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`

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

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

- __project_essay_1:__ "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."

your neighborhood, and your school are all helpful.

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

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

In [1]:

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

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

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

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

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

from tqdm import tqdm
import os

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

os.chdir('C:/Users/kingsubham27091995/Desktop/AppliedAiCouse/DonorsChoose')
```

1.1 Reading Data

In [2]:

```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

In [3]:

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

```
('Number of data points in train data', (109248, 17))
--------------------------------------------------
('The attributes of data :', array(['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'], dtype=object))
```

In [4]:

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

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

Out[4]:

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

Preprocessing of Project_Grade_Category

In [6]:

```
project_grade_category = []

for i in range(len(project_data)):
    a = project_data["project_grade_category"][i].replace(" ", "_")
    project_grade_category.append(a)
```

In [7]:

```
project_grade_category[0:5]
```

Out[7]:

```
['Grades_PreK-2', 'Grades_6-8', 'Grades_6-8', 'Grades_PreK-2', 'Grades_PreK-2']
```

In [8]:

```
project_data.drop(['project_grade_category'], axis=1, inplace=True)
```

In [9]:

```
project_data["project_grade_category"] = project_grade_category
```

In [10]:

```
project_data.head(5)
```

Out[10]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_subject_ca
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Math & Science
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Special Needs

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_subject_cat
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	2016-04-27 00:46:53	Literacy & Language
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Applied Learning
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Literacy & Language

Preprocessing of `project_subject_categories`

In [11]:

```
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]))
```

Preprocessing of `project_subject_subcategories`

In [12]:

```
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"
```

```
    # 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]))
```

Dealing with text int he data (TEXT PREPROCESSING)

Cleaning the Titles

In [13]:

```
# 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 [14]:

```
# 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 [15]:

```
clean_titles = []

for titles in tqdm(project_data["project_title"]):
    title = decontracted(titles)
    title = title.replace('\\r', ' ')
    title = title.replace('\\"', ' ')
    title = title.replace('\\n', ' ')
    title = re.sub('[^A-Za-z0-9]+', ' ', title)
    title = ' '.join(f for f in title.split() if f not in stopwords)
    clean_titles.append(title.lower().strip())
```

```
100%|██████████| 109248/109248 [00:04<00:00, 25251.42it/s]
```

In [16]:

```
project_data["clean_titles"] = clean_titles
```

In [17]:

```
project_data.drop(['project_title'], axis=1, inplace=True)
```

Finding number of words in title and introducing it in a new column

- This can be used as Numerical Feature for Vectorisation

In [18]:

```
title_word_count = []
```

In [19]:

```
for a in project_data["clean_titles"] :
    b = len(a.split())
    title_word_count.append(b)
```

In [20]:

```
project_data["title_word_count"] = title_word_count
```

In [21]:

```
project_data.head(5)
```

Out[21]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_essay_1	p
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	I have been fortunate enough to use the Fairy ...	M c v b
						2016-	Imagine being 8-9 years old	M

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_essay_1	
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	04-27 00:31:25	You're in your th...	
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	2016-04-27 00:46:53	Having a class of 24 students comes with diver...	I I tw ki st
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	I recently read an article about giving studen...	I t in sc
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	My students crave challenge, they eat obstacle...	W pu el sc

Combine 4 Project essays into 1 Essay

In [22]:

```
# 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)
```

Cleaning the Essays

In [23]:

```
clean_essay = []

for ess in tqdm(project_data["essay"]):
    ess = decontracted(ess)
    ess = ess.replace('\\r', ' ')
    ess = ess.replace('\\"', ' ')
    ess = ess.replace('\\n', ' ')
    ess = re.sub('[^A-Za-z0-9]+', ' ', ess)
    ess = ' '.join(f for f in ess.split() if f not in stopwords)
    clean_essay.append(ess.lower().strip())
```

```
100%|██████████| 109248/109248 [01:03<00:00, 1731.77it/s]
```

In [24]:

```
project_data["clean_essays"] = clean_essay
```

In [25]:

```
project_data.drop(['essay'], axis=1, inplace=True)
```

Finding number of words in essays and introducing it in a new column

- This can be used as Numerical Feature for Vectorisation

In [26]:

```
essay_word_count = []
```

In [27]:

```
for ess in project_data["clean_essays"] :
    c = len(ess.split())
    essay_word_count.append(c)
```

In [28]:

```
project_data["essay_word_count"] = essay_word_count
```

In [29]:

```
project_data.head(5)
```

Out[29]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_essay_1	pr
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	I have been fortunate enough to use the Fairy ...	M co va ba
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Imagine being 8-9 years old. You're in your th...	M st au ar
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	2016-04-27 00:46:53	Having a class of 24 students comes with diver...	I l tw ki st
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	I recently read an article about giving studen...	I t in so
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	My students crave challenge, they eat obstacle...	W pu el so

Calculate Sentiment Scores for the Essays Feature

In [30]:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

In [ ]:

```
nltk.download('vader_lexicon')
```

In [31]:

```
analyser = SentimentIntensityAnalyzer()
```

In [32]:

```
## https://medium.com/analytics-vidhya/simplifying-social-media-sentiment-analysis-using-vader-in-python-f9e6ec6fc52f
neg = []
pos = []
neu = []
```

```
compound = []

for a in tqdm(project_data["clean_essays"]) :
    b = analyser.polarity_scores(a)['neg']
    c = analyser.polarity_scores(a)['pos']
    d = analyser.polarity_scores(a)['neu']
    e = analyser.polarity_scores(a)['compound']
    neg.append(b)
    pos.append(c)
    neu.append(d)
    compound.append(e)
```

```
100%|██████████| 109248/109248 [20:28<00:00, 88.95it/s]
```

In [33]:

```
project_data["pos"] = pos
```

In [34]:

```
project_data["neg"] = neg
```

In [35]:

```
project_data["neu"] = neu
```

In [36]:

```
project_data["compound"] = compound
```

In [37]:

```
project_data.head(5)
```

Out[37]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_essay_1	pr
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	I have been fortunate enough to use the Fairy ...	M co va ba
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Imagine being 8-9 years old. You're in your th...	M st au ar
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	2016-04-27 00:46:53	Having a class of 24 students comes with diver...	I l tw ki st
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	I recently read an article about giving studen...	I t in so
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	My students crave challenge, they eat obstacle...	W pu el so

5 rows × 24 columns

Splitting Project_Data into Train and Test Datasets

In [38]:

```
# train test split

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(project_data,
project_data['project_is_approved'], test_size=0.33, stratify = project_data['project_is_approved'
])
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

We don't need the 'project_is_approved' feature now

In [39]:

```
X_train.drop(['project_is_approved'], axis=1, inplace=True)
X_test.drop(['project_is_approved'], axis=1, inplace=True)
X_cv.drop(['project_is_approved'], axis=1, inplace=True)
```

Preparing data for models

In [40]:

```
project_data.columns
```

Out[40]:

```
Index([u'Unnamed: 0', u'id', u'teacher_id', u'teacher_prefix', u'school_state',
       u'Date', u'project_essay_1', u'project_essay_2', u'project_essay_3',
       u'project_essay_4', u'project_resource_summary',
       u'teacher_number_of_previously_posted_projects', u'project_is_approved',
       u'project_grade_category', u'clean_categories', u'clean_subcategories',
       u'clean_titles', u'title_word_count', u'clean_essays',
       u'essay_word_count', u'pos', u'neg', u'neu', u'compound'],
      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
```

2.1 Vectorizing Categorical data

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

One Hot Encoding of Clean Categories

In [41]:

```
# we use count vectorizer to convert the values into one

from sklearn.feature_extraction.text import CountVectorizer

vectorizer_proj = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary
=True)
vectorizer_proj.fit(X_train['clean_categories'].values)

categories_one_hot_train = vectorizer_proj.transform(X_train['clean_categories'].values)
categories_one_hot_test = vectorizer_proj.transform(X_test['clean_categories'].values)
categories_one_hot_cv = vectorizer_proj.transform(X_cv['clean_categories'].values)

print(vectorizer_proj.get_feature_names())

print("Shape of matrix of Train data after one hot encoding ",categories_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",categories_one_hot_test.shape)
print("Shape of matrix of CV data after one hot encoding ",categories_one_hot_cv.shape)
```

```
['SpecialNeeds', 'Music_Arts', 'Math_Science', 'Health_Sports', 'Care_Hunger',
'Literacy_Language', 'AppliedLearning', 'History_Civics', 'Warmth']
('Shape of matrix of Train data after one hot encoding ', (49041, 9))
('Shape of matrix of Test data after one hot encoding ', (36052, 9))
('Shape of matrix of CV data after one hot encoding ', (24155, 9))
```

One Hot Encoding of Clean Sub-Categories

In [42]:

```
# we use count vectorizer to convert the values into one

vectorizer_sub_proj = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False
, binary=True)
vectorizer_sub_proj.fit(X_train['clean_subcategories'].values)

sub_categories_one_hot_train = vectorizer_sub_proj.transform(X_train['clean_subcategories'].values
)
sub_categories_one_hot_test = vectorizer_sub_proj.transform(X_test['clean_subcategories'].values)
sub_categories_one_hot_cv = vectorizer_sub_proj.transform(X_cv['clean_subcategories'].values)


print(vectorizer_sub_proj.get_feature_names())

print("Shape of matrix of Train data after one hot encoding ",sub_categories_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",sub_categories_one_hot_test.shape)
print("Shape of matrix of Cross Validation data after one hot encoding ",sub_categories_one_hot_cv
.shape)
```

```
['Health_Wellness', 'Literature_Writing', 'CommunityService', 'Care_Hunger', 'AppliedSciences', 'S
ocialSciences', 'Other', 'Music', 'Mathematics', 'Warmth', 'EnvironmentalScience',
'ForeignLanguages', 'NutritionEducation', 'TeamSports', 'Extracurricular', 'Literacy',
'SpecialNeeds', 'PerformingArts', 'Health_LifeScience', 'Economics', 'ParentInvolvement',
'EarlyDevelopment', 'FinancialLiteracy', 'ESL', 'Civics_Government', 'CharacterEducation',
'History_Geography', 'VisualArts', 'College_CareerPrep', 'Gym_Fitness']
('Shape of matrix of Train data after one hot encoding ', (49041, 30))
('Shape of matrix of Test data after one hot encoding ', (36052, 30))
('Shape of matrix of Cross Validation data after one hot encoding ', (24155, 30))
```

One Hot Encoding of School States

In [43]:

```
my_counter = Counter()
for state in project_data['school_state'].values:
    my_counter.update(state.split())
```

In [44]:

```
school_state_cat_dict = dict(my_counter)
```

```
sorted_school_state_cat_dict = dict(sorted(school_state_cat_dict.items(), key=lambda kv: kv[1]))
```

In [45]:

```
## we use count vectorizer to convert the values into one hot encoded features

vectorizer_states = CountVectorizer(vocabulary=list(sorted_school_state_cat_dict.keys()),
lowercase=False, binary=True)
vectorizer_states.fit(X_train['school_state'].values)

school_state_categories_one_hot_train = vectorizer_states.transform(X_train['school_state'].values
)
school_state_categories_one_hot_test = vectorizer_states.transform(X_test['school_state'].values)
school_state_categories_one_hot_cv = vectorizer_states.transform(X_cv['school_state'].values)

print(vectorizer_states.get_feature_names())

print("Shape of matrix of Train data after one hot encoding
",school_state_categories_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",school_state_categories_one_hot_test.
shape)
print("Shape of matrix of Cross Validation data after one hot encoding
",school_state_categories_one_hot_cv.shape)
```

```
['WA', 'DE', 'DC', 'WI', 'WV', 'HI', 'FL', 'WY', 'NH', 'NJ', 'NM', 'TX', 'LA', 'NC', 'ND', 'NE', 'T
N', 'NY', 'PA', 'RI', 'NV', 'VA', 'CO', 'AK', 'AL', 'AR', 'VT', 'IL', 'GA', 'IN', 'IA', 'MA', 'AZ',
'CA', 'ID', 'CT', 'ME', 'MD', 'OK', 'OH', 'UT', 'MO', 'MN', 'MI', 'KS', 'MT', 'MS', 'SC', 'KY', 'OR
', 'SD']
('Shape of matrix of Train data after one hot encoding ', (49041, 51))
('Shape of matrix of Test data after one hot encoding ', (36052, 51))
('Shape of matrix of Cross Validation data after one hot encoding ', (24155, 51))
```

One Hot Encoding of Project Grade Category

In [46]:

```
my_counter = Counter()
for project_grade in project_data['project_grade_category'].values:
    my_counter.update(project_grade.split())
```

In [47]:

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

In [48]:

```
## we use count vectorizer to convert the values into one hot encoded features

vectorizer_grade = CountVectorizer(vocabulary=list(sorted_project_grade_cat_dict.keys()),
lowercase=False, binary=True)
vectorizer_grade.fit(X_train['project_grade_category'].values)

project_grade_categories_one_hot_train =
vectorizer_grade.transform(X_train['project_grade_category'].values)
project_grade_categories_one_hot_test = vectorizer_grade.transform(X_test['project_grade_category'
].values)
project_grade_categories_one_hot_cv = vectorizer_grade.transform(X_cv['project_grade_category'].va
lues)

print(vectorizer_grade.get_feature_names())

print("Shape of matrix of Train data after one hot encoding
",project_grade_categories_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",project_grade_categories_one_hot_test
.shape)
print("Shape of matrix of Cross Validation data after one hot encoding
",project_grade_categories_one_hot_cv.shape)
```

```
['Grades_6-8', 'Grades_9-12', 'Grades_PreK-2', 'Grades_3-5']
```

```
('Shape of matrix of Train data after one hot encoding ', (49041, 4))
('Shape of matrix of Test data after one hot encoding ', (36052, 4))
('Shape of matrix of Cross Validation data after one hot encoding ', (24155, 4))
```

One Hot Encoding of Teacher Prefix

In [49]:

```
my_counter = Counter()
for teacher_prefix in project_data['teacher_prefix'].values:
    teacher_prefix = str(teacher_prefix)
    my_counter.update(teacher_prefix.split())
```

In [50]:

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

In [51]:

```
## we use count vectorizer to convert the values into one hot encoded features
## Unlike the previous Categories this category returns a
## ValueError: np.nan is an invalid document, expected byte or unicode string.
## The link below explains h0w to tackle such discrepancies.
## https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-scikit-learn-valueerror-np-nan-is-an-invalid-document/39308809#39308809

vectorizer_teacher = CountVectorizer(vocabulary=list(sorted_teacher_prefix_cat_dict.keys()), lower
case=False, binary=True)
vectorizer_teacher.fit(X_train['teacher_prefix'].values.astype("U"))

teacher_prefix_categories_one_hot_train = vectorizer_teacher.transform(X_train['teacher_prefix'].v
alues.astype("U"))
teacher_prefix_categories_one_hot_test =
vectorizer_teacher.transform(X_test['teacher_prefix'].values.astype("U"))
teacher_prefix_categories_one_hot_cv = vectorizer_teacher.transform(X_cv['teacher_prefix'].values.
astype("U"))

print(vectorizer_teacher.get_feature_names())

print("Shape of matrix after one hot encoding ",teacher_prefix_categories_one_hot_train.shape)
print("Shape of matrix after one hot encoding ",teacher_prefix_categories_one_hot_test.shape)
print("Shape of matrix after one hot encoding ",teacher_prefix_categories_one_hot_cv.shape)
```

```
['nan', 'Mrs.', 'Ms.', 'Mr.', 'Dr.', 'Teacher']
('Shape of matrix after one hot encoding ', (49041, 6))
('Shape of matrix after one hot encoding ', (36052, 6))
('Shape of matrix after one hot encoding ', (24155, 6))
```

2.2 Vectorizing Text data

A) Bag of Words (BOW) with min_df=10

BoW- Training Data-Essay Feature

In [123]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).

vectorizer_bow_essay = CountVectorizer(min_df=10)
vectorizer_bow_essay.fit(X_train["clean_essays"])
text_bow_train = vectorizer_bow_essay.transform(X_train["clean_essays"])
print("Shape of matrix after one hot encoding ",text_bow_train.shape)
```

```
('Shape of matrix after one hot encoding ', (49041, 12110))
```

BoW- Test Data-Essay Feature

In [124]:

```
text_bow_test = vectorizer_bow_essay.transform(X_test["clean_essays"])
print("Shape of matrix after one hot encoding ",text_bow_test.shape)
```

```
('Shape of matrix after one hot encoding ', (36052, 12110))
```

BoW- Cross Validation Data-Essay Feature

In [125]:

```
text_bow_cv = vectorizer_bow_essay.transform(X_cv["clean_essays"])
print("Shape of matrix after one hot encoding ",text_bow_cv.shape)
```

```
('Shape of matrix after one hot encoding ', (24155, 12110))
```

BoW- Training Data-Titles Feature

In [126]:

```
vectorizer_bow_title = CountVectorizer(min_df=10)
vectorizer_bow_title.fit(X_train["clean_titles"])
title_bow_train = vectorizer_bow_title.transform(X_train["clean_titles"])
print("Shape of matrix after one hot encoding ",title_bow_train.shape)
```

```
('Shape of matrix after one hot encoding ', (49041, 2089))
```

BoW- Test Data-Titles Feature

In [127]:

```
title_bow_test = vectorizer_bow_title.transform(X_test["clean_titles"])
print("Shape of matrix after one hot encoding ",title_bow_test.shape)
```

```
('Shape of matrix after one hot encoding ', (36052, 2089))
```

BoW- Cross Validation Data-Titles Feature

In [128]:

```
title_bow_cv = vectorizer_bow_title.transform(X_cv["clean_titles"])
print("Shape of matrix after one hot encoding ",title_bow_cv.shape)
```

```
('Shape of matrix after one hot encoding ', (24155, 2089))
```

B) TFIDF vectorizer with min_df=10

TFIDF- Training Data-Essays Feature

In [129]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf_essay = TfidfVectorizer(min_df=10)
vectorizer_tfidf_essay.fit(X_train["clean_essays"])
text_tfidf_train = vectorizer_tfidf_essay.transform(X_train["clean_essays"])
print("Shape of matrix after one hot encoding ",text_tfidf_train.shape)
```

```
('Shape of matrix after one hot encoding ', (49041, 12110))
```

TFIDF- Test Data-Essays Feature

In [130]:

```
text_tfidf_test = vectorizer_tfidf_essay.transform(X_test["clean_essays"])
print("Shape of matrix after one hot encoding ",text_tfidf_test.shape)
```

```
('Shape of matrix after one hot encoding ', (36052, 12110))
```

TFIDF - Cross Validation Data - Essays Feature

In [131]:

```
text_tfidf_cv = vectorizer_tfidf_essay.transform(X_cv["clean_essays"])
print("Shape of matrix after one hot encoding ",text_tfidf_cv.shape)
```

```
('Shape of matrix after one hot encoding ', (24155, 12110))
```

TFIDF - Training Data - Titles Feature

In [132]:

```
vectorizer_tfidf_titles = TfidfVectorizer(min_df=10)

vectorizer_tfidf_titles.fit(X_train["clean_titles"])
title_tfidf_train = vectorizer_tfidf_titles.transform(X_train["clean_titles"])
print("Shape of matrix after one hot encoding ",title_tfidf_train.shape)
```

```
('Shape of matrix after one hot encoding ', (49041, 2089))
```

TFIDF - Test Data - Titles Feature

In [133]:

```
title_tfidf_test = vectorizer_tfidf_titles.transform(X_test["clean_titles"])
print("Shape of matrix after one hot encoding ",title_tfidf_test.shape)
```

```
('Shape of matrix after one hot encoding ', (36052, 2089))
```

TFIDF - Cross Validation Data - Titles Feature

In [134]:

```
title_tfidf_cv = vectorizer_tfidf_titles.transform(X_cv["clean_titles"])
print("Shape of matrix after one hot encoding ",title_tfidf_cv.shape)
```

```
('Shape of matrix after one hot encoding ', (24155, 2089))
```

C) Using Pretrained Models : AVG W2V

In [64]:

```
'''
# 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)


'''
```

In [72]:

```
# 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())
```

Avg_W2V for Train Data(Essays feature)

In [73]:

```
# average Word2Vec
# compute average word2vec for each review.

avg_w2v_vectors_train = [];

for sentence in tqdm(X_train["clean_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_train.append(vector)

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

```
100%|██████████| 49041/49041 [00:21<00:00, 2281.14it/s]
```

```
49041
300
```

Avg_W2V for Test Data(Essays feature)

In [74]:

```
# average Word2Vec
# compute average word2vec for each review.

avg_w2v_vectors_test = [];

for sentence in tqdm(X_test["clean_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_test.append(vector)

print(len(avg_w2v_vectors_test))
print(len(avg_w2v_vectors_test[0]))
```

```
100%|██████████| 36052/36052 [00:12<00:00, 2896.07it/s]
```

```
36052
300
```

Avg_W2V for Cross Validiation Data(Essays feature)

In [75]:

```
avg_w2v_vectors_cv = [];

for sentence in tqdm(X_cv["clean_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_cv.append(vector)

print(len(avg_w2v_vectors_cv))
print(len(avg_w2v_vectors_cv[0]))
```

```
100%|██████████| 24155/24155 [00:08<00:00, 2772.96it/s]
```

```
24155
300
```

Avg_W2V for Train Data(Titles feature)

In [76]:

```
# Similarly you can vectorize for title also

avg_w2v_vectors_titles_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train["clean_titles"]): # for each title
    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_titles_train.append(vector)

print(len(avg_w2v_vectors_titles_train))
print(len(avg_w2v_vectors_titles_train[0]))
```

```
100%|██████████| 49041/49041 [00:02<00:00, 22400.51it/s]
```

```
49041
300
```

Avg_W2V for Test Data(Titles feature)

In [77]:

```
# Similarly you can vectorize for title also

avg_w2v_vectors_titles_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test["clean_titles"]): # for each title
    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_titles_test.append(vector)

print(len(avg_w2v_vectors_titles_test))
print(len(avg_w2v_vectors_titles_test[0]))
```

```
100%|██████████| 36052/36052 [00:00<00:00, 39035.65it/s]
```

```
36052
300
```

Avg_W2V for Cross Validation Data(Titles feature)

In [78]:

```
# Similarly you can vectorize for title also

avg_w2v_vectors_titles_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv["clean_titles"]): # for each title
    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_titles_cv.append(vector)

print(len(avg_w2v_vectors_titles_cv))
print(len(avg_w2v_vectors_titles_cv[0]))
```

```
100%|██████████| 24155/24155 [00:00<00:00, 43883.47it/s]
```

```
24155
300
```

D) Using Pretrained Models: TFIDF weighted W2V

TFIDF weighted W2V for Training Data(Essays feature)

In [79]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train["clean_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 [80]:

```
# 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["clean_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_train.append(vector)

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

```
100%|██████████| 49041/49041 [01:50<00:00, 443.69it/s]
```

```
49041
300
```

TFIDF weighted W2V for Test Data(Essays feature)

In [81]:

```
# 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["clean_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_test.append(vector)

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

```
100%|██████████| 36052/36052 [01:19<00:00, 452.29it/s]
```

```
36052
300
```

TFIDF weighted W2V for Cross Validation Data(Essays feature)

In [82]:

```
# 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["clean_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_cv.append(vector)

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

```
100%|██████████| 24155/24155 [00:52<00:00, 456.61it/s]
```

```
24155
300
```

TFIDF weighted W2V for Train Data(Titles feature)

In [83]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train["clean_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 [84]:

```
# compute average word2vec for each review.

tfidf_w2v_vectors_titles_train = [];

for sentence in tqdm(X_train["clean_titles"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
```

```
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            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_train.append(vector)

print(len(tfidf_w2v_vectors_titles_train))
print(len(tfidf_w2v_vectors_titles_train[0]))
```

```
100%|██████████| 49041/49041 [00:02<00:00, 24031.91it/s]
```

```
49041
300
```

TFIDF weighted W2V for Test Data(Titles feature)

In [85]:

```
# compute average word2vec for each review.

tfidf_w2v_vectors_titles_test = [];

for sentence in tqdm(X_test["clean_titles"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            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_test.append(vector)

print(len(tfidf_w2v_vectors_titles_test))
print(len(tfidf_w2v_vectors_titles_test[0]))
```

```
100%|██████████| 36052/36052 [00:01<00:00, 22799.56it/s]
```

```
36052
300
```

TFIDF weighted W2V for Cross Validation Data(Titles feature)

In [86]:

```
# compute average word2vec for each review.

tfidf_w2v_vectors_titles_cv = [];

for sentence in tqdm(X_cv["clean_titles"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            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_cv.append(vector)

print(len(tfidf_w2v_vectors_titles_cv))
print(len(tfidf_w2v_vectors_titles_cv[0]))
```

```
100%|██████████| 24155/24155 [00:01<00:00, 22358.97it/s]
```

```
24155
300
```

2.3 Vectorizing Numerical features

In [87]:

```
# https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexes-for-all-groups-in
-one-step
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
# join two dataframes in python:
X_train = pd.merge(X_train, price_data, on='id', how='left')
X_test = pd.merge(X_test, price_data, on='id', how='left')
X_cv = pd.merge(X_cv, price_data, on='id', how='left')
price_data.head(2)
```

Out[87]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21

A) Price Feature

In [89]:

```
# 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(X_train['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_train = price_scalar.transform(X_train['price'].values.reshape(-1, 1))
price_standardized_test = price_scalar.transform(X_test['price'].values.reshape(-1, 1))
price_standardized_cv = price_scalar.transform(X_cv['price'].values.reshape(-1, 1))

print("After Vectorizations ")
print(price_standardized_train.shape, y_train.shape)
print(price_standardized_cv.shape, y_cv.shape)
print(price_standardized_test.shape, y_test.shape)
```

```
After vectorizations
```

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

B) Quantity Feature

In [90]:

```
# 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
import warnings
warnings.filterwarnings("ignore")

# 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)

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

# Now standardize the data with above maen and variance.
quantity_standardized_train = quantity_scalar.transform(X_train['quantity'].values.reshape(-1, 1))
quantity_standardized_test = quantity_scalar.transform(X_test['quantity'].values.reshape(-1, 1))
quantity_standardized_cv = quantity_scalar.transform(X_cv['quantity'].values.reshape(-1, 1))

print("After Vectorizations ")
print(quantity_standardized_train.shape, y_train.shape)
print(quantity_standardized_cv.shape, y_cv.shape)
print(quantity_standardized_test.shape, y_test.shape)
```

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

C) Number of Projects previously proposed by Teacher Feature

In [91]:

```
# 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
import warnings
warnings.filterwarnings("ignore")

# 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)

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

# Now standardize the data with above maen and variance.
ppt_standardized_train =
ppt_scalar.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,
```

```
1))
ppt_standardized_test = ppt_scalar.transform(X_test['teacher_number_of_previously_posted_projects'
].values.reshape(-1, 1))
ppt_standardized_cv = ppt_scalar.transform(X_cv['teacher_number_of_previously_posted_projects'].va
lues.reshape(-1, 1))

print("After Vectorizations ")
print(ppt_standardized_train.shape, y_train.shape)
print(ppt_standardized_cv.shape, y_cv.shape)
print(ppt_standardized_test.shape, y_test.shape)
```

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

D) Title word Count Feature

In [92]:

```
# 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
import warnings
warnings.filterwarnings("ignore")

# 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)

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

# Now standardize the data with above maen and variance.
twc_standardized_train = twc_scalar.transform(X_train['title_word_count'].values.reshape(-1, 1))
twc_standardized_test = twc_scalar.transform(X_test['title_word_count'].values.reshape(-1, 1))
twc_standardized_cv = twc_scalar.transform(X_cv['title_word_count'].values.reshape(-1, 1))

print("After Vectorizations ")
print(twc_standardized_train.shape, y_train.shape)
print(twc_standardized_cv.shape, y_cv.shape)
print(twc_standardized_test.shape, y_test.shape)
```

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

E) Essay word Count Feature

In [93]:

```
# 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
import warnings
warnings.filterwarnings("ignore")

# 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)

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

# Now standardize the data with above mean and variance.
ewc_standardized_train = ewc_scalar.transform(X_train['essay_word_count'].values.reshape(-1, 1))
ewc_standardized_test = ewc_scalar.transform(X_test['essay_word_count'].values.reshape(-1, 1))
ewc_standardized_cv = ewc_scalar.transform(X_cv['essay_word_count'].values.reshape(-1, 1))

print("After Vectorizations ")
print(ewc_standardized_train.shape, y_train.shape)
print(ewc_standardized_cv.shape, y_cv.shape)
print(ewc_standardized_test.shape, y_test.shape)
```

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

F) Essay Sentiments - positives

In [94]:

```
# 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
import warnings
warnings.filterwarnings("ignore")

# 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)

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

# Now standardize the data with above mean and variance.
pos_standardized_train = pos_scalar.transform(X_train['pos'].values.reshape(-1, 1))
pos_standardized_test = pos_scalar.transform(X_test['pos'].values.reshape(-1, 1))
pos_standardized_cv = pos_scalar.transform(X_cv['pos'].values.reshape(-1, 1))

print("After Vectorizations")
print(pos_standardized_train.shape, y_train.shape)
print(pos_standardized_cv.shape, y_cv.shape)
print(pos_standardized_test.shape, y_test.shape)
```

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

G) Essay Sentiments - negatives

In [95]:

```
# 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
```

```
import warnings
warnings.filterwarnings("ignore")

# 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)

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

# Now standardize the data with above mean and variance.
neg_standardized_train = neg_scalar.transform(X_train['neg'].values.reshape(-1, 1))
neg_standardized_test = neg_scalar.transform(X_test['neg'].values.reshape(-1, 1))
neg_standardized_cv = neg_scalar.transform(X_cv['neg'].values.reshape(-1, 1))

print("After Vectorizations ")
print(neg_standardized_train.shape, y_train.shape)
print(neg_standardized_cv.shape, y_cv.shape)
print(neg_standardized_test.shape, y_test.shape)
```

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

H) Essay Sentiments - neutrals

In [96]:

```
# 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
import warnings
warnings.filterwarnings("ignore")

# 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)

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

# Now standardize the data with above mean and variance.
neu_standardized_train = neu_scalar.transform(X_train['neu'].values.reshape(-1, 1))
neu_standardized_test = neu_scalar.transform(X_test['neu'].values.reshape(-1, 1))
neu_standardized_cv = neu_scalar.transform(X_cv['neu'].values.reshape(-1, 1))

print("After Vectorizations ")
print(neu_standardized_train.shape, y_train.shape)
print(neu_standardized_cv.shape, y_cv.shape)
print(neu_standardized_test.shape, y_test.shape)
```

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

I) Essay Sentiments - compound

In [97]:

```
# 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
import warnings
warnings.filterwarnings("ignore")

# 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)

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

# Now standardize the data with above mean and variance.
com_standardized_train = com_scalar.transform(X_train['compound'].values.reshape(-1, 1))
com_standardized_test = com_scalar.transform(X_test['compound'].values.reshape(-1, 1))
com_standardized_cv = com_scalar.transform(X_cv['compound'].values.reshape(-1, 1))

print("After Column Standardisation: ")
print(com_standardized_train.shape, y_train.shape)
print(com_standardized_cv.shape, y_cv.shape)
print(com_standardized_test.shape, y_test.shape)
```

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

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_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.

3. Support Vector Machines

Set 1: Categorical, Numerical features + Project_title(BOW) + Preprocessed_essay (BOW with min_df=10)

In [162]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr = hstack((categories_one_hot_train, sub_categories_one_hot_train,
school_state_categories_one_hot_train, project_grade_categories_one_hot_train,
teacher_prefix_categories_one_hot_train, price_standardized_train, quantity_standardized_train, pp
t_standardized_train, twc_standardized_train, ewc_standardized_train,pos_standardized_train,neg_st
andardized_train,neu_standardized_train, com_standardized_train,title_bow_train, text_bow_train)).
tocsr()
X_te = hstack((categories_one_hot_test, sub_categories_one_hot_test,
school_state_categories_one_hot_test, project_grade_categories_one_hot_test,
teacher_prefix_categories_one_hot_test, price_standardized_test, quantity_standardized_test,
ppt_standardized_test, twc_standardized_test,
ewc_standardized_test,pos_standardized_test,neg_standardized_test,neu_standardized_test,
com_standardized_test, text_bow_test, title_bow_test)).tocsr()
X_cr = hstack((categories_one_hot_cv, sub_categories_one_hot_cv,
school_state_categories_one_hot_cv, project_grade_categories_one_hot_cv,
teacher_prefix_categories_one_hot_cv,price_standardized_cv, quantity_standardized_cv,
ppt_standardized_cv, twc_standardized_cv,
ewc_standardized_cv,pos_standardized_cv,neg_standardized_cv,neu_standardized_cv,
com_standardized_cv, title_bow_cv, text_bow_cv)).tocsr()
```

In [163]:

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

```
Final Data matrix
((49041, 14308), (49041,))
((24155, 14308), (24155,))
```

((36052, 14308), (36052,))
==========

A) GridSearchCV - L2-Regularization

In [164]:

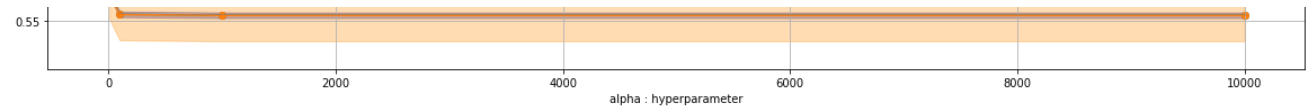
```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier
```

In [165]:

```
sv = SGDClassifier(loss='hinge', penalty='l2')

parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}

clf = GridSearchCV(sv, parameters, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [166]:

```
plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color=
'darkorange')

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


plt.legend()
plt.xlabel("Alpha")
plt.ylabel("AUC")
plt.title("Alpha v/s AUC plot")
plt.grid()
plt.show()
```



Summary:

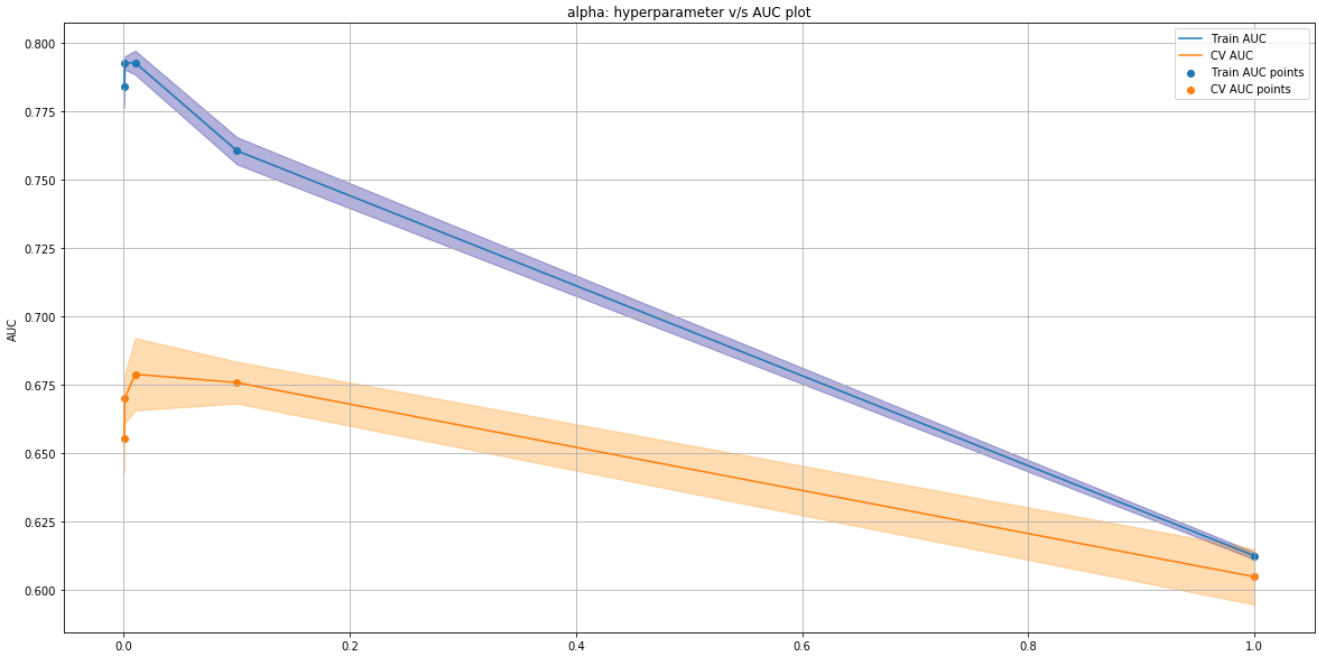
- Need to re-run the GridSearchCV on a smaller set of alpha values to actually visualize what is happening

In [167]:

```
sv = SGDClassifier(loss='hinge', penalty='l2')

parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0]}

clf = GridSearchCV(sv, parameters, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [168]:

```
plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color=
'darkorange')

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


plt.legend()
plt.xlabel("Alpha")
plt.ylabel("AUC")
plt.title("Alpha v/s AUC plot")
plt.grid()
plt.show()
```



Summary

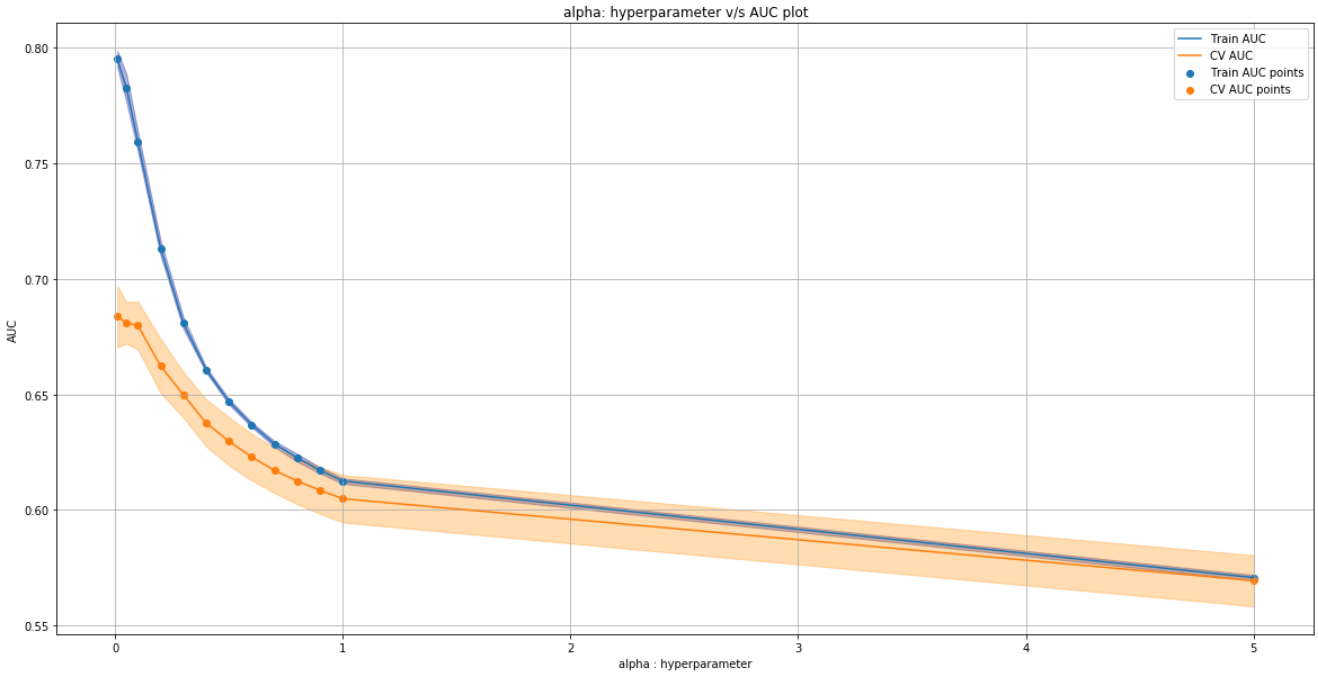
1. Need to re-run the GridSearchCV on a smaller set of alpha to get the appropriate alpha value

In [169]:

```
sv = SGDClassifier(loss='hinge', penalty='l2')

parameters = {'alpha':[0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 5.0]}

clf = GridSearchCV(sv, parameters, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [170]:

```
plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color=
'darkorange')

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


plt.legend()
plt.xlabel("Alpha")
plt.ylabel("AUC")
plt.title("Alpha v/s AUC plot")
plt.grid()
plt.show()
```



Summary

1. For 0.1 there seems to be a huge difference between the Train and the Test model.
2. 0.05 seems to be the best value

In [ ]:

```
best_alpha_l2=0.05
```

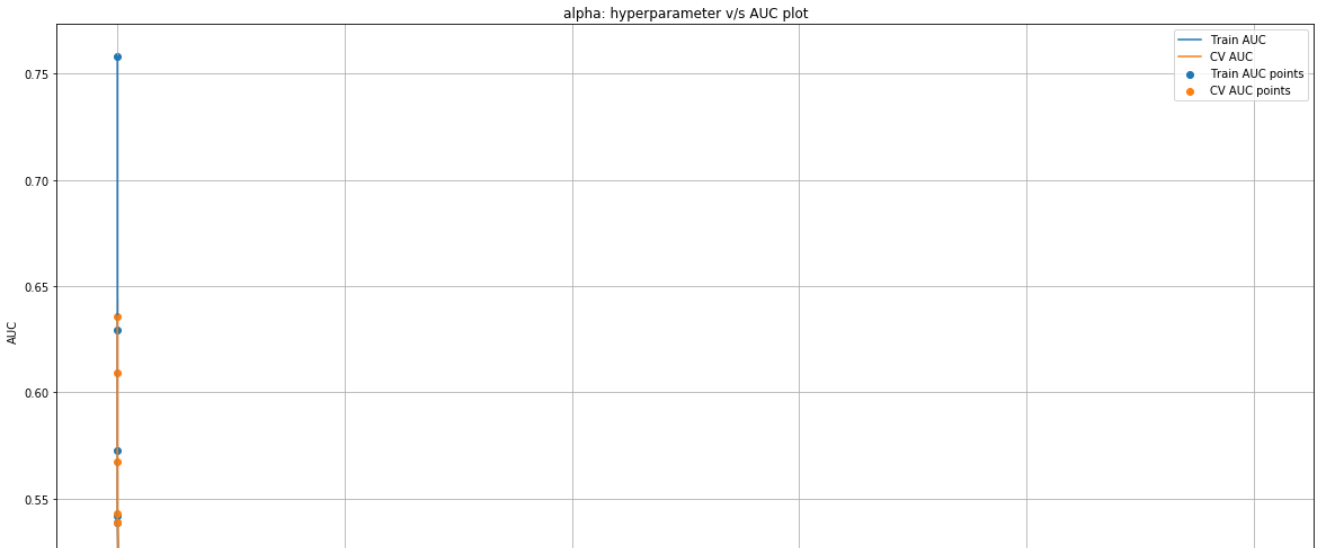
B) GridSearchCV - L1-Regularization

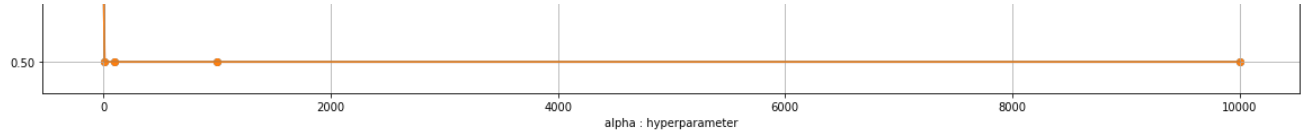
In [171]:

```
sv = SGDClassifier(loss='hinge', penalty='l1')

parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}

clf = GridSearchCV(sv, parameters, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [172]:

```
plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color=
'darkorange')

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


plt.legend()
plt.xlabel("Alpha")
plt.ylabel("AUC")
plt.title("Alpha")
plt.grid()
plt.show()
```





Summary

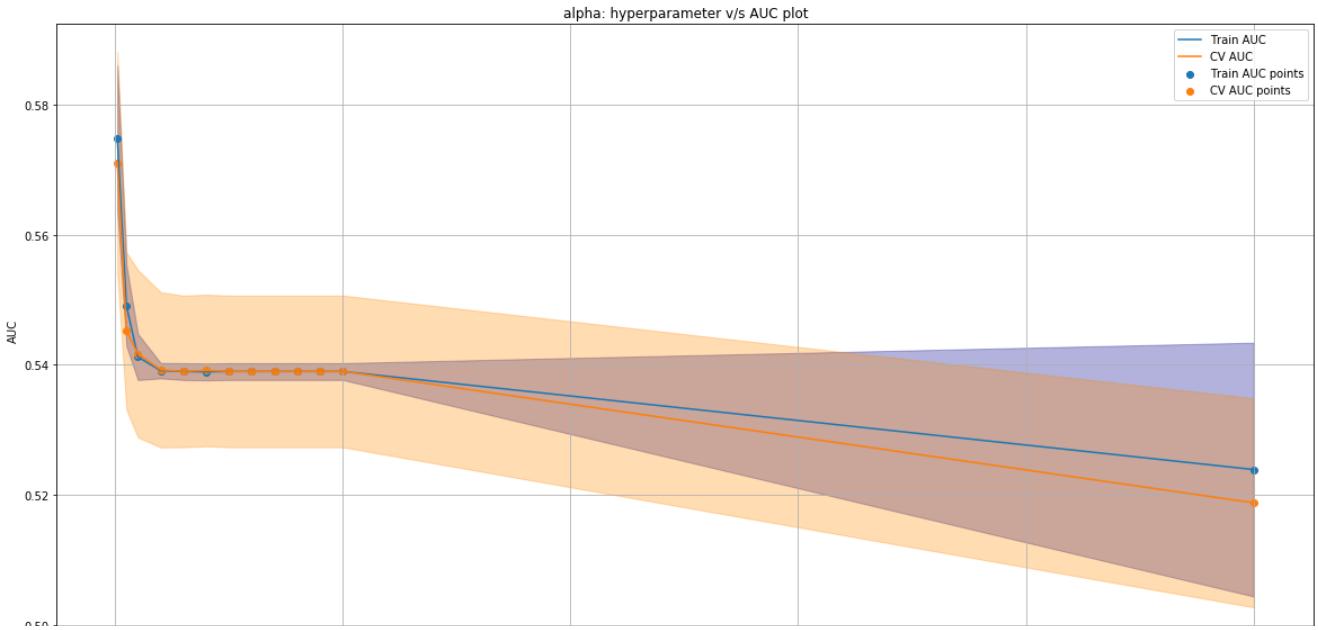
1. Need to re-run the GridSearchCV on a smaller set of alpha to get more clarity.

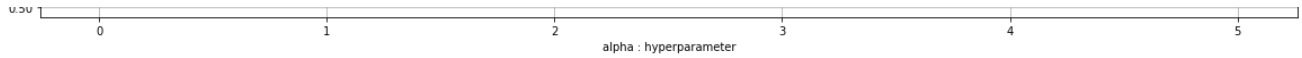
In [175]:

```
sv = SGDClassifier(loss='hinge', penalty='l1')

parameters = {'alpha':[0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 5.0]}

clf = GridSearchCV(sv, parameters, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [176]:

```
plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color=
'darkorange')

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


plt.legend()
plt.xlabel("alpha : hyperparameter")
plt.ylabel("AUC")
plt.title("alpha: hyperparameter v/s AUC plot")
plt.grid()
plt.show()
```





Summary

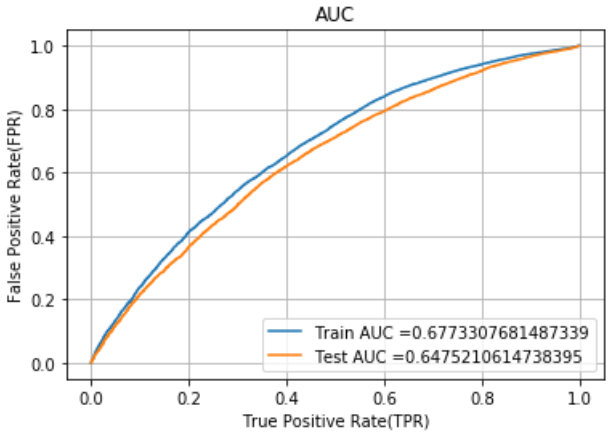
1. L1 regularization yields a comparitively thicker AUC . Can't choose the appropriate alpha .

I will choose our best alpha to be the one which I found using 10 Fold CV Using L2-Regularisation.

C) Train the model using the best hyper parameter value(L2)

In [178]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


model = SGDClassifier(loss='hinge', penalty='l2', alpha=best_alpha_l2)

model.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 = model.decision_function(X_tr)
y_test_pred = model.decision_function(X_te)

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

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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



D) Confusion Matrix

In [180]:

```
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(fpr*(1-tpr))]

    # (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))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

Train Data

In [181]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
('the maximum value of tpr*(1-fpr)', 0.25, 'for threshold', 1.007)
[[ 3713  3713]
 [10310 31305]]
```

In [182]:

```
conf_matr_df_train_1 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds,
train_fpr, train_fpr)), range(2),range(2))
```

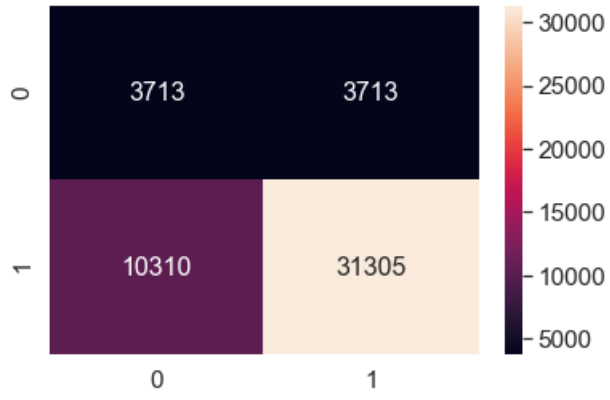
```
('the maximum value of tpr*(1-fpr)', 0.25, 'for threshold', 1.007)
```

In [383]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_1, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[383]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a8667f510>
```



Test Data

In [183]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

```
Test confusion matrix
('the maximum value of tpr*(1-fpr)', 0.24999999161092998, 'for threshold', 1.031)
[[ 3586  1873]
 [13514 17079]]
```

In [184]:

```
conf_matr_df_test_1 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, tes
t_fpr, test_fpr)), range(2),range(2))
```

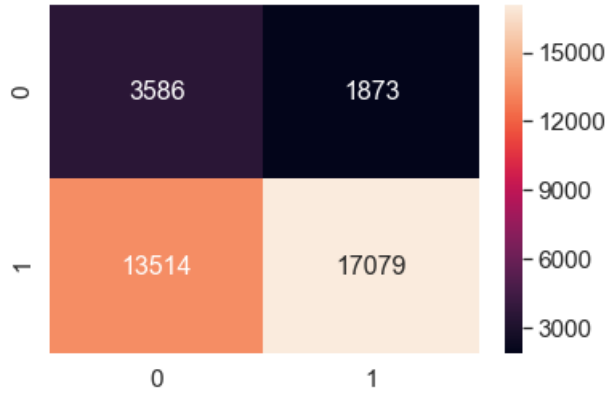
```
('the maximum value of tpr*(1-fpr)', 0.24999999161092998, 'for threshold', 1.031)
```

In [382]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_1, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[382]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a86867a50>
```



Set 2 : Categorical, Numerical features + Project_title(TFIDF) + Preprocessed_essay (TFIDF min_df=10)

In [185]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr = hstack((categories_one_hot_train, sub_categories_one_hot_train,
school_state_categories_one_hot_train, project_grade_categories_one_hot_train,
teacher_prefix_categories_one_hot_train, price_standardized_train, quantity_standardized_train, pp
t_standardized_train, twc_standardized_train, ewc_standardized_train,
title_tfidf_train,pos_standardized_train,neg_standardized_train,neu_standardized_train,
com_standardized_train, text_tfidf_train)).tocsr()
X_te = hstack((categories_one_hot_test, sub_categories_one_hot_test,
school_state_categories_one_hot_test, project_grade_categories_one_hot_test,
teacher_prefix_categories_one_hot_test, price_standardized_test, quantity_standardized_test,
ppt_standardized_test, twc_standardized_test, ewc_standardized_test,
pos_standardized_test,neg_standardized_test,neu_standardized_test,
com_standardized_test,text_tfidf_test, title_tfidf_test)).tocsr()
X_cr = hstack((categories_one_hot_cv, sub_categories_one_hot_cv,
school_state_categories_one_hot_cv, project_grade_categories_one_hot_cv,
teacher_prefix_categories_one_hot_cv,price_standardized_cv, quantity_standardized_cv,
ppt_standardized_cv, twc_standardized_cv,
ewc_standardized_cv,pos_standardized_cv,neg_standardized_cv,neu_standardized_cv,
com_standardized_cv, title_tfidf_cv, text_tfidf_cv)).tocsr()
```

In [186]:

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

```
print("="*100)
```

```
Final Data matrix
((49041, 14308), (49041,))
((24155, 14308), (24155,))
((36052, 14308), (36052,))
====================================================================================================
```

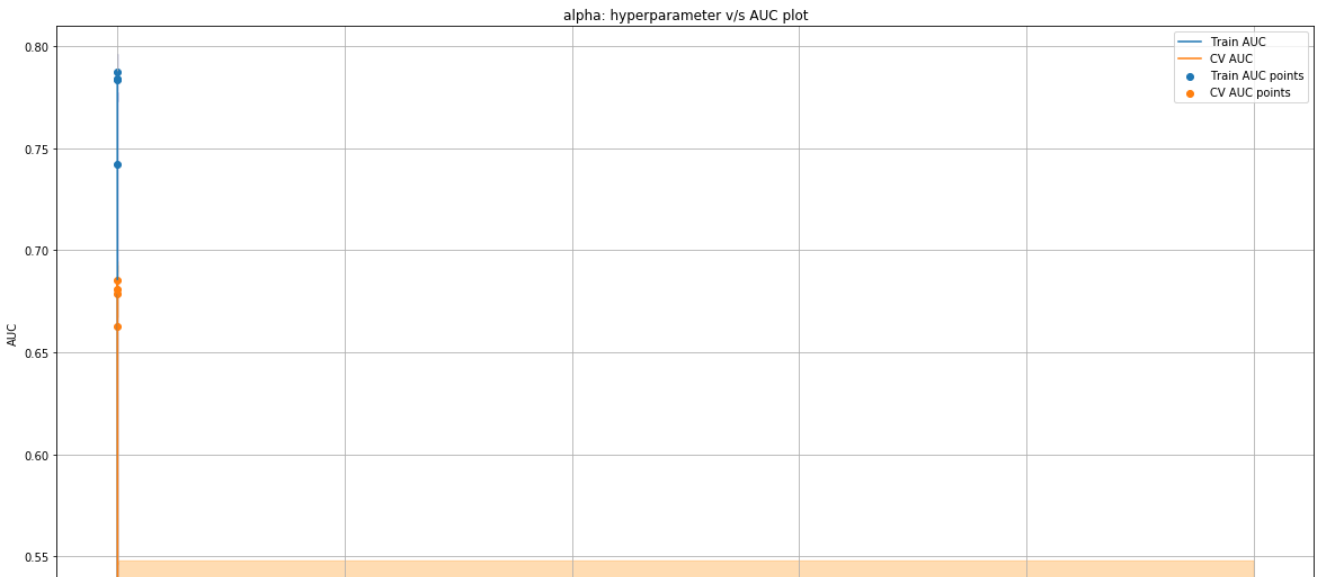
A) GridSearchCV - L2-Regularization

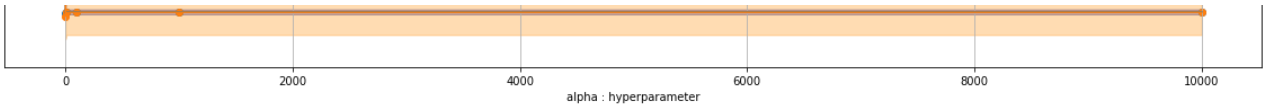
In [187]:

```
sv = SGDClassifier(loss='hinge', penalty='l2')

parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}

clf = GridSearchCV(sv, parameters, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [188]:

```
plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color=
'darkorange')

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


plt.legend()
plt.xlabel("Alpha")
plt.ylabel("AUC")
plt.title("Alpha v/s AUC plot")
plt.grid()
plt.show()
```





Summary

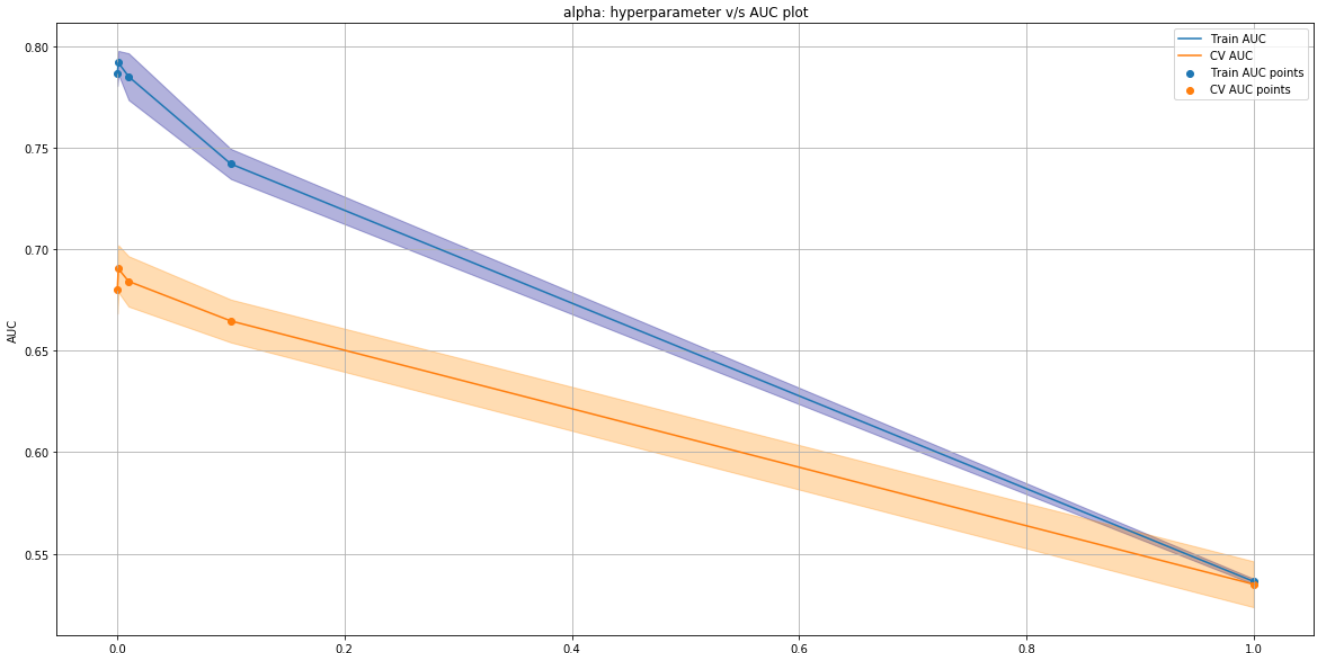
1. Need to re-run the GridSearchCV on a smaller set of alpha to get a better clarity.

In [189]:

```
sv = SGDClassifier(loss='hinge', penalty='l2')

parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0]}

clf = GridSearchCV(sv, parameters, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [190]:

```
plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color=
'darkorange')

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


plt.legend()
plt.xlabel("Alpha")
plt.ylabel("AUC")
plt.title("Alpha v/s AUC plot")
plt.grid()
plt.show()
```



Summary

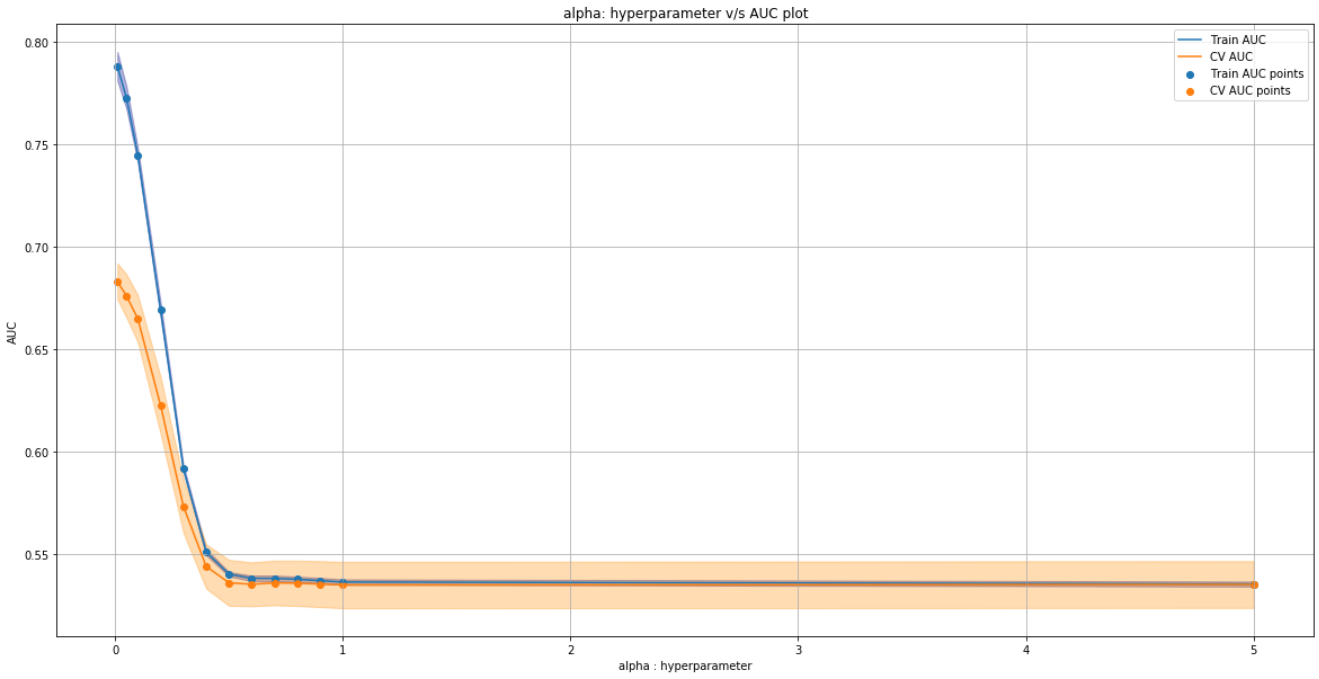
1. Need to re-run the GridSearchCV on a smaller set of parameter values.

In [191]:

```
sv = SGDClassifier(loss='hinge', penalty='l2')

parameters = {'alpha':[0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 5.0]}

clf = GridSearchCV(sv, parameters, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [192]:

```
plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color=
'darkorange')

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


plt.legend()
plt.xlabel("Alpha")
plt.ylabel("AUC")
plt.title("Alpha v/s AUC plot")
plt.grid()
plt.show()
```



Summary

1. For 0.1 there seems to be a huge difference between the Train and the Test model.
2. 0.4 seems to be the best one

In [ ]:

```
best_alpha_l2=0.4
```

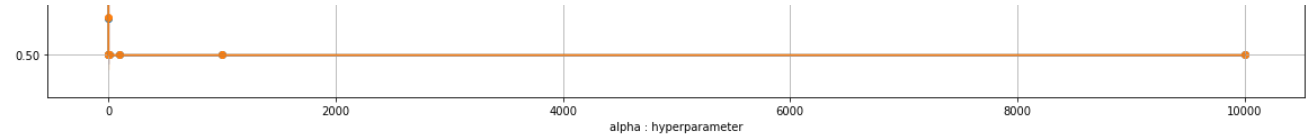
B) GridSearchCV - L1-Regularization

In [193]:

```
sv = SGDClassifier(loss='hinge', penalty='l1')

parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}

clf = GridSearchCV(sv, parameters, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [194]:

```
plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color=
'darkorange')

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


plt.legend()
plt.xlabel("alpha : hyperparameter")
plt.ylabel("AUC")
plt.title("alpha: hyperparameter v/s AUC plot")
plt.grid()
plt.show()
```



Summary

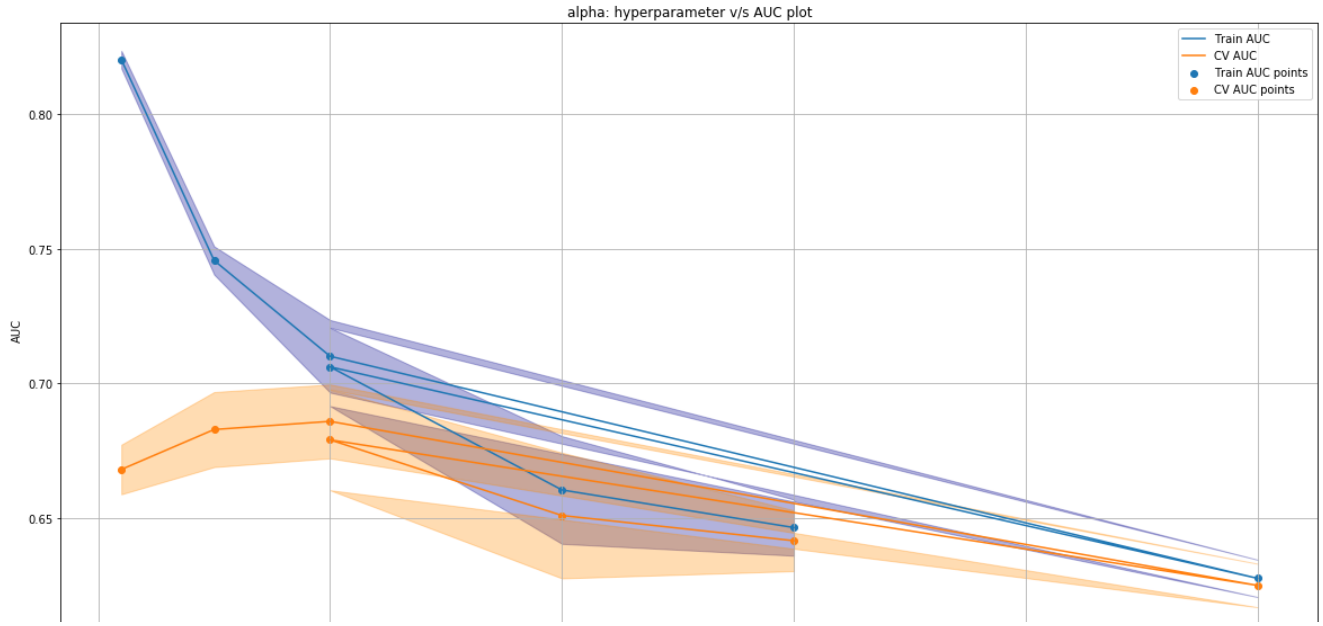
- Need to re-run the GridSearchCV on a smaller set of alpha to get more clarity.

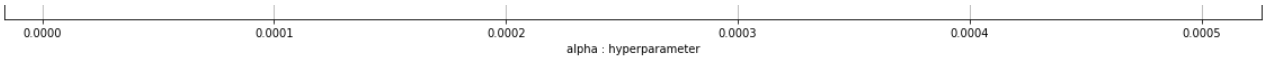
In [223]:

```
sv = SGDClassifier(loss='hinge', penalty='l1')

parameters = {'alpha':[0.00001,0.00005,0.0001, 0.0005, 0.0001, 0.0002, 0.0003]}

clf = GridSearchCV(sv, parameters, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [225]:

```
plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color=
'darkorange')

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


plt.legend()
plt.xlabel("Alpha")
plt.ylabel("AUC")
plt.title("Alpha v/s AUC plot")
plt.grid()
plt.show()
```





Summary

1. 0.0001 was chosen as an appropriate value for my parameter.
2. L1 Regularization seems to yield better parameter value when compared to L2 Regularization. ### So, we will choose best alpha to be the one we found using penalty as L1-Regularisation

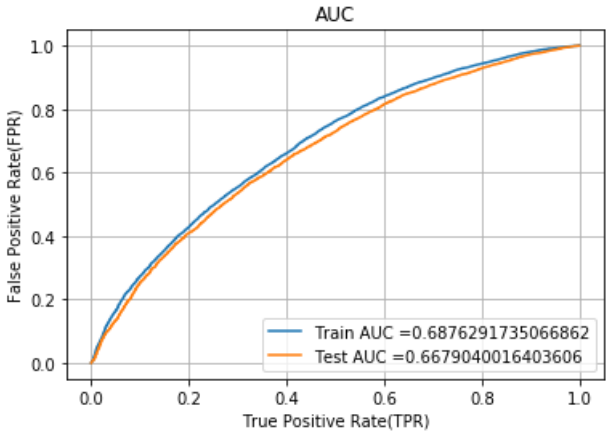
In [ ]:

```
best_alpha_l1=0.0001
```

C) Train the model using the best hyper parameter value

In [222]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


model = SGDClassifier(loss='hinge', penalty='l1', alpha=best_alpha_l1)

model.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 = model.decision_function(X_tr)
y_test_pred = model.decision_function(X_te)

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

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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



D) Confusion Matrix

Train Data

In [226]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
('the maximum value of tpr*(1-fpr)', 0.2499999818661462, 'for threshold', 1.268)
[[ 3714  3712]
 [ 9897 31718]]
```

In [227]:

```
conf_matr_df_train_2 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds,
train_fpr, train_fpr)), range(2),range(2))
```

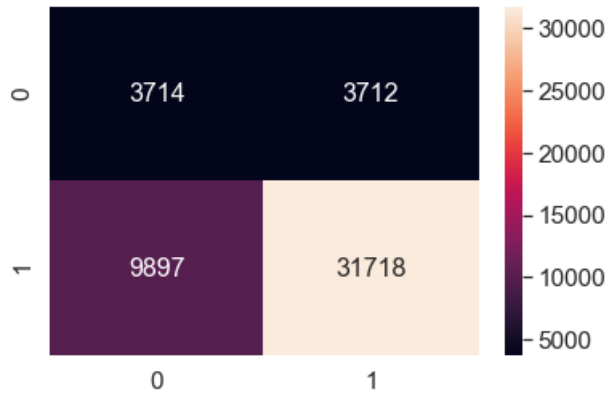
```
('the maximum value of tpr*(1-fpr)', 0.2499999818661462, 'for threshold', 1.268)
```

In [381]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_2, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[381]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b2f8c2e50>
```



Test Data

In [228]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

Test confusion matrix
('the maximum value of tpr*(1-fpr)', 0.24999999161092998, 'for threshold', 1.38)
[[ 3420  2039]
 [11828 18765]]
```

In [229]:

```
conf_matr_df_test_2 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, tes
t_fpr, test_fpr)), range(2),range(2))
```

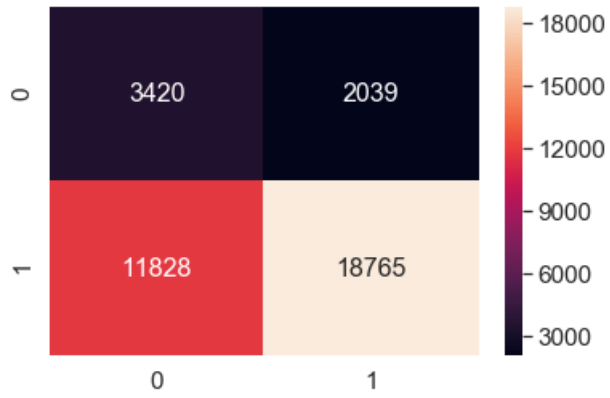
```
('the maximum value of tpr*(1-fpr)', 0.24999999161092998, 'for threshold', 1.38)
```

In [380]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_2, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[380]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b2fff9a90>
```



Set 3 : Categorical, Numerical features + Project_title(AVG W2V) + Preprocessed_essay (AVG W2V)

In [231]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr = hstack((categories_one_hot_train, sub_categories_one_hot_train,
school_state_categories_one_hot_train, project_grade_categories_one_hot_train,
teacher_prefix_categories_one_hot_train, price_standardized_train, quantity_standardized_train, pp
t_standardized_train, twc_standardized_train, ewc_standardized_train,pos_standardized_train,neg_st
andardized_train,neu_standardized_train, com_standardized_train, avg_w2v_vectors_train,
avg_w2v_vectors_titles_train)).tocsr()
X_te = hstack((categories_one_hot_test, sub_categories_one_hot_test,
school_state_categories_one_hot_test, project_grade_categories_one_hot_test,
teacher_prefix_categories_one_hot_test, price_standardized_test, quantity_standardized_test,
ppt_standardized_test, twc_standardized_test,
ewc_standardized_test,pos_standardized_test,neg_standardized_test,neu_standardized_test,
com_standardized_test, avg_w2v_vectors_test, avg_w2v_vectors_titles_test)).tocsr()
X_cr = hstack((categories_one_hot_cv, sub_categories_one_hot_cv,
school_state_categories_one_hot_cv, project_grade_categories_one_hot_cv,
teacher_prefix_categories_one_hot_cv,price_standardized_cv, quantity_standardized_cv,
ppt_standardized_cv, twc_standardized_cv, ewc_standardized_cv, pos_standardized_cv,neg_standardized
_cv,neu_standardized_cv, com_standardized_cv,avg_w2v_vectors_cv, avg_w2v_vectors_titles_cv)).tocsr
()
```

In [232]:

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

```
Final Data matrix
((49041, 709), (49041,))
((24155, 709), (24155,))
((36052, 709), (36052,))
====================================================================================================
```

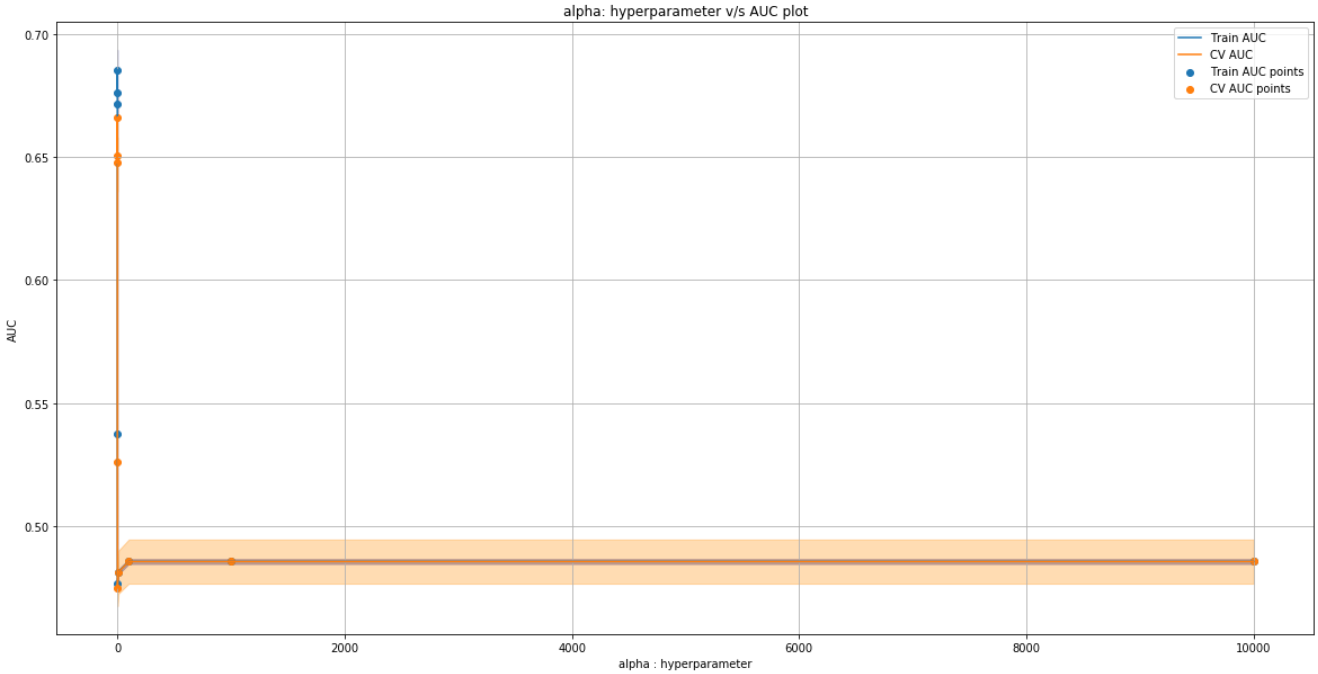
A) GridSearchCV - L2-Regularization

In [233]:

```
sv = SGDClassifier(loss='hinge', penalty='l2')

parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}

clf = GridSearchCV(sv, parameters, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [234]:

```
plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color=
'darkorange')

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


plt.legend()
plt.xlabel("alpha")
plt.ylabel("AUC")
plt.title("alpha v/s AUC plot")
plt.grid()
plt.show()
```



Summary

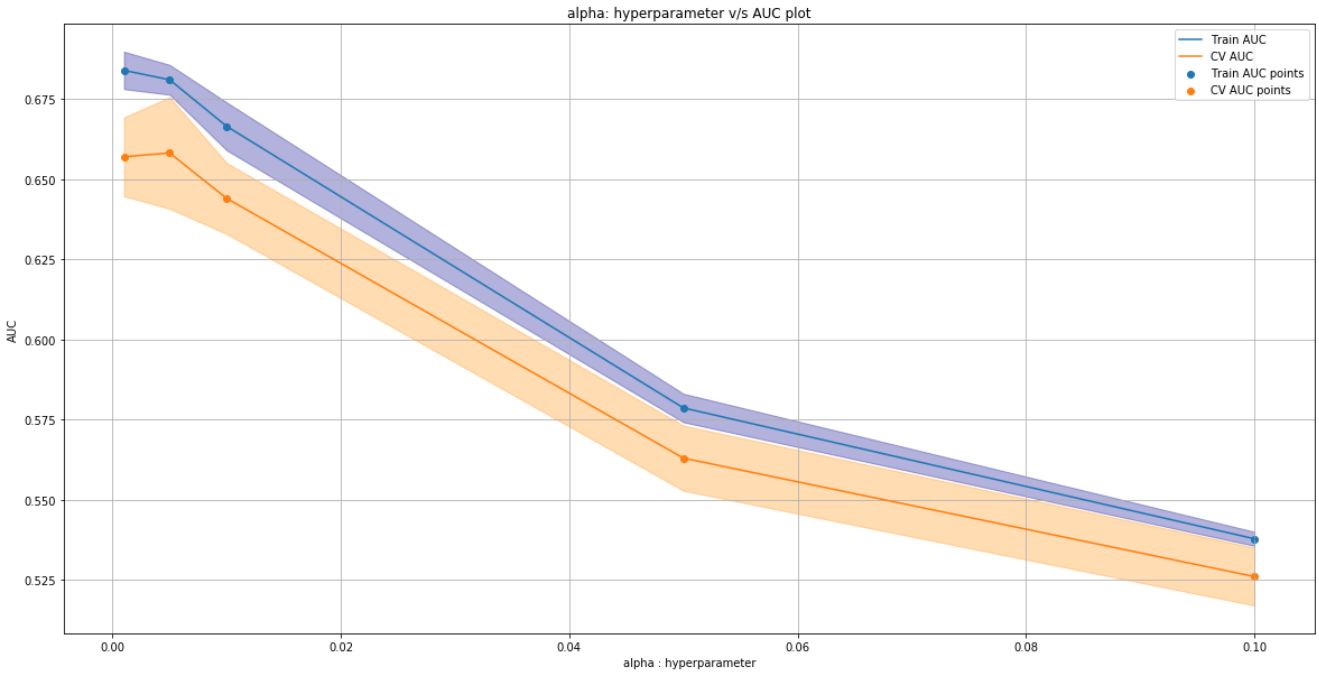
1. Need to re-run the GridSearchCV on a smaller set of alpha to get more clarity.
2. I shall give a try to the values in the range of 10^-3 to 10^-1

In [235]:

```
sv = SGDClassifier(loss='hinge', penalty='l2')

parameters = {'alpha':[0.001, 0.005, 0.01, 0.05, 0.1]}

clf = GridSearchCV(sv, parameters, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [236]:

```
plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color=
'darkorange')

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


plt.legend()
plt.xlabel("alpha ")
plt.ylabel("AUC")
plt.title("alpha v/s AUC plot")
plt.grid()
plt.show()
```



Summary

1. 0.001 is considered as the best value, because the points after and before have a lesser AUC score.

In [ ]:

```
best_alpha_l2=0.001
```

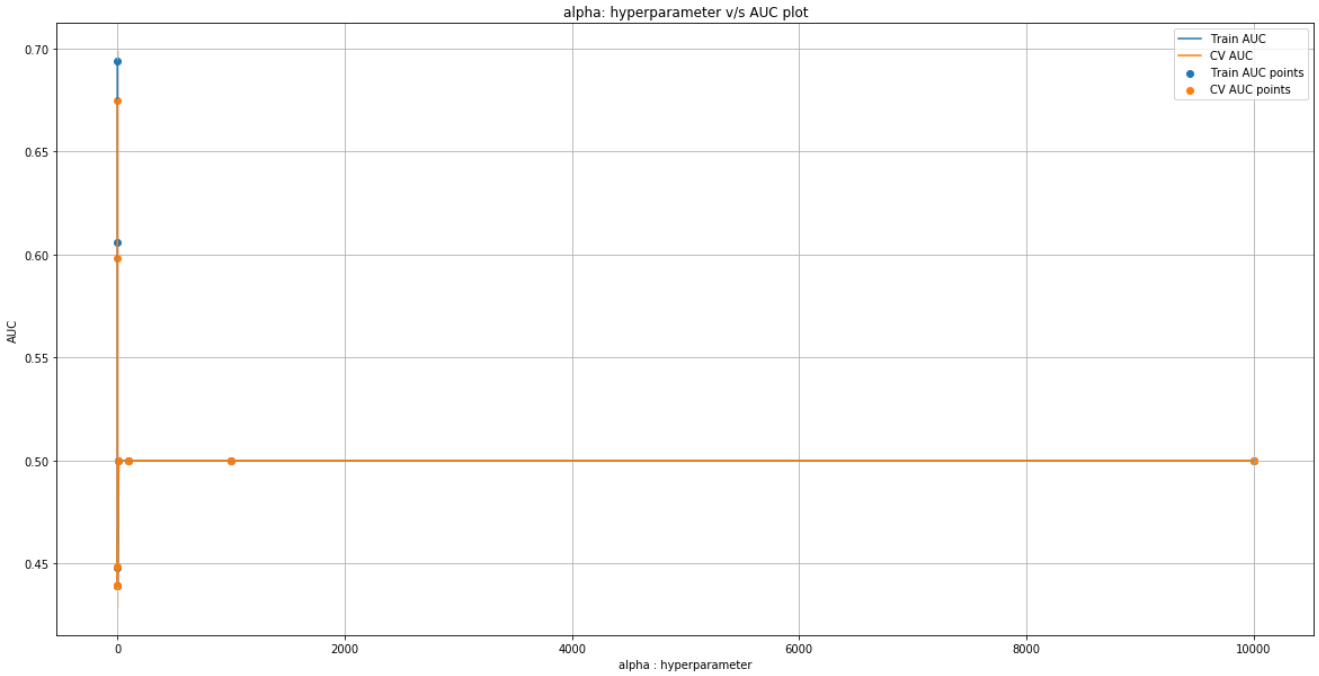
B) GridSearchCV - L1-Regularization

In [237]:

```
sv = SGDClassifier(loss='hinge', penalty='l1')

parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}

clf = GridSearchCV(sv, parameters, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [238]:

```
plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color=
'darkorange')

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


plt.legend()
plt.xlabel("alpha")
plt.ylabel("AUC")
plt.title("alpha v/s AUC plot")
plt.grid()
plt.show()
```



Summary

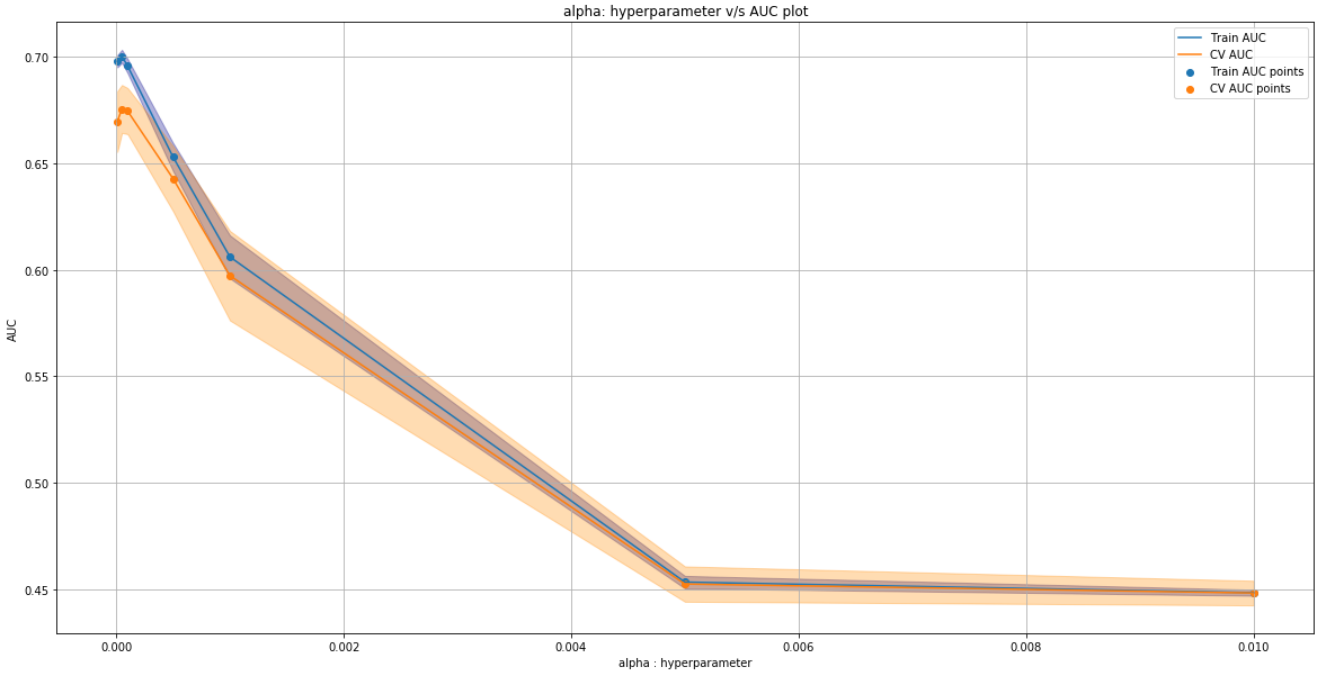
1. Need to re-run the GridSearchCV on a smaller set of alpha to get more clarity.

In [239]:

```
sv = SGDClassifier(loss='hinge', penalty='l1')

parameters = {'alpha':[0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01]}

clf = GridSearchCV(sv, parameters, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [240]:

```
plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color=
'darkorange')

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


plt.legend()
plt.xlabel("alpha")
plt.ylabel("AUC")
plt.title("alpha v/s AUC plot")
plt.grid()
plt.show()
```



Summary

1. Values around 0.00005 to 0.00011 had almost similar AUC scores and similar Difference in Test and Cross Validation AUC scores.

BOTH L1 & L2 PERFORM EQUALLY GOOD HERE

BOTH L1 & L2 PERFORM EQUALLY GOOD HERE

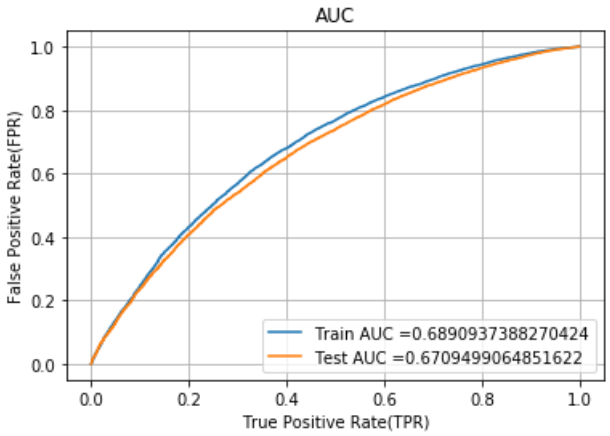
In [ ]:

```
best_alpha_l1=0.00005
```

C) Train the model using the best hyper parameter value (L2)

In [259]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


model = SGDClassifier(loss='hinge', penalty='l2', alpha=best_alpha_l2)

model.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 = model.decision_function(X_tr)
y_test_pred = model.decision_function(X_te)

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

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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



D) Confusion Matrix (L2)

Train Data

In [260]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
```

```
('the maximum value of tpr*(1-fpr)', 0.25, 'for threshold', 1.009)
[[ 3713  3713]
 [ 9671 31944]]
```

In [261]:

```
conf_matr_df_train_3_l2 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred,
tr_thresholds, train_fpr, train_fpr)), range(2),range(2))
```

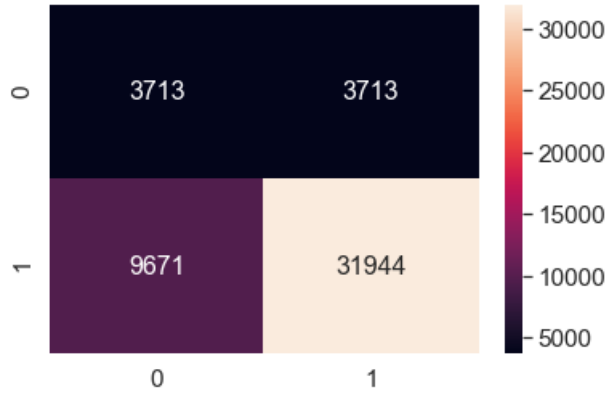
```
('the maximum value of tpr*(1-fpr)', 0.25, 'for threshold', 1.009)
```

In [379]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_3_l2, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[379]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a86bd9b10>
```



Test Data

In [262]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

Test confusion matrix
('the maximum value of tpr*(1-fpr)', 0.24999999161092998, 'for threshold', 1.021)
[[ 3341  2118]
 [11037 19556]]
```

In [263]:

```
conf_matr_df_test_3_l2 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds,
test_fpr, test_fpr)), range(2),range(2))
```

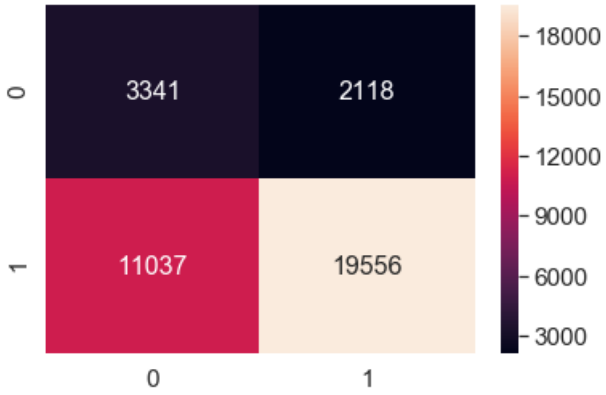
```
('the maximum value of tpr*(1-fpr)', 0.24999999161092998, 'for threshold', 1.021)
```

In [378]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_3_l2, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[378]:

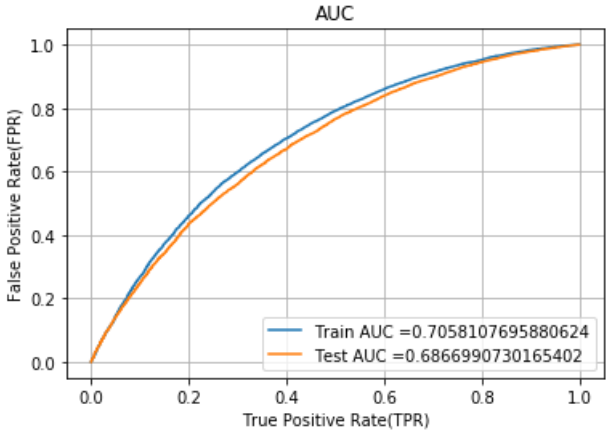
```
<matplotlib.axes._subplots.AxesSubplot at 0x1aafc25c90>
```



E) Train the model using the best hyper parameter value (L1)

In [267]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


model = SGDClassifier(loss='hinge', penalty='l1', alpha=best_alpha_l1)

model.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 = model.decision_function(X_tr)
y_test_pred = model.decision_function(X_te)

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

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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



F) Confusion Matrix (L1)

Train Data

In [268]:

In [268]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
('the maximum value of tpr*(1-fpr)', 0.25, 'for threshold', 0.045)
[[ 3713  3713]
 [ 8672 32943]]
```

In [269]:

```
conf_matr_df_train_3_l1 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred,
tr_thresholds, train_fpr, train_fpr)), range(2),range(2))
```

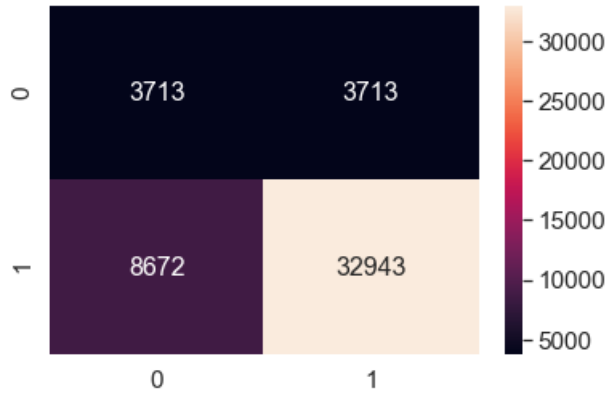
```
('the maximum value of tpr*(1-fpr)', 0.25, 'for threshold', 0.045)
```

In [377]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_3_l1, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[377]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1aafc259d0>
```



Test Data

In [270]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

Test confusion matrix
('the maximum value of tpr*(1-fpr)', 0.24999999161092998, 'for threshold', 0.765)
[[ 3367  2092]
 [10555 20038]]
```

In [271]:

```
conf_matr_df_test_3_l1 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds,
test_fpr, test_fpr)), range(2),range(2))
```

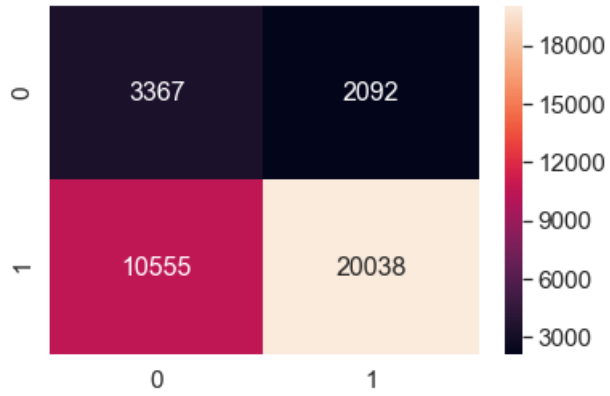
```
('the maximum value of tpr*(1-fpr)', 0.24999999161092998, 'for threshold', 0.765)
```

In [376]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_3_l1, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[376]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1aafd7bc90>
```



Set 4 : Categorical, Numerical features + Project_title(TFIDF W2V) + Preprocessed_essay (TFIDF W2V)

In [272]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr = hstack((categories_one_hot_train, sub_categories_one_hot_train,
school_state_categories_one_hot_train, project_grade_categories_one_hot_train,
teacher_prefix_categories_one_hot_train, price_standardized_train, quantity_standardized_train, pp
t_standardized_train, twc_standardized_train, ewc_standardized_train,pos_standardized_train,neg_st
andardized_train,neu_standardized_train, com_standardized_train, tfidf_w2v_vectors_train,
tfidf_w2v_vectors_titles_train)).tocsr()
X_te = hstack((categories_one_hot_test, sub_categories_one_hot_test,
school_state_categories_one_hot_test, project_grade_categories_one_hot_test,
teacher_prefix_categories_one_hot_test, price_standardized_test, quantity_standardized_test,
ppt_standardized_test, twc_standardized_test,
ewc_standardized_test,pos_standardized_test,neg_standardized_test,neu_standardized_test,
com_standardized_test, tfidf_w2v_vectors_test, tfidf_w2v_vectors_titles_test)).tocsr()
X_cr = hstack((categories_one_hot_cv, sub_categories_one_hot_cv,
school_state_categories_one_hot_cv, project_grade_categories_one_hot_cv,
teacher_prefix_categories_one_hot_cv,price_standardized_cv, quantity_standardized_cv,
ppt_standardized_cv, twc_standardized_cv,
ewc_standardized_cv,pos_standardized_cv,neg_standardized_cv,neu_standardized_cv,
com_standardized_cv, tfidf_w2v_vectors_cv, tfidf_w2v_vectors_titles_cv)).tocsr()
```

In [273]:

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

```
Final Data matrix
((49041, 709), (49041,))
((24155, 709), (24155,))
((36052, 709), (36052,))
====================================================================================================
```

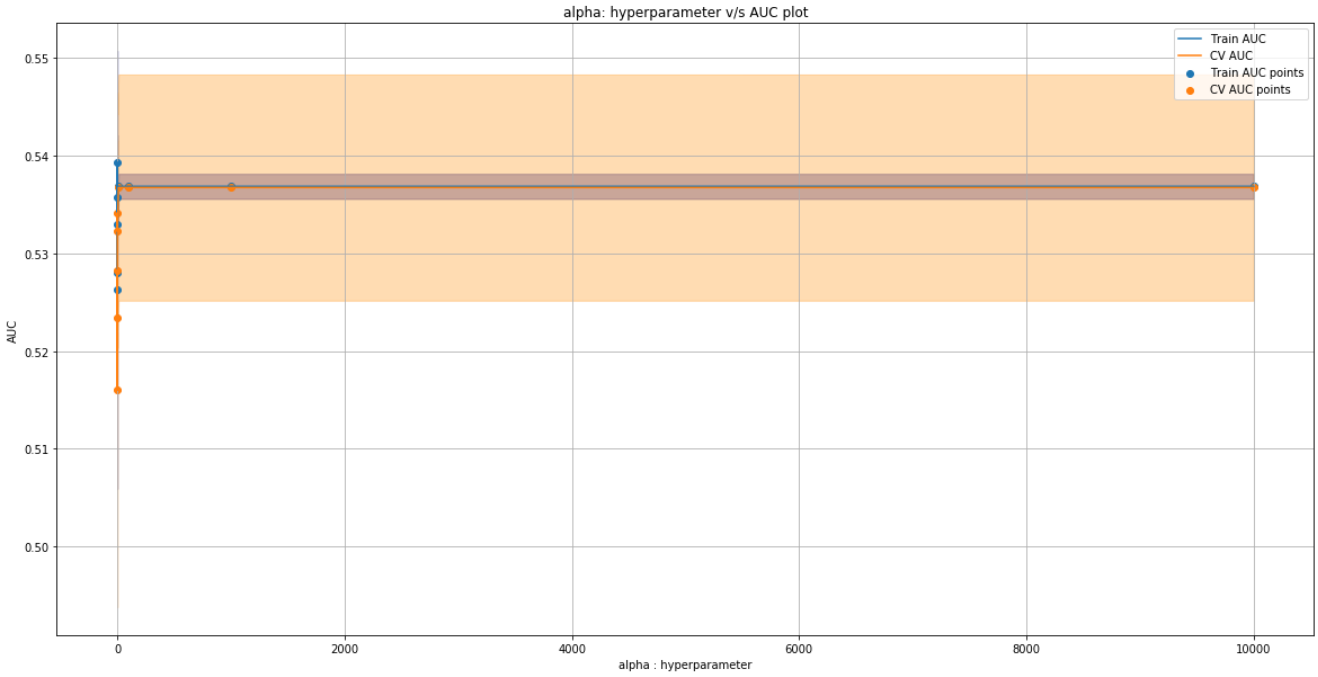
A) GridSearchCV - L2-Regularization

In [274]:

```
sv = SGDClassifier(loss='hinge', penalty='l2')

parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}

clf = GridSearchCV(sv, parameters, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [275]:

```
plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color=
'darkorange')

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


plt.legend()
plt.xlabel("alpha ")
plt.ylabel("AUC")
plt.title("alpha v/s AUC plot")
plt.grid()
plt.show()
```



Summary

1. Need to re-run the GridSearchCV on a smaller set of alpha to get more clarity

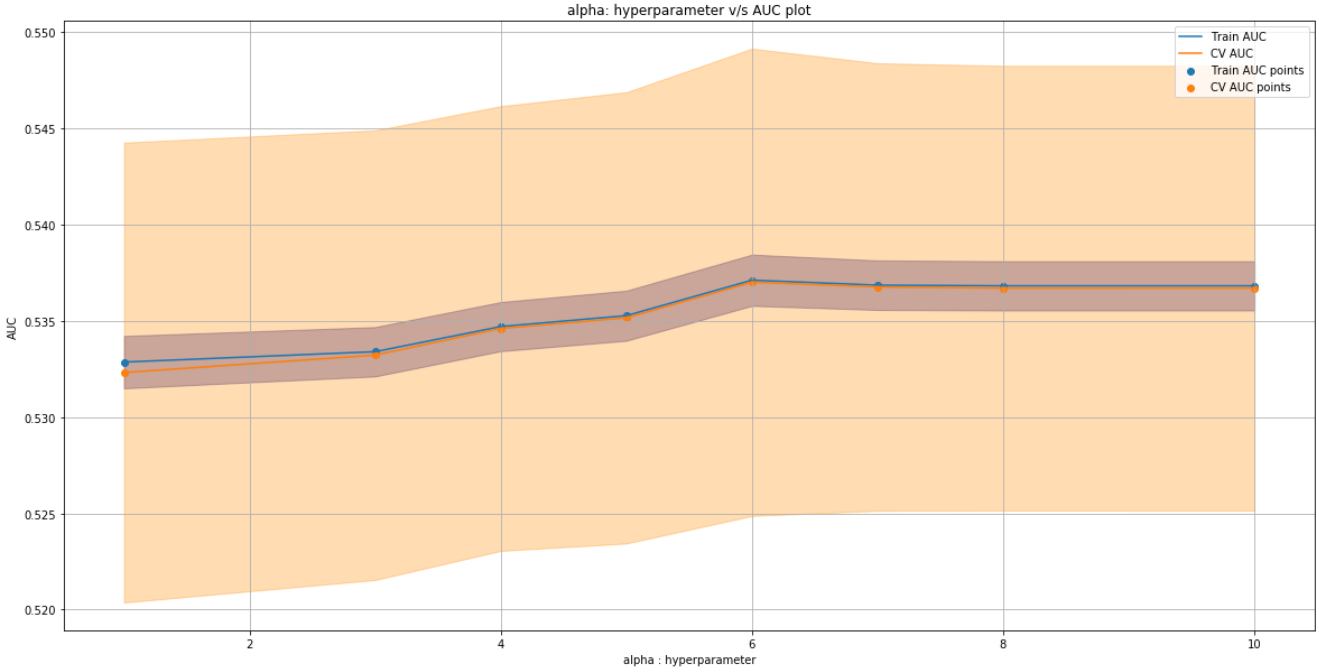
In [278]:

```
sv = SGDClassifier(loss='hinge', penalty='l2')
```

```
parameters = {'alpha':[1, 3, 4, 5, 6, 7, 8, 10]}

clf = GridSearchCV(sv, parameters, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [279]:

```
plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color=
'darkorange')

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


plt.legend()
plt.xlabel("alpha ")
plt.ylabel("AUC")
plt.title("alpha v/s AUC plot")
plt.grid()
plt.show()
```



Summary

- Alpha value 6 seems to be a better hyperparameter value when compared to the other hyperparameters.

In [ ]:

```
best_alpha_l2=6.0
```

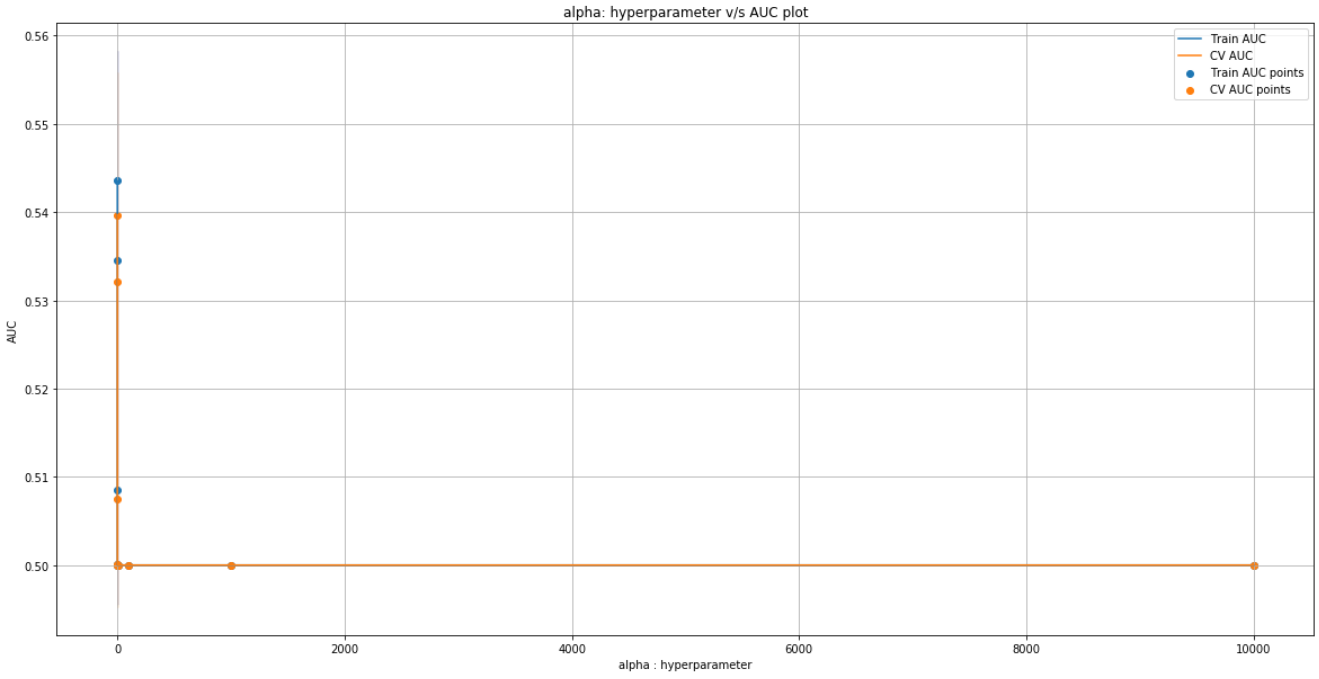
B) GridSearchCV - L1-Regularization

In [280]:

```
sv = SGDClassifier(loss='hinge', penalty='l1')

parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}

clf = GridSearchCV(sv, parameters, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [281]:

```
plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color=
'darkorange')

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


plt.legend()
plt.xlabel("alpha")
plt.ylabel("AUC")
plt.title("alpha v/s AUC plot")
plt.grid()
plt.show()
```



Summary

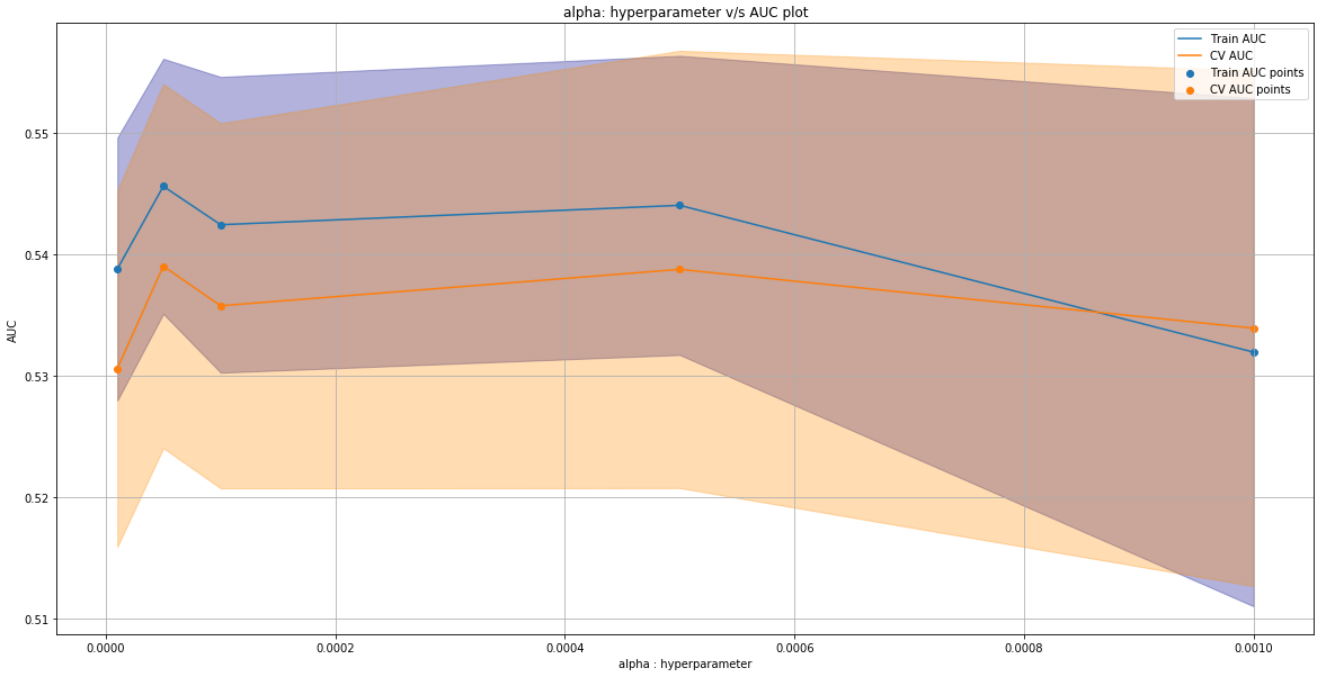
1. Need to re-run the GridSearchCV on a smaller set of parameter values.

In [284]:

```
sv = SGDClassifier(loss='hinge', penalty='l1')

parameters = {'alpha':[0.00001, 0.00005, 0.0001, 0.0005, 0.001]}

clf = GridSearchCV(sv, parameters, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [285]:

```
plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color=
'darkorange')

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


plt.legend()
plt.xlabel("alpha")
plt.ylabel("AUC")
plt.title("alpha v/s AUC plot")
plt.grid()
plt.show()
```



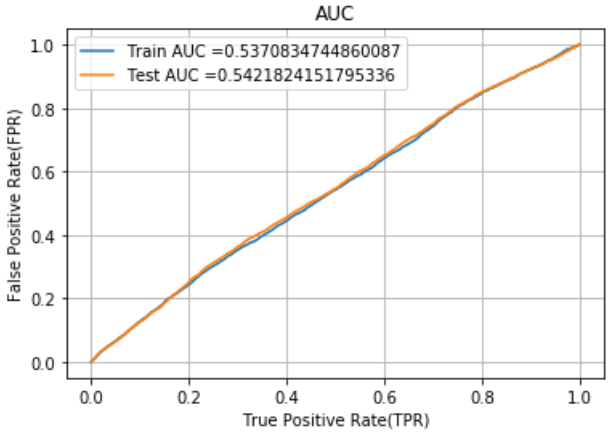
In [ ]:

```
best_alpha_l1=0.0005
```

C) Train the model using the best hyper parameter value (L2)

In [296]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


model = SGDClassifier(loss='hinge', penalty='l2', alpha= best_alpha_l2)

model.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 = model.decision_function(X_tr)
y_test_pred = model.decision_function(X_te)

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

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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



D) Confusion Matrix (L2)

Train Data

In [297]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
('the maximum value of tpr*(1-fpr)', 0.25, 'for threshold', 0.93)
[[ 3713  3713]
 [19045 22570]]
```

In [298]:

```
conf_matr_df_train_4_l2 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred,
tr_thresholds, train_fpr, train_fpr)), range(2),range(2))
```

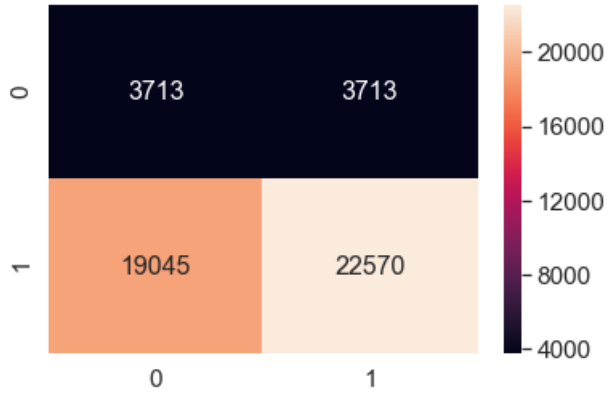
('the maximum value of tpr*(1-fpr)', 0.25, 'for threshold', 0.93)

In [374]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_4_l2, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[374]:

<matplotlib.axes._subplots.AxesSubplot at 0x1a86274310>



Test Data

In [299]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

Test confusion matrix
('the maximum value of tpr*(1-fpr)', 0.24999999161092995, 'for threshold', 0.938)
[[ 3088  2371]
 [15672 14921]]
```

In [300]:

```
conf_matr_df_test_4_l2 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds,
test_fpr, test_fpr)), range(2),range(2))
```

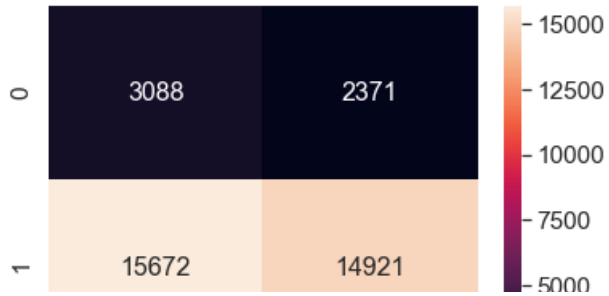
('the maximum value of tpr*(1-fpr)', 0.24999999161092995, 'for threshold', 0.938)

In [373]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_4_l2, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[373]:

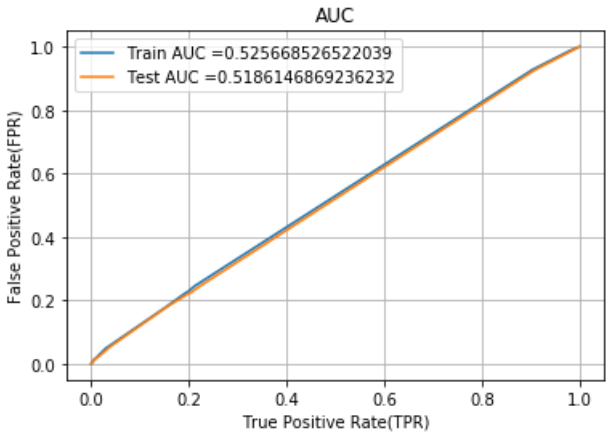
<matplotlib.axes._subplots.AxesSubplot at 0x1a8681e210>



E) Train the model using the best hyper parameter value (L1)

In [301]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


model = SGDClassifier(loss='hinge', penalty='l1', alpha=best_alpha_l1)

model.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 = model.decision_function(X_tr)
y_test_pred = model.decision_function(X_te)

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

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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



F) Confusion Matrix (L1)

Train Data

In [302]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
('the maximum value of tpr*(1-fpr)', 0.16834535982607168, 'for threshold', 1.016)
```

```
[[ 5835  1591]
 [31317 10298]]
```

In [303]:

```
conf_matr_df_train_4_l1 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred,
tr_thresholds, train_fpr, train_fpr)), range(2),range(2))
```

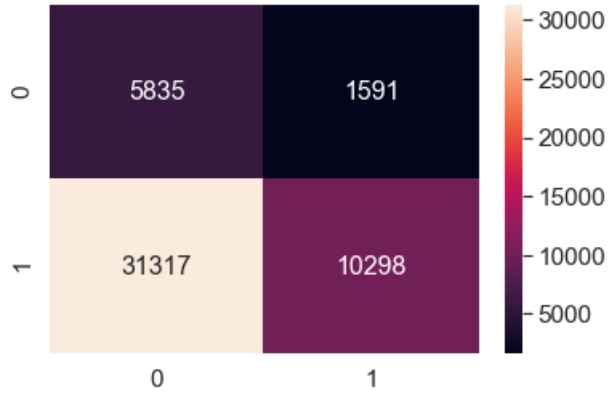
```
('the maximum value of tpr*(1-fpr)', 0.16834535982607168, 'for threshold', 1.016)
```

In [372]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_4_l1, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[372]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b302aab90>
```



Test Data

In [304]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

Test confusion matrix
('the maximum value of tpr*(1-fpr)', 0.17262451150025732, 'for threshold', 1.016)
[[ 4249  1210]
 [23141  7452]]
```

In [305]:

```
conf_matr_df_test_4_l1 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds,
test_fpr, test_fpr)), range(2),range(2))
```

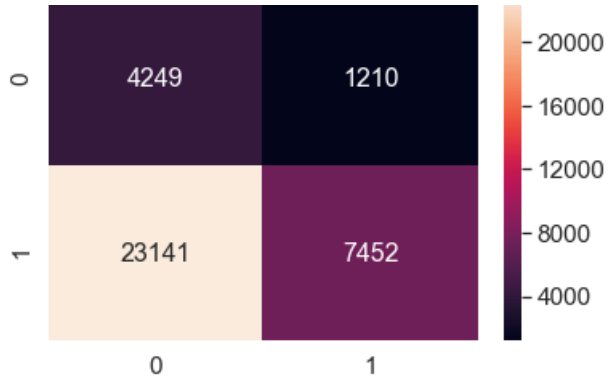
```
('the maximum value of tpr*(1-fpr)', 0.17262451150025732, 'for threshold', 1.016)
```

In [371]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_4_l1, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[371]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b31418750>
```



Set 5 : Categorical features, Numerical features by TruncatedSVD on TfidfVectorizer

A) Using Elbow method to narrow down the best number of Components

NOTE

- Dimensionality reduction using truncated SVD (aka LSA).

  This transformer performs linear dimensionality reduction by means of truncated singular value decomposition (SVD). Contrary to PCA, this estimator does not center the data before computing the singular value decomposition. This means it can work with scipy.sparse matrices efficiently.

  In particular, truncated SVD works on term count/tf-idf matrices as returned by the vectorizers in sklearn.feature_extraction.text. In that context, it is known as latent semantic analysis (LSA).

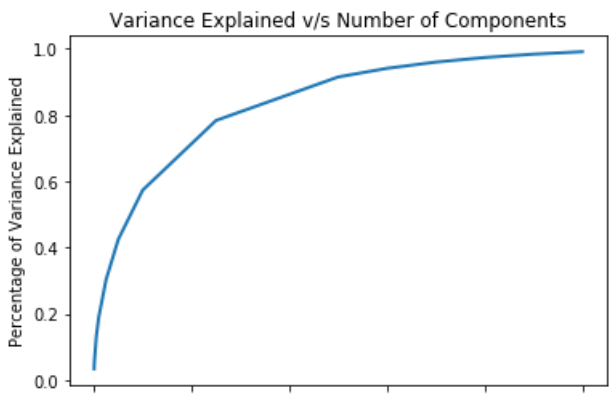
  This estimator supports two algorithms: a fast randomized SVD solver, and a “naive” algorithm that uses ARPACK as an eigensolver on (X *X.T) or (X.T* X), whichever is more efficient.

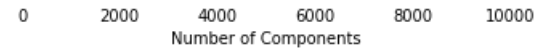
In [ ]:

```
## https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html
from sklearn.decomposition import TruncatedSVD
index = [1,5,10,50,100,250,500,1000,2000,5000,6000,7000,8000,9000,10000]
variance_sum = []

for i in tqdm(index):
    svd = TruncatedSVD(n_components= i, n_iter=7, random_state=42)
    svd.fit(text_tfidf_train)
    variance_sum.append(svd.explained_variance_ratio_.sum())
```

In [325]:

```
plt.xlabel("Number of Components")
plt.ylabel("Percentage of Variance Explained")
plt.title("Variance Explained v/s Number of Components")
plt.plot(index,variance_sum,lw=2)
plt.show()
```





Summary

- With increase in the number of components ,Percentage of Variance explained also increses, and after a certain point it tapers and becomes contant.
- We can see that at No. of components =6000 , 90% variance is explained

Training Data for SVD

In [331]:

```
svd = TruncatedSVD(n_components= 5000, n_iter=7, random_state=42)
svd.fit(text_tfidf_train)
svd_train = svd.transform(text_tfidf_train)
```

In [332]:

```
print("Shape of matrix after Decomposition ",svd_train.shape)
```

```
('Shape of matrix after Decomposition ', (49041, 5000))
```

Test Data for SVD

In [333]:

```
svd_test = svd.transform(text_tfidf_test)
print("Shape of matrix after Decomposition ",svd_test.shape)
```

```
('Shape of matrix after Decomposition ', (36052, 5000))
```

Cross Validation Data for SVD

In [334]:

```
svd_cv = svd.transform(text_tfidf_cv)
print("Shape of matrix after Decomposition ",svd_cv.shape)
```

```
('Shape of matrix after Decomposition ', (24155, 5000))
```

In [335]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack


X_tr = hstack((categories_one_hot_train, sub_categories_one_hot_train,
school_state_categories_one_hot_train, project_grade_categories_one_hot_train,
teacher_prefix_categories_one_hot_train, price_standardized_train, quantity_standardized_train, pp
t_standardized_train, twc_standardized_train, ewc_standardized_train, pos_standardized_train,neg_s
tandardized_train,neu_standardized_train,com_standardized_train,svd_train)).tocsr()
X_te = hstack((categories_one_hot_test, sub_categories_one_hot_test,
school_state_categories_one_hot_test, project_grade_categories_one_hot_test,
teacher_prefix_categories_one_hot_test, price_standardized_test, quantity_standardized_test,
ppt_standardized_test, twc_standardized_test,
ewc_standardized_test,pos_standardized_test,neg_standardized_test,neu_standardized_test,com_standar
dized_test,svd_test)).tocsr()
X_cr = hstack((categories_one_hot_cv, sub_categories_one_hot_cv,
school_state_categories_one_hot_cv, project_grade_categories_one_hot_cv,
teacher_prefix_categories_one_hot_cv,price_standardized_cv, quantity_standardized_cv,
ppt_standardized_cv, twc_standardized_cv, ewc_standardized_cv, pos_standardized_cv,neg_standardized
_cv,neu_standardized_cv,com_standardized_cv,svd_cv)).tocsr()
```

In [336]:

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

```
Final Data matrix
((49041, 5109), (49041,))
((24155, 5109), (24155,))
((36052, 5109), (36052,))
====================================================================================================
```

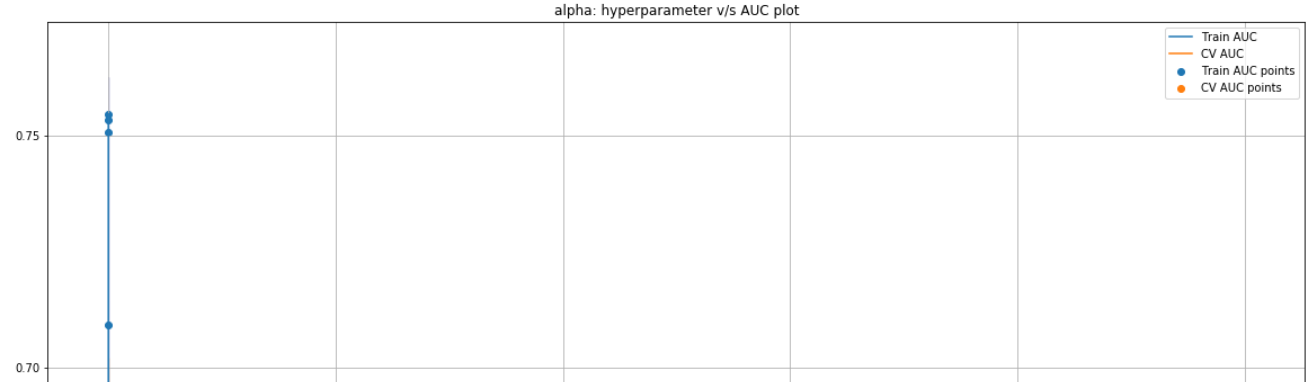
A) GridSearchCV - L2-Regularization

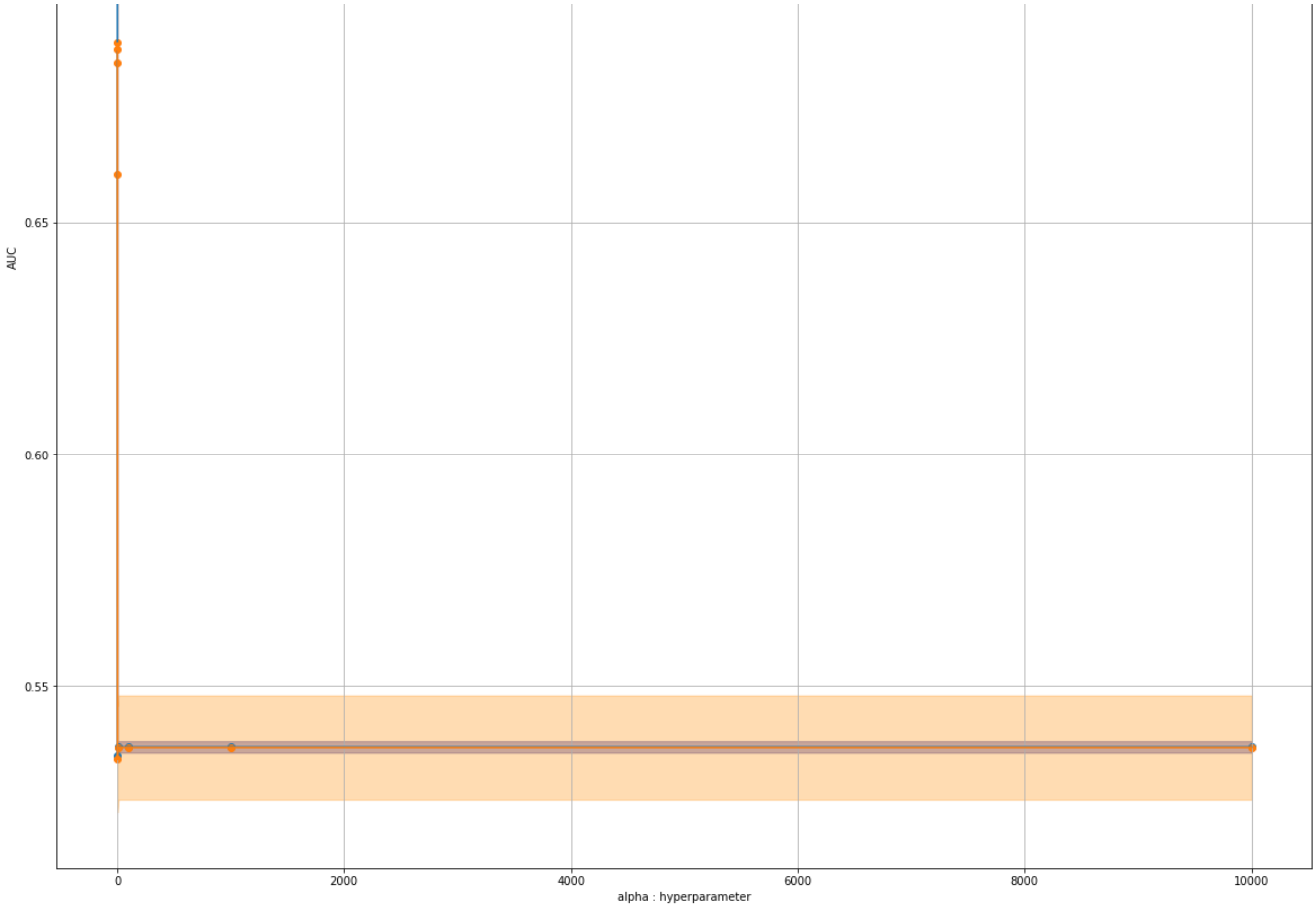
In [337]:

```
sv = SGDClassifier(loss='hinge', penalty='l2')

parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}

clf = GridSearchCV(sv, parameters, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [341]:

```
plt.figure(figsize=(20,20))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color=
'darkorange')

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


plt.legend()
plt.xlabel("alpha ")
plt.ylabel("AUC")
plt.title("alpha v/s AUC plot")
plt.grid()
plt.show()
```





Summary

1. Need to re-run the GridSearchCV on a smaller set of alpha for more clarity.
2. Alpha values in the range of 0.1 to 1 seems to be a suitable range

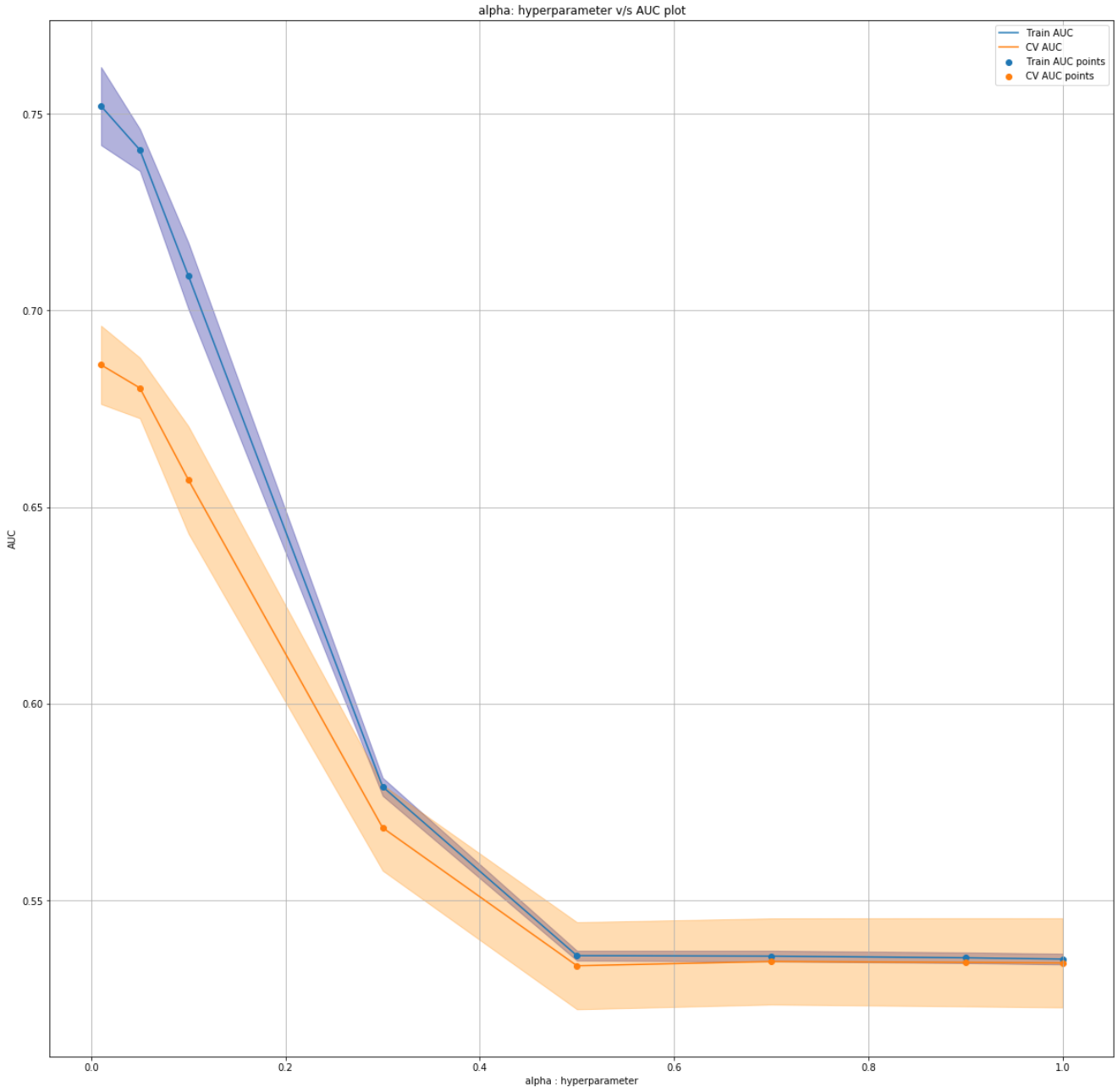
In [342]:

```
sv = SGDClassifier(loss='hinge', penalty='l2')

parameters = {'alpha':[0.01, 0.05, 0.1, 0.3, 0.5, 0.7, 0.9, 1.0]}

clf = GridSearchCV(sv, parameters, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [343]:

```
plt.figure(figsize=(20,20))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color=
'darkorange')

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


plt.legend()
```

```
plt.xlabel("Alpha")
plt.ylabel("AUC")
plt.title("Alpha vs AUC plot")
plt.grid()
plt.show()
```



Summary

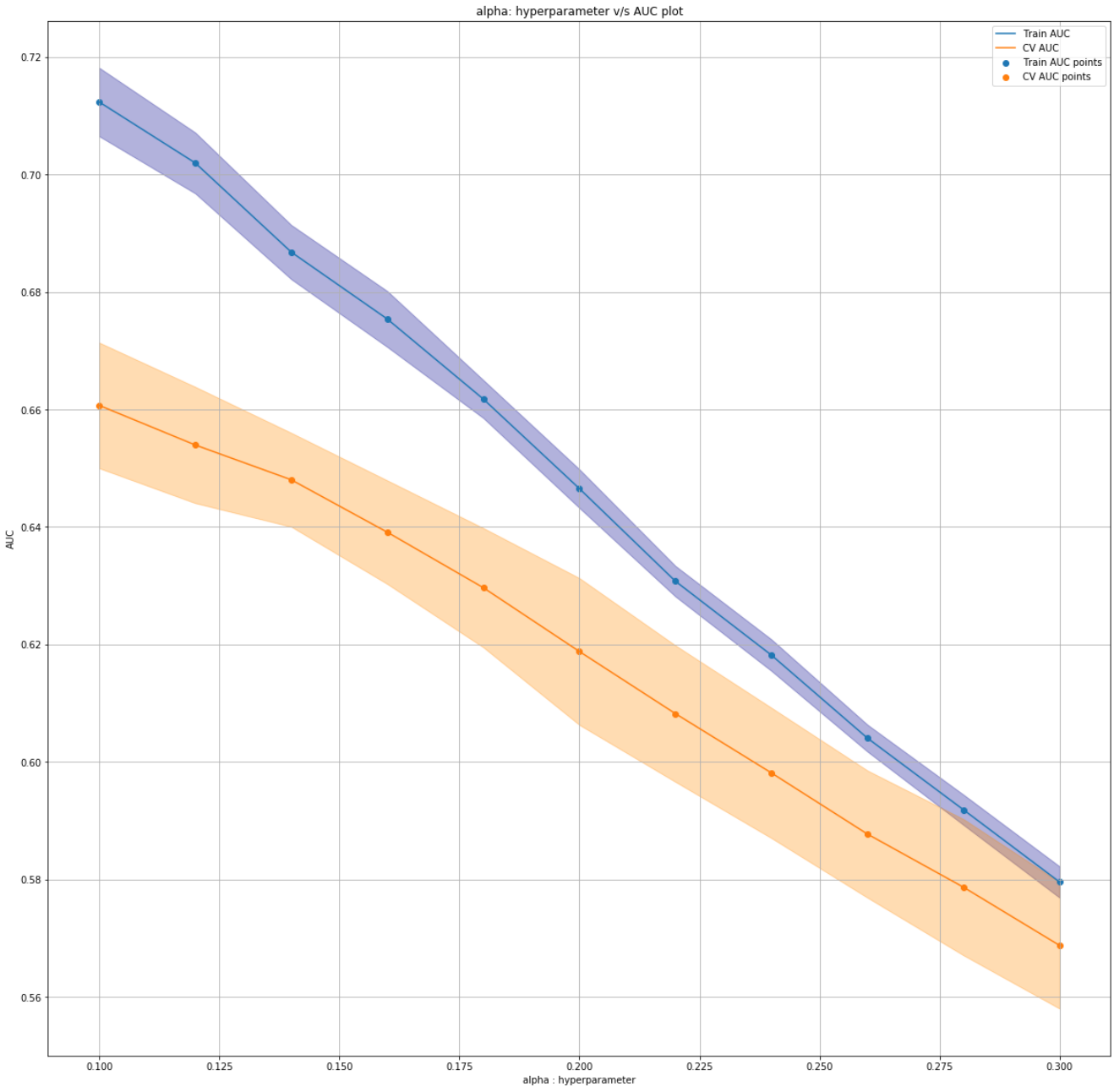
1. Need to re-run the GridSearchCV on a smaller set of alpha to get more clarity
2. Alpha values in the range of 0.1 to 0.5 seems to be a suitable range.

In [344]:

```
sv = SGDClassifier(loss='hinge', penalty='l2')

parameters = {'alpha':[0.10, 0.12, 0.14, 0.16, 0.18, 0.2, 0.22, 0.24, 0.26, 0.28, 0.3]}

clf = GridSearchCV(sv, parameters, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [345]:

```
plt.figure(figsize=(20,20))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color=
'darkorange')

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


plt.legend()
plt.xlabel("alpha )
plt.ylabel("AUC")
plt.title("alpha v/s AUC plot")
plt.grid()
plt.show()
```



In [ ]:

```
best_alpha_l2=0.2
```

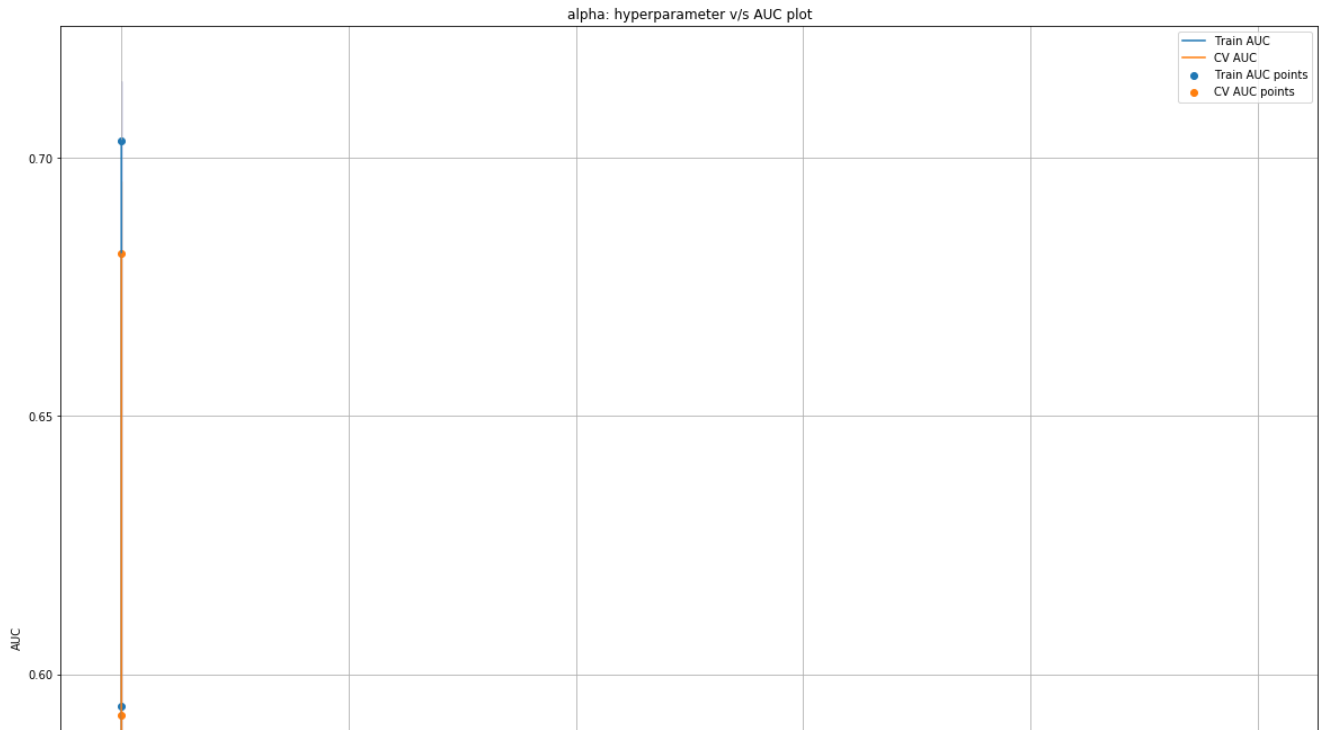
B) GridSearchCV - L1-Regularization

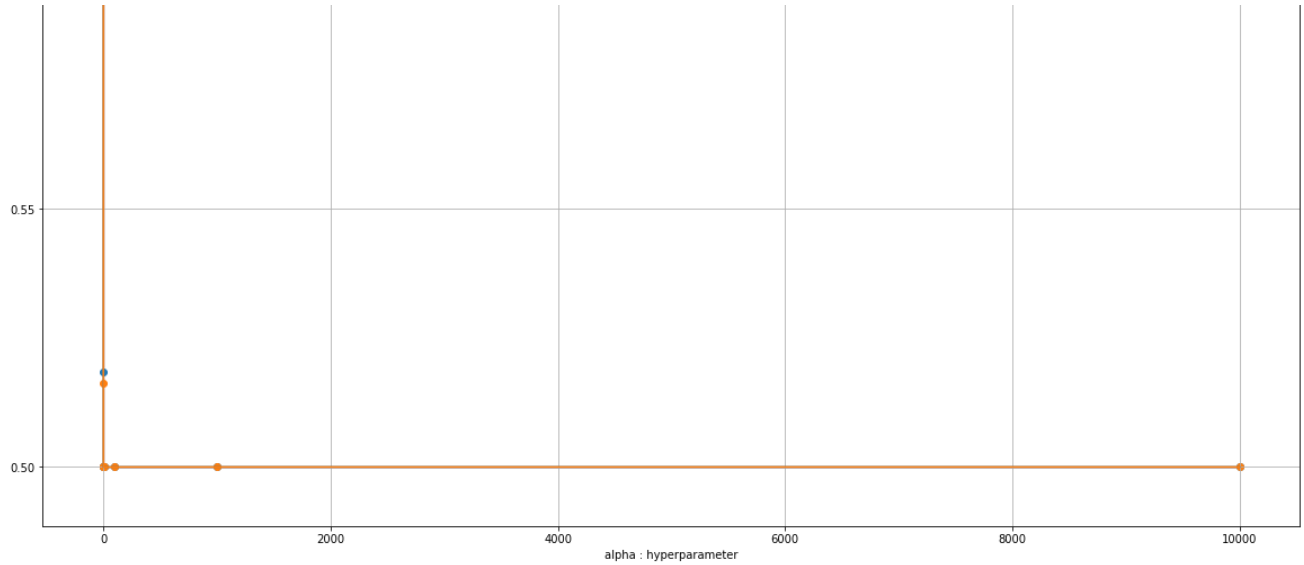
In [346]:

```
sv = SGDClassifier(loss='hinge', penalty='l1')

parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}

clf = GridSearchCV(sv, parameters, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [347]:

```
plt.figure(figsize=(20,20))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color=
'darkorange')

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


plt.legend()
plt.xlabel("Alpha")
plt.ylabel("AUC")
plt.title("Alpha v/s AUC plot")
plt.grid()
plt.show()
```





Summary

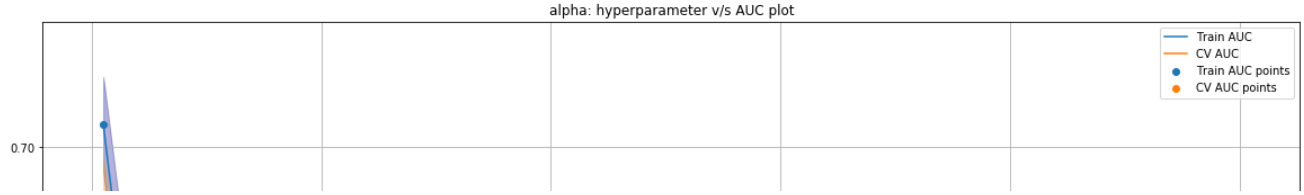
1. Need to re-run the GridSearchCV on a smaller set of alpha to get more clarity.

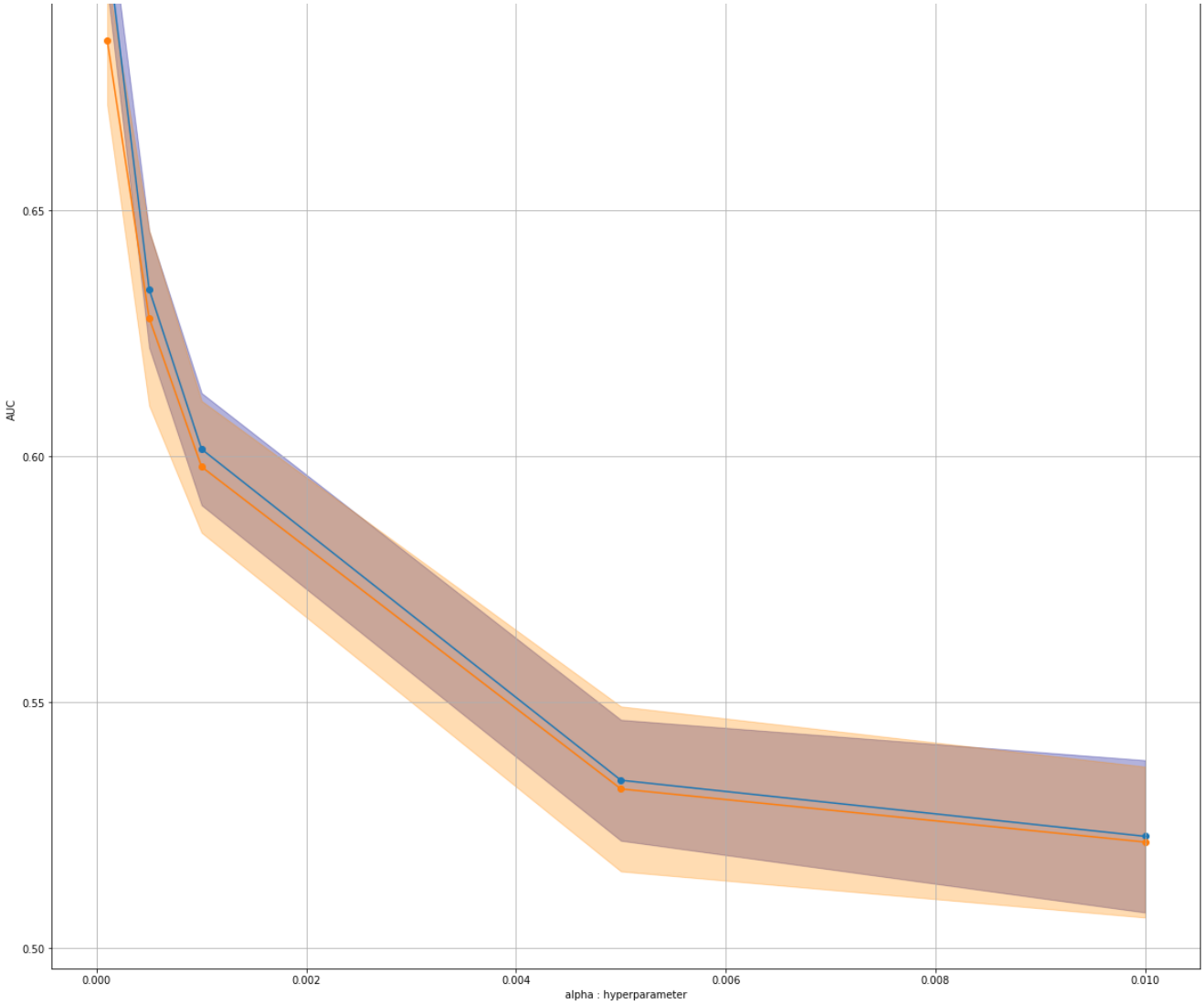
In [348]:

```
sv = SGDClassifier(loss='hinge', penalty='l1')

parameters = {'alpha':[0.0001, 0.0005, 0.001, 0.005, 0.01]}

clf = GridSearchCV(sv, parameters, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [349]:

```
plt.figure(figsize=(20,20))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color=
'darkorange')

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


plt.legend()
plt.xlabel("alpha : hyperparameter")
plt.ylabel("AUC")
plt.title("alpha: hyperparameter v/s AUC plot")
plt.grid()
plt.show()
```





In [ ]:

```
best_alpha_l1=0.0005
```

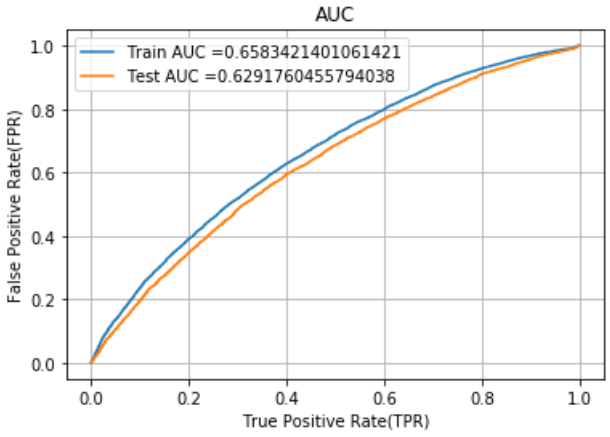
BOTH MODELS ARE EQUALLY GOOD

C) Train the model using the best hyper parameter value (L2)

In [351]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


model = SGDClassifier(loss='hinge', penalty='l2', alpha= best_alpha_l2)

model.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 = model.decision_function(X_tr)
y_test_pred = model.decision_function(X_te)

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

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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



D) Confusion Matrix (L2)

Train Data

In [352]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
('the maximum value of tpr*(1-fpr)', 0.25, 'for threshold', 1.0)
[[ 3713  3713]
 [11677 29938]]
```

In [353]:

```
conf_matr_df_train_5_l2 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred,
tr_thresholds, train_fpr, train_fpr)), range(2),range(2))
```

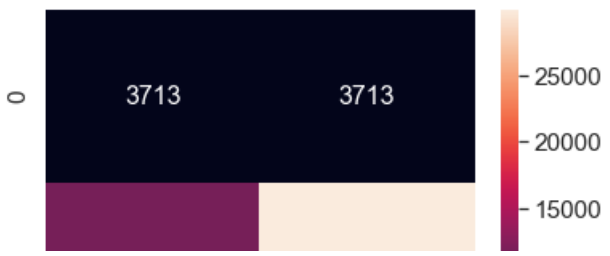
```
('the maximum value of tpr*(1-fpr)', 0.25, 'for threshold', 1.0)
```

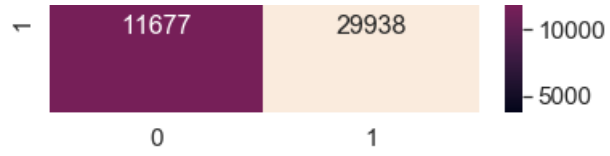
In [370]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_5_l2, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[370]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b30c52bd0>
```





Test Data

In [354]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

Test confusion matrix
('the maximum value of tpr*(1-fpr)', 0.24999999161092998, 'for threshold', 1.0)
[[ 3355  2104]
 [12963 17630]]
```

In [355]:

```
conf_matr_df_test_5_l2 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds,
test_fpr, test_fpr)), range(2),range(2))
```

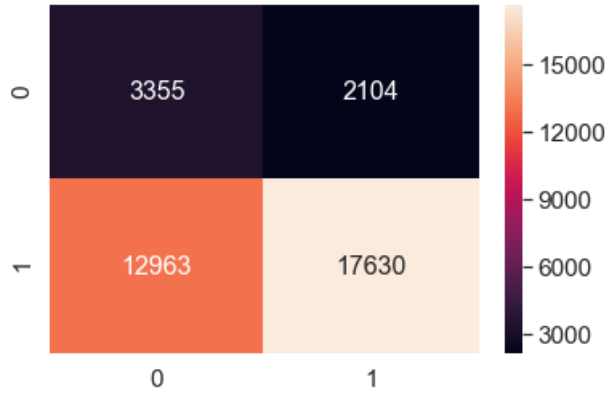
```
('the maximum value of tpr*(1-fpr)', 0.24999999161092998, 'for threshold', 1.0)
```

In [369]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_5_l2, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[369]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b30552c50>
```

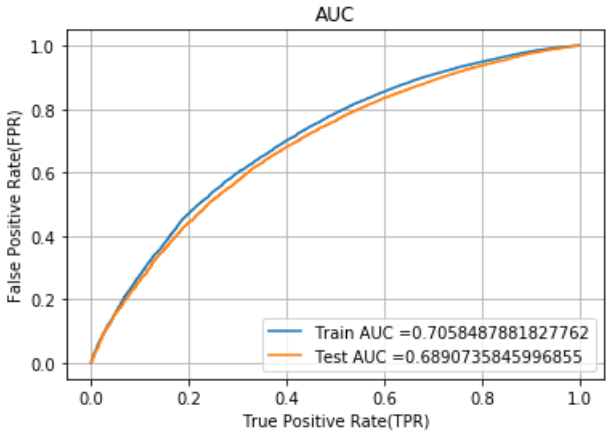


E) Train the model using the best hyper parameter value (L1)

In [358]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


model = SGDClassifier(loss='hinge', penalty='l1', alpha=best_alpha_l1)

model.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 = model.decision_function(X_tr)
y_test_pred = model.decision_function(X_te)

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

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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



F) Confusion Matrix (L1)

Train Data

In [359]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
('the maximum value of tpr*(1-fpr)', 0.25, 'for threshold', 1.048)
[[ 3713  3713]
 [ 8870 32745]]
```

In [360]:

```
conf_matr_df_train_5_l1 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred,
tr_thresholds, train_fpr, train_fpr)), range(2),range(2))
```

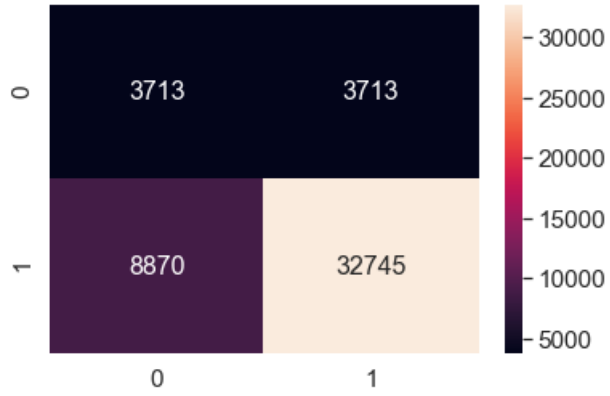
```
('the maximum value of tpr*(1-fpr)', 0.25, 'for threshold', 1.048)
```

In [368]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_5_l1, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[368]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b2fd1a610>
```



Test Data

In [361]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

Test confusion matrix
('the maximum value of tpr*(1-fpr)', 0.24999999161092998, 'for threshold', 1.1)
[[ 3457  2002]
 [10734 19859]]
```

In [362]:

```
conf_matr_df_test_5_l1 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds,
test_fpr, test_fpr)), range(2),range(2))
```

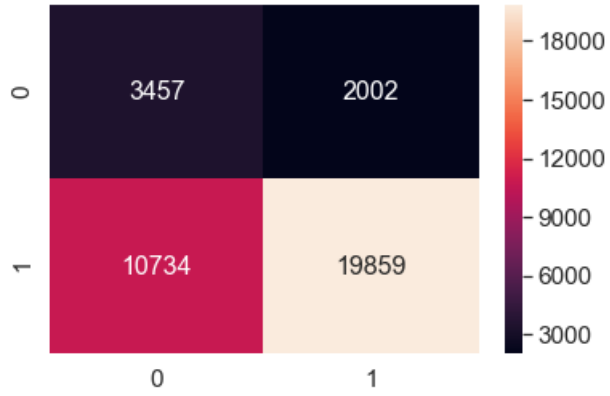
```
('the maximum value of tpr*(1-fpr)', 0.24999999161092998, 'for threshold', 1.1)
```

In [367]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_5_l1, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[367]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a86e9f650>
```



3. Conclusion

In [1]:

```
# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable

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

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Penalty","Alpha:Hyper Parameter", "AUC"]

x.add_row(["BOW", "Linear SVM(SGD)","L2 performs slightly better than L1", 0.3, 0.647])
x.add_row(["TFIDF", "Linear SVM(SGD)", "L1 performs better than L2",0.0001, 0.667])
x.add_row(["AVG W2V", "Linear SVM(SGD)", "L1 & L2 both works good", "L1:0.00005 & L2:0.005",
"L1:0.686 & L2:0.67"])
x.add_row(["TFIDF W2V", "Linear SVM(SGD)", "Neither L1 or L2 performs good", "L1:0.0005 & L2:6.0",
"L1:0.518 & L2:0.542"])
x.add_row(["TRUNCATED SVD", "Linear SVM(SGD)", "L1 & L2 both behave similarly", "L1:0.0001 &
L2:0.18", "L1:0.69 & L2:0.63"])


print(x)
```

```
+---------------+-----------------+-------------------------------------+-----------------------+--
---------------+
|   Vectorizer  |      Model      |               Penalty               | Alpha:Hyper Parameter |
AUC          |
+---------------+-----------------+-------------------------------------+-----------------------+--
---------------+
|      BOW      | Linear SVM(SGD) | L2 performs slightly better than L1 |          0.3          |
0.647        |
|     TFIDF     | Linear SVM(SGD) |      L1 performs better than L2     |         0.0001        |
0.667        |
|    AVG W2V    | Linear SVM(SGD) |       L1 & L2 both works good       | L1:0.00005 & L2:0.005 |
L1:0.686 & L2:0.67 |
|   TFIDF W2V   | Linear SVM(SGD) |    Neither L1 or L2 performs good   |   L1:0.0005 & L2:6.0  | I
1:0.518 & L2:0.542 |
| TRUNCATED SVD | Linear SVM(SGD) |    L1 & L2 both behave similarly    |  L1:0.0001 & L2:0.18  |
L1:0.69 & L2:0.63  |
+---------------+-----------------+-------------------------------------+-----------------------+--
---------------+
```

SUMMARY NOTES:

- SVM is a support-vector machine which is a special linear-model. From a theoretical view it's a convex-optimization problem and we can get the global-optimum in polynomial-time.
- SVC (libsvm) and LinearSVC (liblinear) make different assumptions in regards to the optimization-problem, which results in different performances on the same task (linear-kernel: LinearSVC much more efficient than SVC in general; but some tasks can't be tackled by LinearSVC).
- SGD is an Stochastic Gradient Descent-based (this is a general optimization method!) optimizer which can optimize many different convex-optimization problems
- sklearn says: Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to discriminative learning of linear classifiers under convex loss functions. Now it's actually even more versatile, but here it's enough to note that it subsumes (some) SVMs, logistic-regression and others.

1. Specialized SVM-methods

- scale worse with the number of samples
- do not need hyper-parameter tuning

2. SGD-based methods

- scale better for huge-data in general
- need hyper-parameter tuning
- solve only a subset of the tasks approachable by the the above (no kernel- methods!)

NOTE:

- Don't use Accuracy to find Optimal hyperparameter , as this is imbalanced data.
- You can get the optimal hyperparameter from GridSearch or RandomisedSearch from ROC-AUC curve