 [1507 2682]]
```

In [120]:

```
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[120]:

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

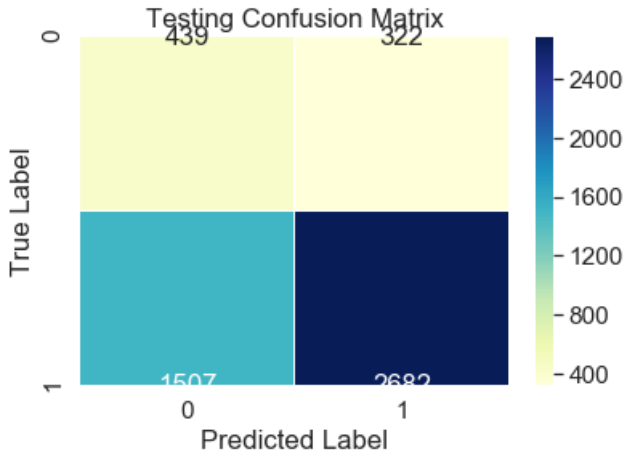


In [121]:

```
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[121]:

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



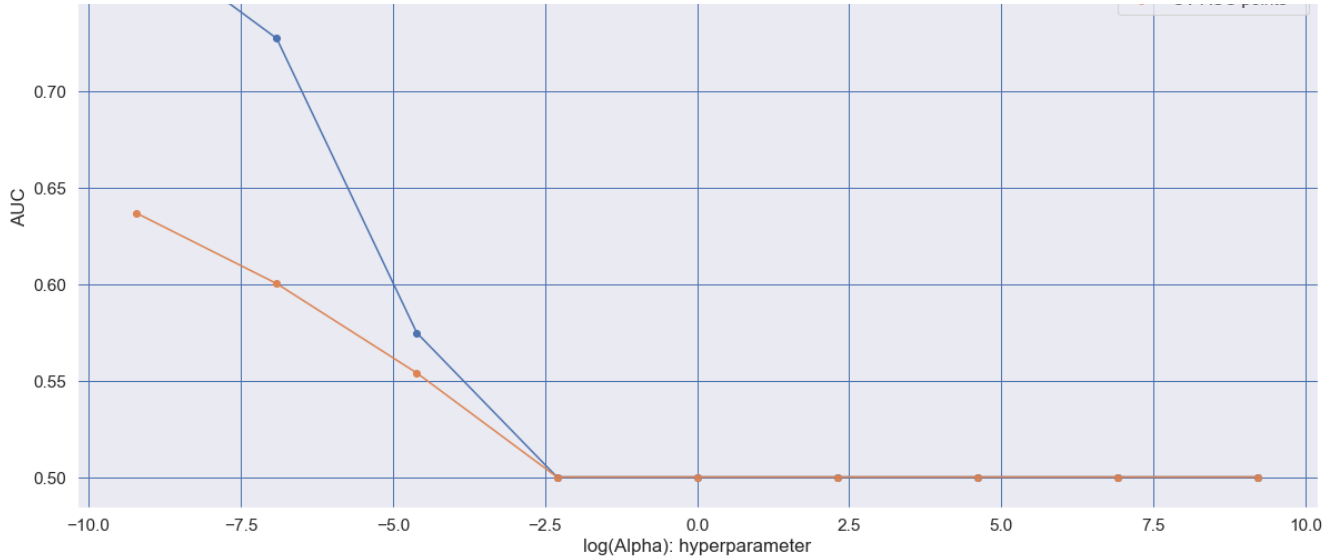
In [122]:

```
# 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='balan
ced', 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 gr
eater 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 posi
tive 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
:18<00:00,  2.02s/it]
```

In [123]:

```
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()
```

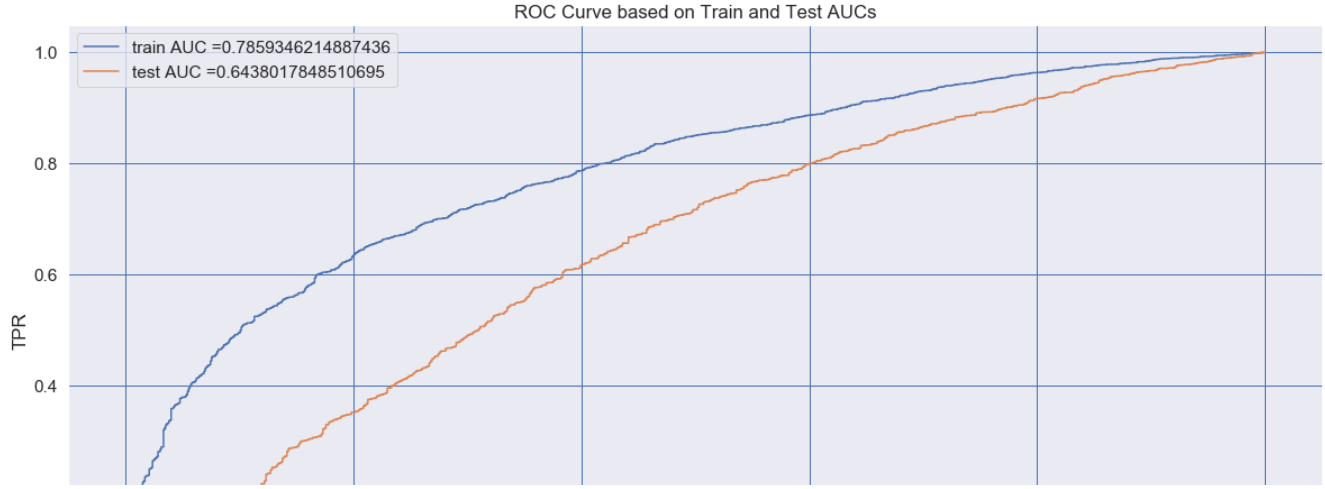


In [124]:

```
# 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',
class_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 estimates of the positive
class
# not the predicted outputs

y_train_pred = svm_output_tfidf.predict_proba(X_tr)[:,1]  # returning probability 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 [125]:

```
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()
```



In [126]:

```
# Set - 4 TFIDF W2V
```

In [127]:

```
# 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 [128]:

```
# 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 tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_train.append(vector)

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

```
100%|██████████████████████████████████████████████████████████████████████████| 6733/6733
[00:39<00:00, 169.08it/s]
```

```
6733
300
```

In [129]:

```
# 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 tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
```

```
    tfidf_w2v_vectors_cv.append(vector)

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

```
100%|████████████████████████████████████████████████████████████████| 3317/3317
[00:19<00:00, 166.57it/s]
```

```
3317
300
```

In [130]:

```
# 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/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_test.append(vector)

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

```
100%|████████████████████████████████████████████████████████████████| 4950/4950
[00:28<00:00, 172.44it/s]
```

```
4950
300
```

In [131]:

```
# 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 [132]:

```
# 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 li
st
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/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_pj_title_train.append(vector)
```

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

```
100%|████████████████████████████████████████████████████████████████████████████| 6733/6733
[00:00<00:00, 19733.19it/s]
```

```
6733
300
```

In [133]:

```
# 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 tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_pj_title_cv.append(vector)

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

```
100%|████████████████████████████████████████████████████████████████████████████| 3317/3317
[00:00<00:00, 16762.14it/s]
```

```
3317
300
```

In [134]:

```
# 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 lis
t
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 tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_pj_title_test.append(vector)

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

```
100%|████████████████████████████████████████████████████████████████████████████| 4950/4950
[00:00<00:00, 19891.00it/s]
```

```
4950
300
```

In [135]:

```
# 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)).tocsr()

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()
```

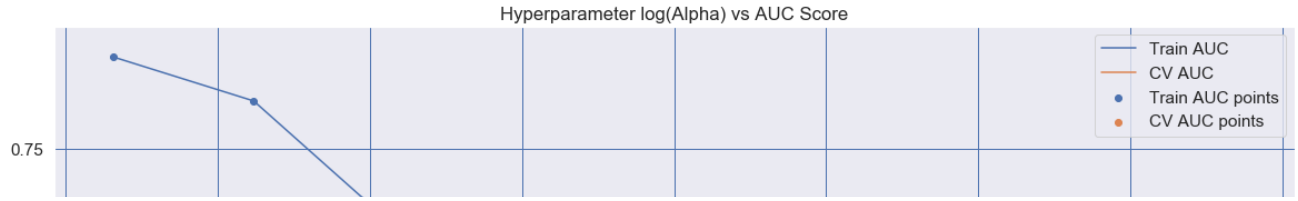
In [136]:

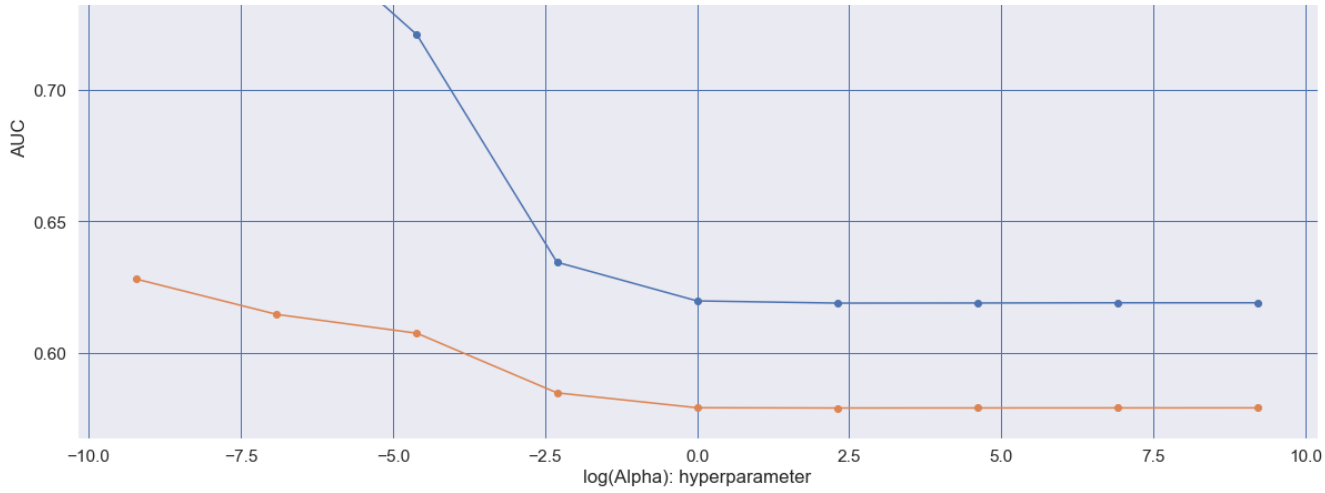
```
# SVM With L2 Penality
# L2 regulaization is default penality

# 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='balan
ced', 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 gr
eater 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 posi
tive 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,  1.42it/s]
```

In [137]:

```
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()
```



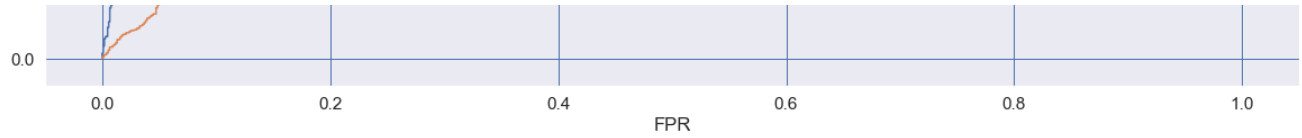


In [138]:

```
# 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', 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 [139]:

```
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()
```



In [140]:

```
# 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)
```

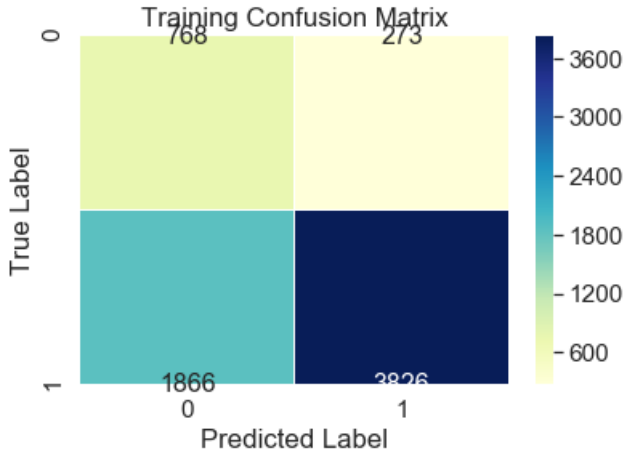
```
====================================================================================================

the maximum value of tpr*(1-fpr) 0.4958959538742884 for threshold 0.839
Train confusion matrix
[[ 768  273]
 [1866 3826]]
Test confusion matrix
[[ 429  332]
 [1514 2675]]
```

In [141]:

```
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[141]:

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

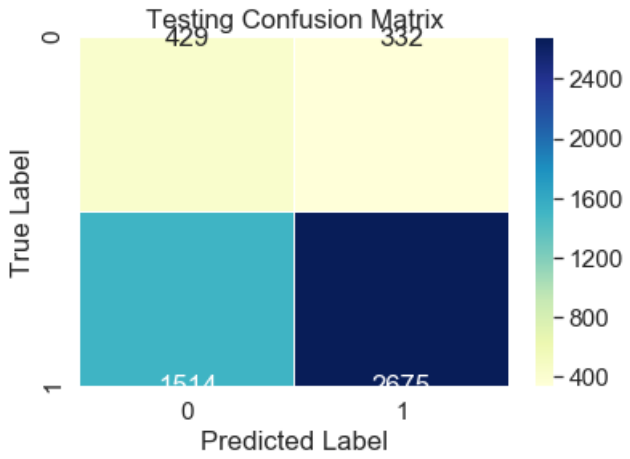


In [142]:

```
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[142]:

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



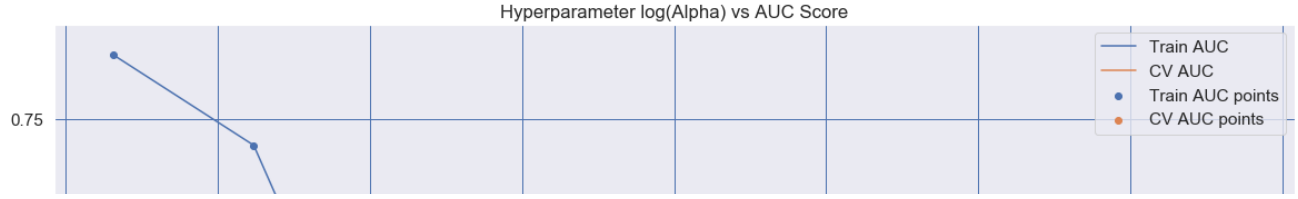
In [143]:

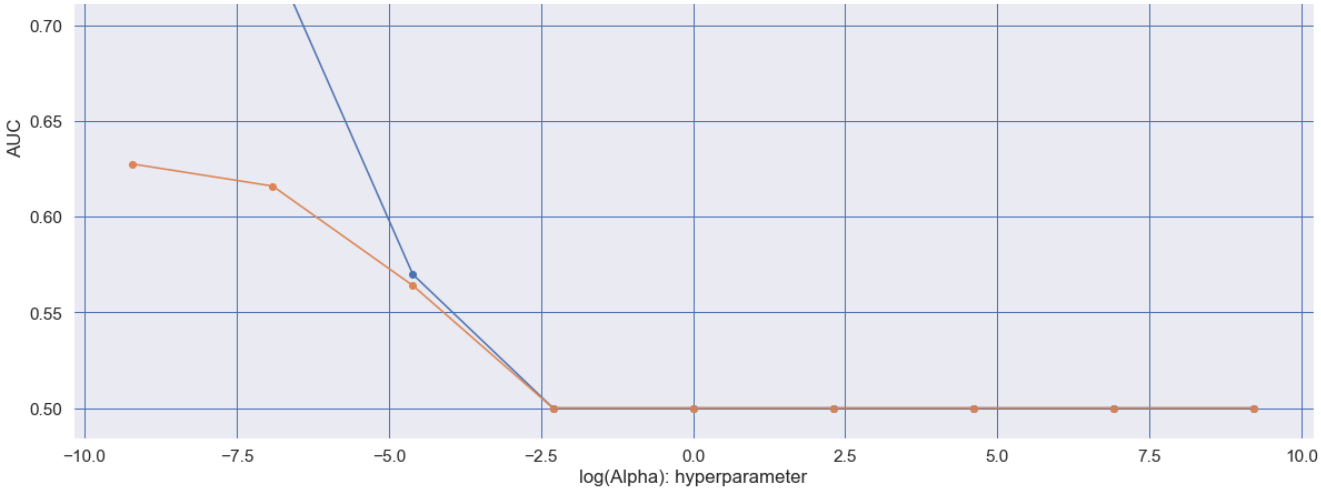
```
# SVM With L1 Penality
# L2 regulaization is default penality

# 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='balan
ced', 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 gr
eater 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 posi
tive 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
:17<00:00,  1.91s/it]
```

In [144]:

```
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 [145]:

```
# 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',
class_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 estimates of the positive
class
# not the predicted outputs

y_train_pred = svm_output_tfidf.predict_proba(X_tr)[:,1]  # returning probability 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 [146]:

```
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()
```

In [147]:

```
# 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)
```

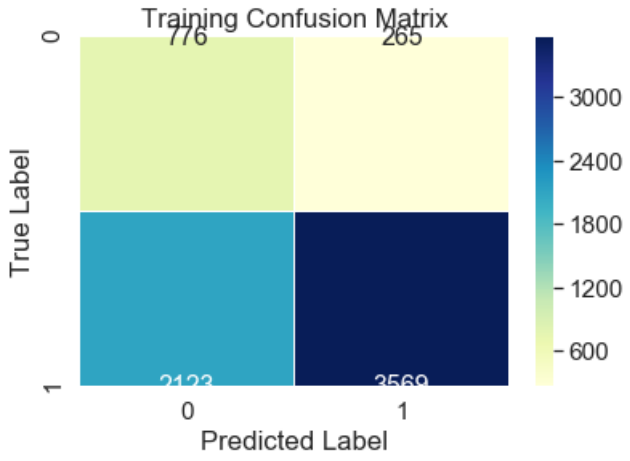
```
====================================================================================================

the maximum value of tpr*(1-fpr) 0.4674042406113912 for threshold 0.842
Train confusion matrix
[[ 776  265]
 [2123 3569]]
Test confusion matrix
[[ 439  322]
 [1677 2512]]
```

In [148]:

```
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[148]:

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

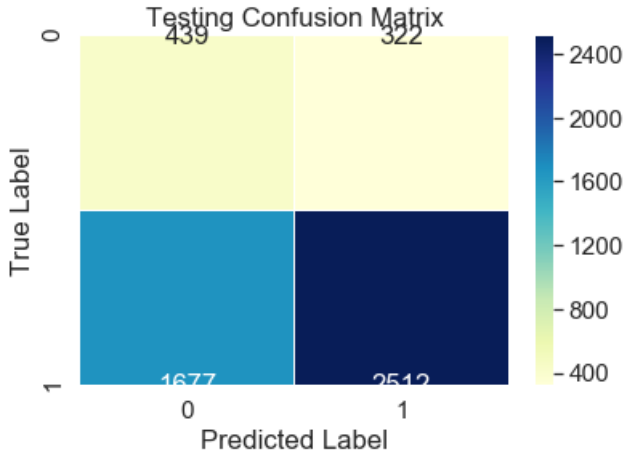


In [149]:

```
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[149]:

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



2.5 Support Vector Machines with added Features `Set 5`

In [150]:

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

In [151]:

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

Out[151]:

```
Normalizer(copy=True, norm='l2')
```

In [152]:

```
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 [153]:

```
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 [154]:

```
def return_pj_title_word_count(_string):
    return len(_string.strip().split(" "))
```

In [155]:

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

Out[155]:

```
[5, 4, 8]
```

In [156]:

```
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 [157]:

```
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'].values.reshape(1,-1))
X_cv_pj_words_norm = normalizer.transform(X_cv['pj_title_word_count'].values.reshape(1,-1))
X_test_pj_words_norm = normalizer.transform(X_test['pj_title_word_count'].values.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 [158]:

```
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 [159]:

```
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 [160]:

```
sid = SentimentIntensityAnalyzer()
```

In [161]:

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

In [162]:

```
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%|██████████████████████████████████████████████████████████████████████████████| 6733/6733
[00:23<00:00, 281.92it/s]
```

In [163]:

```
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%|██████████████████████████████████████████████████████████████████████████| 3317/3317
[00:12<00:00, 273.96it/s]
```

In [164]:

```
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%|██████████████████████████████████████████████████████████████████████████| 4950/4950
[00:17<00:00, 288.06it/s]
```

In [165]:

```
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.reshape(1,-1))
X_test_ess_neg_norm = normalizer.transform(X_test['essay_scores_neg'].values.reshape(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 [166]:

```
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.reshape(1,-1))
X_test_ess_neu_norm = normalizer.transform(X_test['essay_scores_neu'].values.reshape(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 [167]:

```
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 [168]:

```
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 [169]:

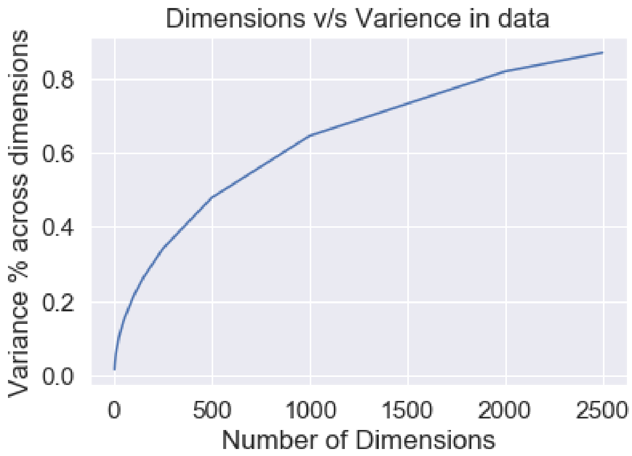
```
from sklearn.decomposition import TruncatedSVD
from scipy.sparse import random as sparse_random
from sklearn.random_projection import sparse_random_matrix

# applying trucated SVD over n number of values to find best number of components
```

```
# reference - https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.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
[03:21<00:00, 16.82s/it]
```

In [170]:

```
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()
```



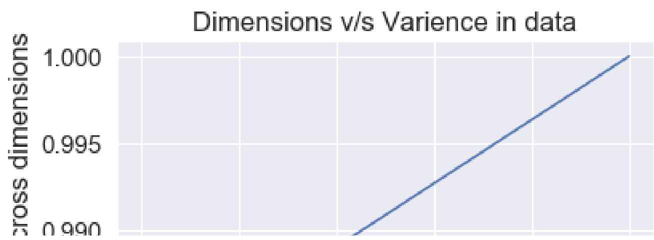
In [172]:

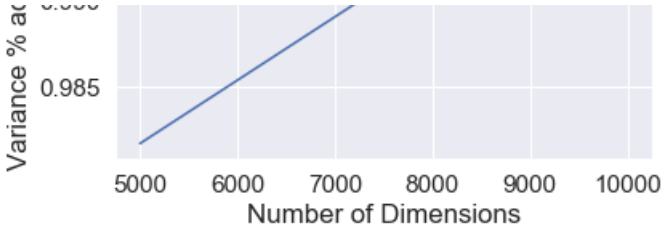
```
# Lets add 5000 also to number of components
_n_comps2 = [5000,10000]
_varience_sum2 = []
for i in tqdm(_n_comps2):
    svd = TruncatedSVD(n_components = i,random_state=42)
    svd.fit(X_train_essay_tfidf)
    _varience_sum2.append(svd.explained_variance_ratio_.sum())
```

```
  0%|                                                        |
[00:00<?, ?it/s]
 50%|████████████████████                     | 1/2 [05:0
<05:01, 301.84s/it]
100%|████████████████████████████████████████| 2/2 [14:
32<00:00, 436.04s/it]
```

In [173]:

```
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()
```





In [174]:

```
# concatenating all data
X_tr = hstack((X_train_quantity_norm, X_train_pj_words_norm, X_train_essay_words_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()
```

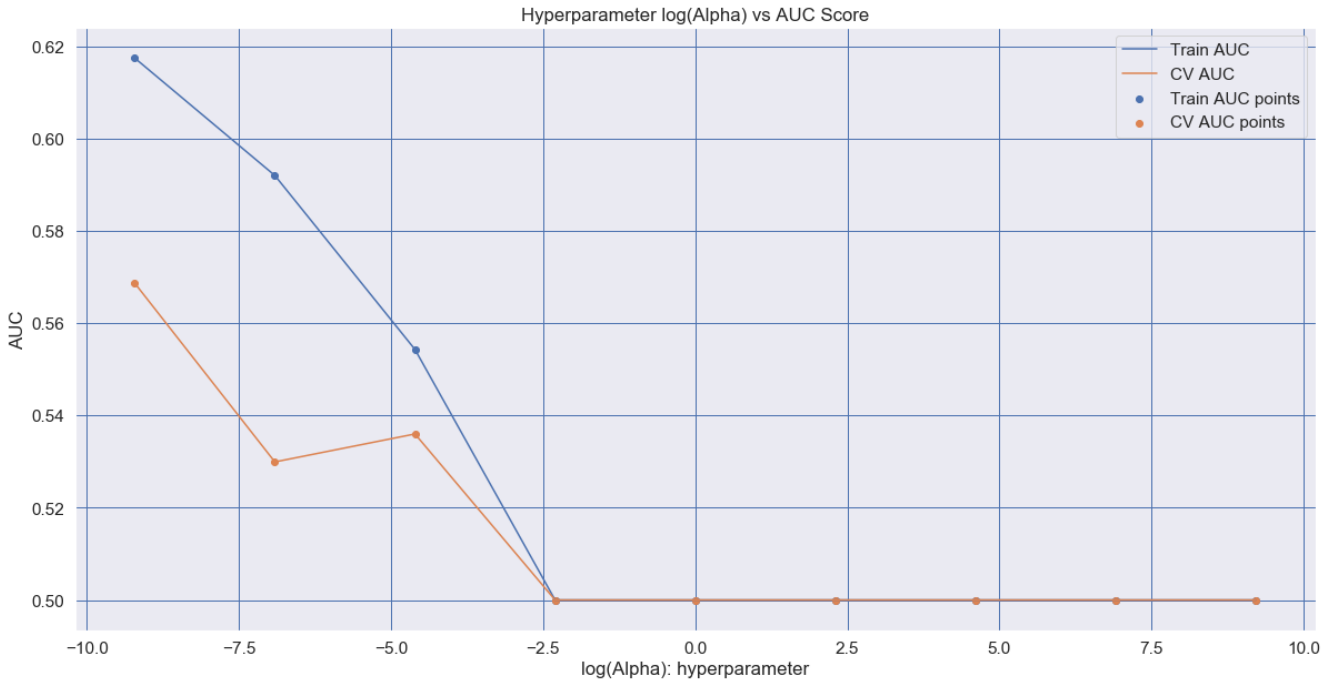
In [175]:

```
# 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='balan
ced', 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 gr
eater 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 posi
tive 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))
```

```
  0%|                                                                                           |
[00:00<?, ?it/s]
 11%|████████                                                                  | 1/9 [00:
00:07,  1.14it/s]
 22%|████████████████                                                          | 2/9
[00:01<00:04,  1.47it/s]
 44%|████████████████████████████████                                          | 4/9
[00:01<00:02,  2.03it/s]
 67%|████████████████████████████████████████████████                          | 6/9
[00:01<00:01,  2.77it/s]
100%|████████████████████████████████████████████████████████████████████████| 9/9 [00
:01<00:00,  5.96it/s]
```

In [176]:

```
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()
```



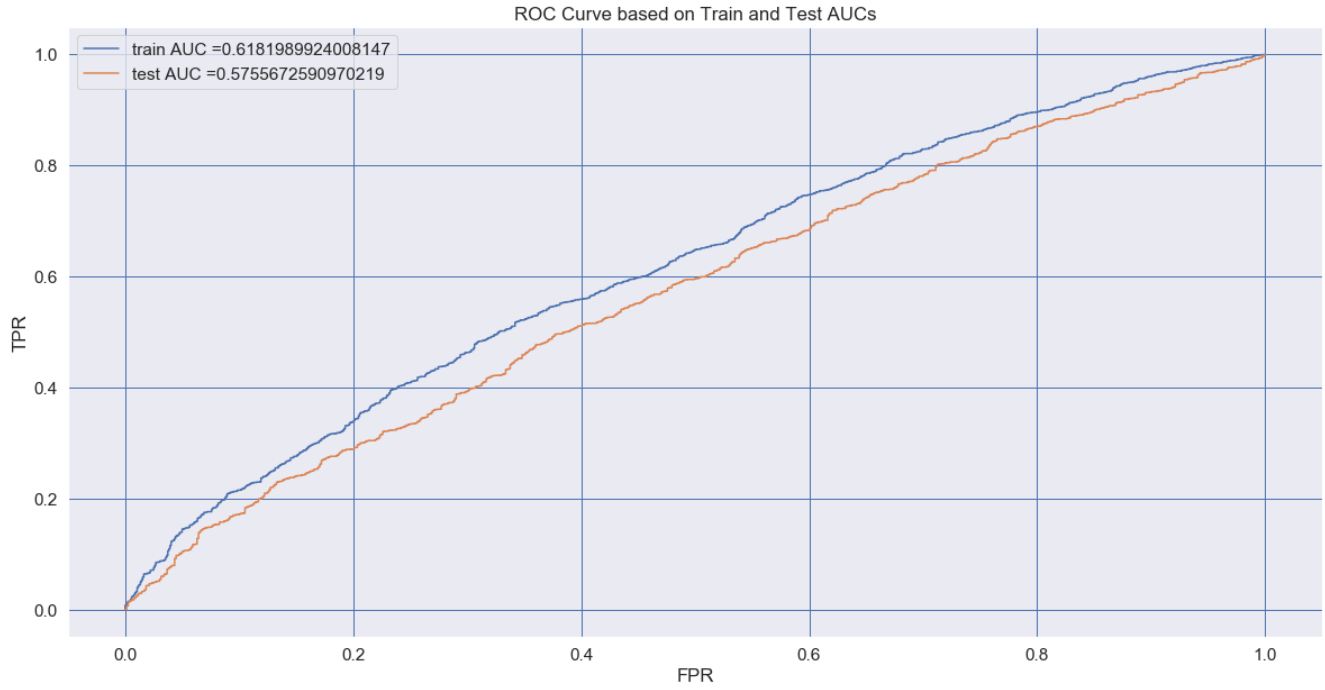
In [177]:

```
# 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',
class_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 estimates of the positive
class
# not the predicted outputs

y_train_pred = svm_output_tfidf.predict_proba(X_tr)[:,1]  # returning probability 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 [178]:

```
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()
```



In [179]:

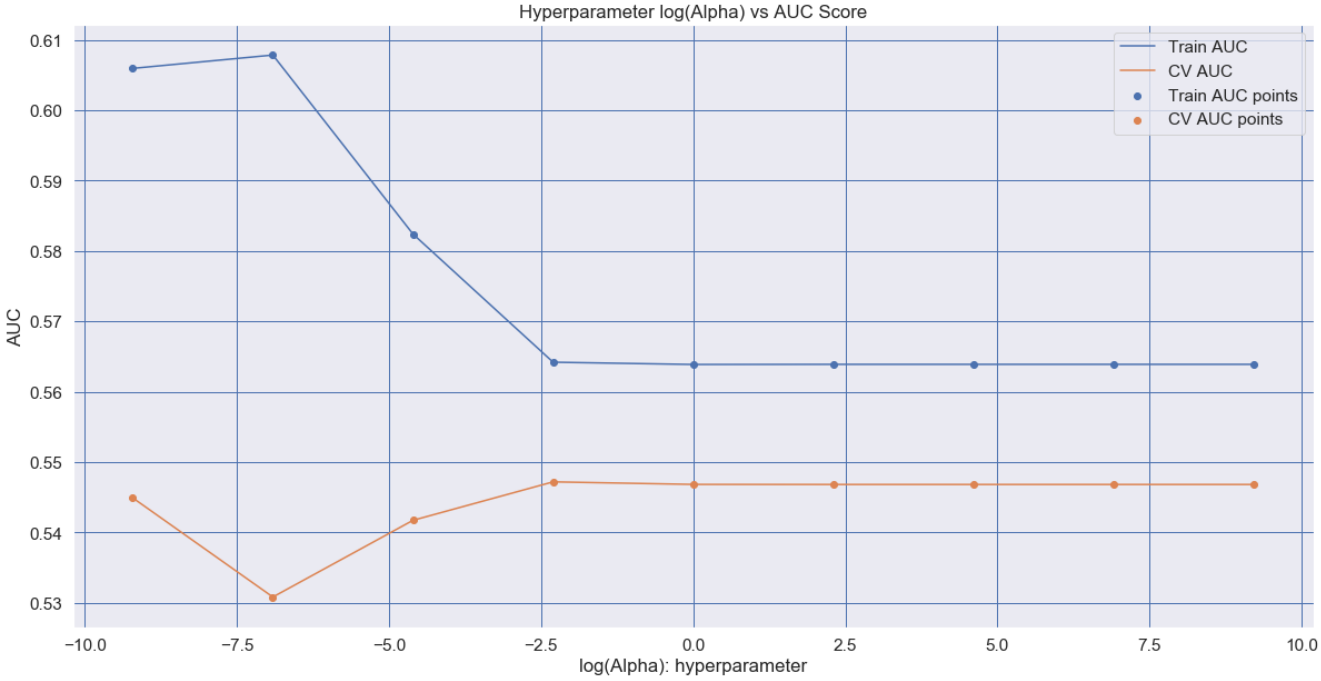
```
# with L2 Regularization for set 5
train_auc = []
cv_auc = []
alpha = [10**-4, 10**-3,10**-2,10**-1,1,10,10**2,10**3,10**4]#alpha=1/C
for i in tqdm(alpha):
    base_estimator_svm_output_tfidf = SGDClassifier(loss="hinge", penalty='l2', class_weight='balan
ced', alpha=i)
    # Since SGDClassifier 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 gr
eater 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 posi
tive 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))
```

```
  0%|                                                                                           |
[00:00<?, ?it/s]
 22%|████████████████▍                                                          | 2/9
[00:00<00:00, 12.13it/s]
 44%|████████████████████████████████▊                                          | 4/9
[00:00<00:00, 13.53it/s]
 78%|████████████████████████████████████████████████████████▏                  | 7/9 [00:
00<00:00, 15.51it/s]
100%|████████████████████████████████████████████████████████████████████████| 9/9 [00
:00<00:00, 16.90it/s]
```

In [180]:

```
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()
```

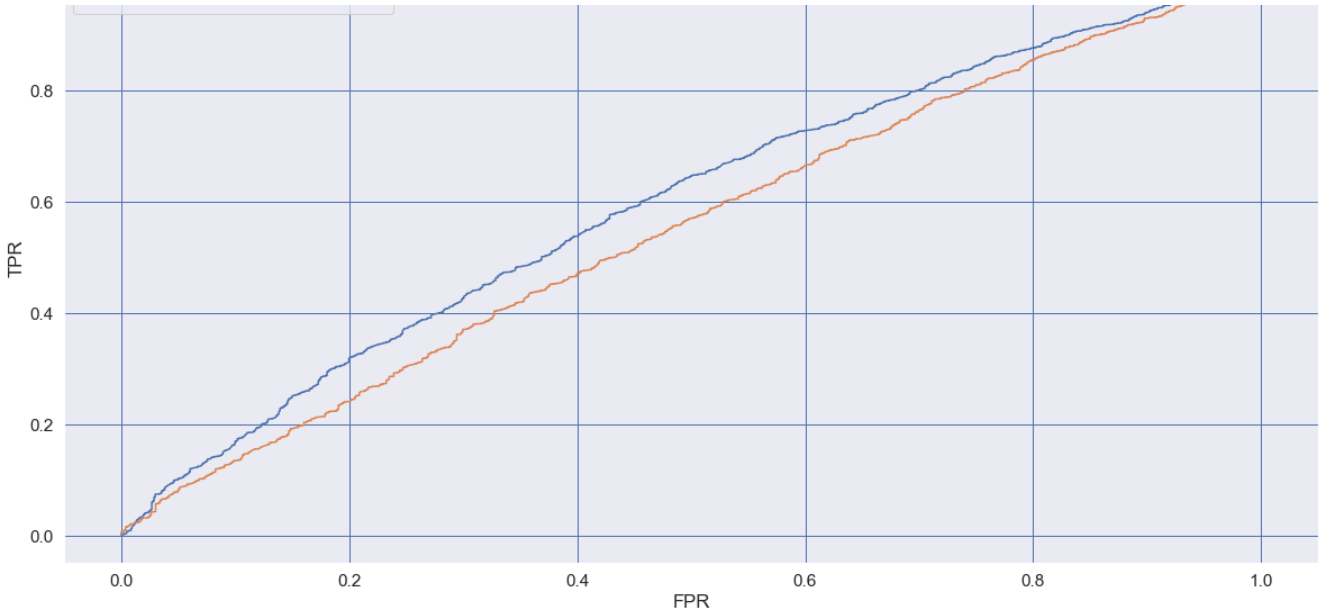


In [181]:

```
# 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',
class_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 estimates of the positive
class
# not the predicted outputs

y_train_pred = svm_output_tfidf.predict_proba(X_tr)[:,1]  # returning probability 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 [182]:

```
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()
```



In [183]:

```
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))
```

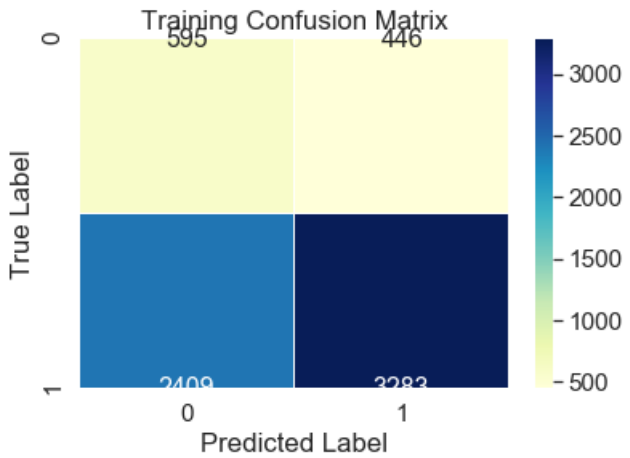
```
the maximum value of tpr*(1-fpr) 0.32966453414232894 for threshold 0.845
```

In [184]:

```
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[184]:

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

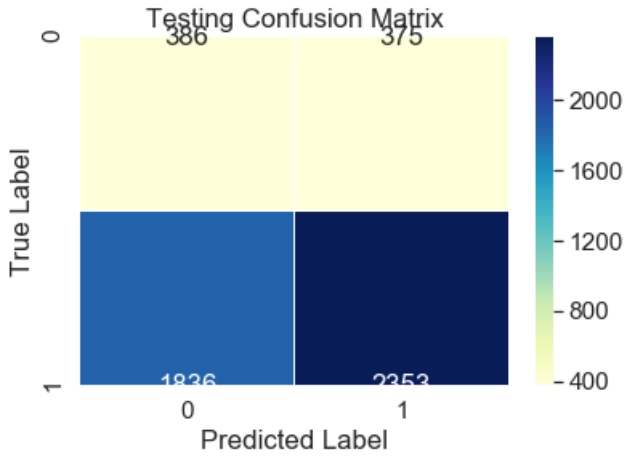


In [185]:

```
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[185]:

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



In [187]:

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

3. Conclusion

In [188]:

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

In [189]:

```
# 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   |
+------------+----------------+-------------------------+-----------+----------+
```