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`

Feature	Description
`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 (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Postal_codes)). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy` • `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • `My students need hands on literacy materials to manage sensory needs!`
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*
`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`

Feature	Description
teacher_prefix	Teacher's title. One of the following enumerated values: - nan - Dr. - Mr. - Mrs. - Ms. - Teacher.
teacher_number_of_previously_posted_projects	Number of project applications previously submitted by the same teacher. **Example:** `2`

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

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

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

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

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

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

Notes on the Essay Data

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

- __project_essay_1:__ "Introduce us to your classroom"
- __project_essay_2:__ "Tell us more about your students"
- __project_essay_3:__ "Describe how your students will use the materials you're requesting"

- __project_essay_3:__ "Close by sharing why your project will make a difference"

  Starting on May 17, 2016, the number of essays was reduced from 4 to 2, and the prompts for the first 2 essays were changed to the following:
- __project_essay_1:__ "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."
- __project_essay_2:__ "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

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

In [1]:

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

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

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

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

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

from tqdm import tqdm
import os

from chart_studio import plotly
import plotly.offline as offline
import chart_studio.plotly
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
#pip install chart-studio==1.0.0
```

1.1 Reading Data

In [2]:

```
project_data = pd.read_csv("C:\\Users\\Admin\\Assignments-ML\\Assignment-T-SNE\\train_data.csv")
resource_data = pd.read_csv("C:\\Users\\Admin\\Assignments-ML\\Assignment-T-SNE\\resources.csv")
```

In [3]:

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

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

In [4]:

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


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


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


project_data.head(2)
```

Out[4]:

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

In [5]:

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

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

Out[5]:

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

1.2 preprocessing of `project_subject_categories`

In [6]:

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

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

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [7]:

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

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

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

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

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

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

1.3 Text preprocessing

In [8]:

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

In [9]:

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

Out[9]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	Engine STEAI the Pr Class
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	Se Toc F

In [10]:

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

In [11]:

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

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

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

In [12]:

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

In [13]:

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

In [14]:

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

In [15]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their'
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'tho
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', '
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', '
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before',
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again'
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'f
            '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', '
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't",
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mus
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'were
            'won', "won't", 'wouldn', "wouldn't"]
```

In [16]:

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

```
100%|████████████████████████████████████████████████████████████████████| 109248/109248 [01:27<00:00,
1244.81it/s]
```

In [17]:

```
project_data['clean_essays']= preprocessed_essays
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
project_data.drop(['project_essay_4'], axis=1, inplace=True)
```

In [18]:

```
project_data.head(3)
```

Out[18]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	Engine STEAI the Pr Class
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	Se Toc F
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	2016-04-27 00:46:53	Grades PreK-2	M Lea \ M List C

1.4 Preprocessing of `project_title`

In [19]:

```
# similarly you can preprocess the titles also
project_data.head(2)
```

Out[19]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	Engine STEAI the Pr Class
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	Se Toc F

In [20]:

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

```
Engineering STEAM into the Primary Classroom
==================================================
```

In [21]:

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

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

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

```
Health Nutritional Cooking in Kindergarten
==================================================
```

In [22]:

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

```
100%|████████████████████████████████████████████████████████████████████| 109248/109248 [00:04<00:00, 2
7303.16it/s]
```

In [23]:

```
project_data['clean_titles'] = preprocessed_titles
```

In [24]:

```
project_data['teacher_prefix']= project_data.teacher_prefix.fillna(' ')
```

In [25]:

```
#Preprocessing the project_grade_category
project_grade_category_clean=[]
for grade in tqdm(project_data['project_grade_category'].values):
    grade = grade.replace(' ', '_')
    project_grade_category_clean.append(grade)
project_data['project_grade_category']=project_grade_category_clean
#print(project_grade_category_clean)
```

```
100%|██████████████████████████████████████████████████████████████████████| 109248/109248 [00:00<00:00, 93
8697.03it/s]
```

In [26]:

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

1.5 Preparing data for models

In [27]:

```
project_data.shape
```

Out[27]:

```
(109248, 18)
```

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

In [28]:

```
data=project_data.head(50000)
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
X.head(1)
print(X.shape)
#print(y.shape)
```

```
(50000, 17)
```

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

In [29]:

```
# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
#X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)

print()

print(X_train.shape)
print(X_test.shape)

#X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
print(len(y_train))
print(len(y_test))
df_y_train=pd.DataFrame(data=y_train,columns=['project_is_approved'])

df_y_train['project_is_approved'].value_counts()

#df_y_test=pd.DataFrame(data=y_test,columns=['project_is_approved'])

#df_y_test['project_is_approved'].value_counts()
#df_y_test['project_is_approved'].value_counts()
```

```
(33500, 17)
(16500, 17)
33500
16500
```

Out[29]:

```
1    28135
0     5365
Name: project_is_approved, dtype: int64
```

BOW vectorization of essays

In [30]:

```
print(X_train.shape, y_train.shape)
#print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)


vectorizer = CountVectorizer(min_df=10,ngram_range=(1,2), max_features=5000)
vectorizer.fit(X_train['clean_essays'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['clean_essays'].values)
#X_cv_essay_bow = vectorizer.transform(X_cv['clean_essays'].values)
X_test_essay_bow = vectorizer.transform(X_test['clean_essays'].values)

print("After vectorizations of essay")
print(X_train_essay_bow.shape, y_train.shape)
#print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
print("="*100)
print("NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME")
```

```
(33500, 17) (33500,)
(16500, 17) (16500,)
====================================================================================================
After vectorizations of essay
(33500, 5000) (33500,)
(16500, 5000) (16500,)
====================================================================================================
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

TFIDF vectorization of essays

In [31]:

```python
print(X_train.shape, y_train.shape)
#print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)


vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,2), max_features=5000)
vectorizer.fit(X_train['clean_essays'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer.transform(X_train['clean_essays'].values)
#X_cv_essay_tfidf = vectorizer.transform(X_cv['clean_essays'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['clean_essays'].values)

print("After TFIDF vectorizations of essay")
print(X_train_essay_tfidf.shape, y_train.shape)
#print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
print("NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME")
```

```
(33500, 17) (33500,)
(16500, 17) (16500,)
====================================================================================================
After TFIDF vectorizations of essay
(33500, 5000) (33500,)
(16500, 5000) (16500,)
====================================================================================================
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

avg_w2v_title vectorization of Essays

In [32]:

```
with open("C:\\Users\\Admin\\Assignments-ML\\Assignment-T-SNE\\glove_vectors", 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())


# average Word2Vec
# compute average word2vec for each review.
X_train_essay_avg_w2v_vec = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_essays'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_train_essay_avg_w2v_vec.append(vector)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 33500/33500 [00:13<00:00,
2459.19it/s]
```

In [33]:

```
with open("C:\\Users\\Admin\\Assignments-ML\\Assignment-T-SNE\\glove_vectors", 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())


# average Word2Vec
# compute average word2vec for each review.
X_test_essay_avg_w2v_vec = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_essays'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_test_essay_avg_w2v_vec.append(vector)
```

```
100%|████████████████████████████████████████████████████| 16500/16500 [00:06<00:00,
2458.96it/s]
```

tfidf_w2v_vec vectorization of essays

In [34]:

```
tfidf_model = TfidfVectorizer()
#tfidf_model.fit(preprocessed_essays)
tfidf_model.fit(X_train['clean_essays'].values)
# 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())

# average Word2Vec0
# compute average word2vec for each review.
X_train_essay_tfidf_w2v_vec = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_essays'].values): # 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
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_train_essay_tfidf_w2v_vec.append(vector)
```

```
100%|██████████████████████████████████████████████████| 33500/33500 [01:36<00:00,
348.32it/s]
```

In [35]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['clean_essays'].values)
# 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())

# average Word2Vec0
# compute average word2vec for each review.
X_test_essay_tfidf_w2v_vec = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_essays'].values): # 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
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_essay_tfidf_w2v_vec.append(vector)
```

```
100%|██████████████████████████████████████████████████| 16500/16500 [00:46<00:00,
353.71it/s]
```

BOW vectorizer titles

In [36]:

```
print(X_train.shape, y_train.shape)
#print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)


vectorizer = CountVectorizer(min_df=10,ngram_range=(1,2), max_features=5000)
vectorizer.fit(X_train['clean_titles'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_bow = vectorizer.transform(X_train['clean_titles'].values)
#X_cv_title_bow = vectorizer.transform(X_cv['clean_titles'].values)
X_test_title_bow = vectorizer.transform(X_test['clean_titles'].values)

print("After BOW vectorizations of titles")
print(X_train_title_bow.shape, y_train.shape)
#print(X_cv_title_bow.shape, y_cv.shape)
print(X_test_title_bow.shape, y_test.shape)
print("="*100)
```

```
(33500, 17) (33500,)
(16500, 17) (16500,)
====================================================================================================
After BOW vectorizations of titles
(33500, 2707) (33500,)
(16500, 2707) (16500,)
====================================================================================================
```

TFIDF vectorization of titles

In [37]:

```
print(X_train.shape, y_train.shape)
#print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)


vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,2), max_features=5000)
vectorizer.fit(X_train['clean_titles'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_tfidf = vectorizer.transform(X_train['clean_titles'].values)
#X_cv_title_tfidf = vectorizer.transform(X_cv['clean_titles'].values)
X_test_title_tfidf = vectorizer.transform(X_test['clean_titles'].values)

print("After TFIDF vectorizations of titles")
print(X_train_title_tfidf.shape, y_train.shape)
#print(X_cv_title_tfidf.shape, y_cv.shape)
print(X_test_title_tfidf.shape, y_test.shape)
print("="*100)
```

```
(33500, 17) (33500,)
(16500, 17) (16500,)
====================================================================================================
After TFIDF vectorizations of titles
(33500, 2707) (33500,)
(16500, 2707) (16500,)
====================================================================================================
```

avg_w2v_title vectorization of titles

In [38]:

```
with open("C:\\Users\\Admin\\Assignments-ML\\Assignment-T-SNE\\glove_vectors", 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())


# average Word2Vec
# compute average word2vec for each review.
X_train_title_avg_w2v_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_titles'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_train_title_avg_w2v_title.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████████████| 33500/33500 [00:00<00:00, 4
7214.60it/s]
```

In [39]:

```python
with open("C:\\Users\\Admin\\Assignments-ML\\Assignment-T-SNE\\glove_vectors", 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())


# average Word2Vec
# compute average word2vec for each review.
X_test_title_avg_w2v_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_titles'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_test_title_avg_w2v_title.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████████████| 16500/16500 [00:00<00:00, 4
6160.79it/s]
```

tfidf_w2v_vec vectorization of titles

In [40]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['clean_titles'].values)
# 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())

# average Word2Vec0
# compute average word2vec for each review.
X_train_title_tfidf_w2v_vec = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_titles'].values): # 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
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_train_title_tfidf_w2v_vec.append(vector)
```

```
100%|██████████████████████████████████████████████████| 33500/33500 [00:01<00:00, 2
1770.54it/s]
```

In [41]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['clean_titles'].values)
# 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())

# average Word2Vec0
# compute average word2vec for each review.
X_test_title_tfidf_w2v_vec = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_titles'].values): # 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
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_title_tfidf_w2v_vec.append(vector)
```

```
100%|██████████████████████████████████████████████████████████| 16500/16500 [00:00<00:00, 2
2494.49it/s]
```

In [42]:

```
print(X_train.shape, y_train.shape)
#print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)

vectorizer = CountVectorizer(min_df=10,ngram_range=(1,2), max_features=5000)
vectorizer.fit(X_train['project_resource_summary'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_resource_bow = vectorizer.transform(X_train['project_resource_summary'].values)
#X_cv_resource_bow = vectorizer.transform(X_cv['project_resource_summary'].values)
X_test_resource_bow = vectorizer.transform(X_test['project_resource_summary'].values)

print("After vectorizations of project_resource")
print(X_train_resource_bow.shape, y_train.shape)
#print(X_cv_resource_bow.shape, y_cv.shape)
print(X_test_resource_bow.shape, y_test.shape)
print("="*100)
```

```
(33500, 17) (33500,)
(16500, 17) (16500,)
====================================================================================================
After vectorizations of project_resource
(33500, 5000) (33500,)
(16500, 5000) (16500,)
====================================================================================================
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [43]:

```
#Project Grade Category
my_counter = Counter()
for word in X_train['clean_categories'].values:
    my_counter.update(word.split())

sorted_cat_dict = dict(my_counter)


#print(X_train.shape, y_train.shape)
#print(X_cv.shape, y_cv.shape)
#print(X_test.shape, y_test.shape)

print("="*100)

vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()),min_df=10,ngram_range=(1,2), max_featur
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on train data

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

print(vectorizer.get_feature_names())
print("After vectorizations of project categories")
print(X_train_cat_hot.shape, y_train.shape)
#print(X_cv_cat_hot.shape, y_cv.shape)
print(X_test_cat_hot.shape, y_test.shape)
print("="*100)
```

```
====================================================================================================
['Math_Science', 'Literacy_Language', 'Health_Sports', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds', 'Hi
story_Civics', 'Warmth', 'Care_Hunger']
After vectorizations of project categories
(33500, 9) (33500,)
(16500, 9) (16500,)
====================================================================================================
```

In [44]:

```
#print(X_train.shape, y_train.shape)
#print(X_cv.shape, y_cv.shape)
#print(X_test.shape, y_test.shape)

my_counter = Counter()
for word in X_train['clean_subcategories'].values:
    my_counter.update(word.split())

sorted_sub_cat_dict = dict(my_counter)

print("="*100)

vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()),min_df=10,ngram_range=(1,4), max_fe
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_subcat_hot = vectorizer.transform(X_train['clean_subcategories'].values)
#X_cv_subcat_hot = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_subcat_hot = vectorizer.transform(X_test['clean_subcategories'].values)

print(vectorizer.get_feature_names())
print("After vectorizations of sub categories")
print(X_train_subcat_hot.shape, y_train.shape)
#print(X_cv_subcat_hot.shape, y_cv.shape)
print(X_test_subcat_hot.shape, y_test.shape)
print("="*100)
```

```
====================================================================================================
['EnvironmentalScience', 'Literacy', 'Health_Wellness', 'TeamSports', 'AppliedSciences', 'Music', 'Performi
ngArts', 'NutritionEducation', 'Mathematics', 'EarlyDevelopment', 'VisualArts', 'SpecialNeeds', 'Literature
_Writing', 'Other', 'Gym_Fitness', 'College_CareerPrep', 'ForeignLanguages', 'Health_LifeScience', 'ParentI
nvolvement', 'CharacterEducation', 'SocialSciences', 'ESL', 'FinancialLiteracy', 'History_Geography', 'Comm
unityService', 'Extracurricular', 'Civics_Government', 'Economics', 'Warmth', 'Care_Hunger']
After vectorizations of sub categories
(33500, 30) (33500,)
(16500, 30) (16500,)
====================================================================================================
```

In [45]:

```
#print(X_train.shape, y_train.shape)
#print(X_cv.shape, y_cv.shape)
#print(X_test.shape, y_test.shape)

print("="*100)
vectorizer = CountVectorizer(lowercase=False)
vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on train data

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

print(vectorizer.get_feature_names())
print("After vectorizations of teacher prefix")
print(X_train_prefix_hot.shape, y_train.shape)
#print(X_cv_prefix_hot.shape, y_cv.shape)
print(X_test_prefix_hot.shape, y_test.shape)
print("="*100)
```

```
====================================================================================================
['Dr', 'Mr', 'Mrs', 'Ms', 'Teacher']
After vectorizations of teacher prefix
(33500, 5) (33500,)
(16500, 5) (16500,)
====================================================================================================
```

In [46]:

```
#print(X_train.shape, y_train.shape)
#print(X_cv.shape, y_cv.shape)
#print(X_test.shape, y_test.shape)

print("="*100)
vectorizer = CountVectorizer(lowercase=False)
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train data

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

print(vectorizer.get_feature_names())
print("After vectorizations of school state")
print(X_train_sch_state_hot.shape, y_train.shape)
#print(X_cv_sch_state_hot.shape, y_cv.shape)
print(X_test_sch_state_hot.shape, y_test.shape)
print("="*100)
```

```
====================================================================================================
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'IA', 'ID', 'IL', 'IN', 'KS', 'K
Y', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM', 'NV', 'NY',
'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA', 'WI', 'WV', 'WY']
After vectorizations of school state
(33500, 51) (33500,)
(16500, 51) (16500,)
====================================================================================================
```

In [47]:

```
#Project Grade Category
my_counter = Counter()
for word in X_train['project_grade_category'].values:
    my_counter.update(word.split())

project_grade_dict = dict(my_counter)
sorted_project_grade_dict = dict(sorted(project_grade_dict.items(), key=lambda kv: kv[1]))
#sorted_project_grade_dict.pop('Grades')
## we use count vectorizer to convert the values into one hot encoded features

vectorizer = CountVectorizer(vocabulary=list(sorted_project_grade_dict.keys()), lowercase=False, binary=True
vectorizer.fit(X_train['project_grade_category'].values)
print(vectorizer.get_feature_names())

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_hot = vectorizer.transform(X_train['project_grade_category'].values)
#X_cv_grade_hot = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_hot = vectorizer.transform(X_test['project_grade_category'].values)

print("After vectorizations of project grade")
print(X_train_grade_hot.shape, y_train.shape)
#print(X_cv_grade_hot.shape, y_cv.shape)
print(X_test_grade_hot.shape, y_test.shape)
print("="*100)
```

```
['Grades_9-12', 'Grades_6-8', 'Grades_3-5', 'Grades_PreK-2']
After vectorizations of project grade
(33500, 4) (33500,)
(16500, 4) (16500,)
====================================================================================================
```

In [48]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardS
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 da
#print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

# we use the fitted CountVectorizer to convert the text to vector
X_train_price_std = price_scalar.transform(X_train['price'].values.reshape(-1, 1))
#X_cv_price_std = price_scalar.transform(X_cv['price'].values.reshape(-1, 1))
X_test_price_std = price_scalar.transform(X_test['price'].values.reshape(-1, 1))
#print(vectorizer.get_feature_names())
print("After vectorizations of price")
print(X_train_price_std.shape, y_train.shape)
#print(X_cv_price_std.shape, y_cv.shape)
print(X_test_price_std.shape, y_test.shape)
print("="*100)
```

```
After vectorizations of price
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

In [49]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardS
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)

projects_scalar = StandardScaler()
projects_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)) # finding
#print(f"Mean : {projects_scalar.mean_[0]}, Standard deviation : {np.sqrt(projects_scalar.var_[0])}")

# we use the fitted CountVectorizer to convert the text to vector
X_train_prev_projects_std = projects_scalar.transform(X_train['teacher_number_of_previously_posted_projects'
#X_cv_prev_projects_std = projects_scalar.transform(X_cv['teacher_number_of_previously_posted_projects'].val
X_test_prev_projects_std = projects_scalar.transform(X_test['teacher_number_of_previously_posted_projects'].

#print(projects_scalar.get_feature_names())
print("After vectorizations of previously posted projects")
print(X_train_prev_projects_std.shape, y_train.shape)
#print(X_cv_prev_projects_std.shape, y_cv.shape)
print(X_test_prev_projects_std.shape, y_test.shape)
print("="*100)

#teacher_no_of_prev_Proj_standardized
```

```
After vectorizations of previously posted projects
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

In [50]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardS
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)

quantity_scalar = StandardScaler()
quantity_scalar.fit(X_train['quantity'].values.reshape(-1,1)) # finding the mean and standard deviation of t
#print(f"Mean : {quantity_scalar.mean_[0]}, Standard deviation : {np.sqrt(quantity_scalar.var_[0])}")

# we use the fitted CountVectorizer to convert the text to vector
X_train_qty_std = quantity_scalar.transform(X_train['quantity'].values.reshape(-1, 1))
#X_cv_qty_std = quantity_scalar.transform(X_cv['quantity'].values.reshape(-1, 1))
X_test_qty_std = quantity_scalar.transform(X_test['quantity'].values.reshape(-1, 1))

print("After vectorizations of quantity")
print(X_train_qty_std.shape, y_train.shape)
#print(X_cv_qty_std.shape, y_cv.shape)
print(X_test_qty_std.shape, y_test.shape)
print("="*100)

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

```
After vectorizations of quantity
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

1.5.4 Merging all the above features

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

In [51]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train_set1 = hstack((X_train_essay_bow,X_train_title_bow,X_train_resource_bow,X_train_cat_hot,X_train_subc
X_train_sch_state_hot,X_train_grade_hot,X_train_price_std,X_train_prev_projects_std,X_train_qty_std)).tocsr(

#X_cval_set1 =  hstack((X_cv_essay_bow,X_cv_title_bow,X_cv_resource_bow,X_cv_cat_hot,X_cv_subcat_hot,X_cv_pr
#X_cv_grade_hot,X_cv_price_std,X_cv_prev_projects_std,X_cv_qty_std)).tocsr()


X_test_set1 = hstack((X_test_essay_bow,X_test_title_bow,X_test_resource_bow,X_test_cat_hot,X_test_subcat_hot
X_test_sch_state_hot,X_test_grade_hot,X_test_price_std,X_test_prev_projects_std,X_test_qty_std)).tocsr()

print(X_train_set1.shape, y_train.shape)
#print(X_cval_set1.shape, y_cv.shape)
print(X_test_set1.shape, y_test.shape)
print("="*100)
```

```
(33500, 12809) (33500,)
(16500, 12809) (16500,)
====================================================================================================
```

In [52]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train_set2 = hstack((X_train_essay_tfidf,X_train_title_tfidf,X_train_cat_hot,X_train_subcat_hot,X_train_pr
X_train_sch_state_hot,X_train_grade_hot,X_train_price_std,X_train_prev_projects_std,X_train_qty_std)).tocsr(

#X_cval_set2 =  hstack((X_cv_essay_tfidf,X_cv_title_tfidf,X_cv_cat_hot,X_cv_subcat_hot,X_cv_prefix_hot,X_cv_
#X_cv_grade_hot,X_cv_price_std,X_cv_prev_projects_std,X_cv_qty_std)).tocsr()


X_test_set2 = hstack((X_test_essay_tfidf,X_test_title_tfidf,X_test_cat_hot,X_test_subcat_hot,X_test_prefix_h
X_test_sch_state_hot,X_test_grade_hot,X_test_price_std,X_test_prev_projects_std,X_test_qty_std)).tocsr()

print(X_train_set2.shape, y_train.shape)
#print(X_cval_set2.shape, y_cv.shape)
print(X_test_set2.shape, y_test.shape)
print("="*100)
```

```
(33500, 7809) (33500,)
(16500, 7809) (16500,)
====================================================================================================
```

In [53]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train_set3 = hstack((X_train_essay_avg_w2v_vec,X_train_title_avg_w2v_title,X_train_cat_hot,X_train_subcat_
X_train_sch_state_hot,X_train_grade_hot,X_train_price_std,X_train_prev_projects_std,X_train_qty_std)).tocsr(

#X_cval_set3 =  hstack((X_cv_essay_avg_w2v_vec,X_cv_title_avg_w2v_title,X_cv_cat_hot,X_cv_subcat_hot,X_cv_pr
#X_cv_grade_hot,X_cv_price_std,X_cv_prev_projects_std,X_cv_qty_std)).tocsr()

#X_cval_set3 =  hstack((X_cv_essay_avg_w2v_vec,X_cv_title_avg_w2v_title,X_cv_cat_hot,X_cv_subcat_hot,X_cv_pr
#X_cv_grade_hot,X_cv_price_std,X_cv_prev_projects_std,X_cv_qty_std)).tocsr()



X_test_set3 = hstack((X_test_essay_avg_w2v_vec,X_test_title_avg_w2v_title,X_test_cat_hot,X_test_subcat_hot,X
X_test_sch_state_hot,X_test_grade_hot,X_test_price_std,X_test_prev_projects_std,X_test_qty_std)).tocsr()

print(X_train_set3.shape, y_train.shape)
#print(X_cval_set3.shape, y_cv.shape)
print(X_test_set3.shape, y_test.shape)
print("="*100)
```

```
(33500, 702) (33500,)
(16500, 702) (16500,)
====================================================================================================
```

In [54]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train_set4 = hstack((X_train_essay_tfidf_w2v_vec,X_train_title_tfidf_w2v_vec,X_train_cat_hot,X_train_subca
X_train_sch_state_hot,X_train_grade_hot,X_train_price_std,X_train_prev_projects_std,X_train_qty_std)).tocsr(

#X_cval_set4 =  hstack((X_cv_essay_tfidf_w2v_vec,X_cv_title_tfidf_w2v_vec,X_cv_cat_hot,X_cv_subcat_hot,X_cv_
#X_cv_grade_hot,X_cv_price_std,X_cv_prev_projects_std,X_cv_qty_std)).tocsr()


X_test_set4 = hstack((X_test_essay_tfidf_w2v_vec,X_test_title_tfidf_w2v_vec,X_test_cat_hot,X_test_subcat_hot
X_test_sch_state_hot,X_test_grade_hot,X_test_price_std,X_test_prev_projects_std,X_test_qty_std)).tocsr()

print(X_train_set4.shape, y_train.shape)
#print(X_cval_set4.shape, y_cv.shape)
print(X_test_set4.shape, y_test.shape)
print("="*100)
```

```
(33500, 702) (33500,)
(16500, 702) (16500,)
====================================================================================================
```

Applying SVM

SVM on set1

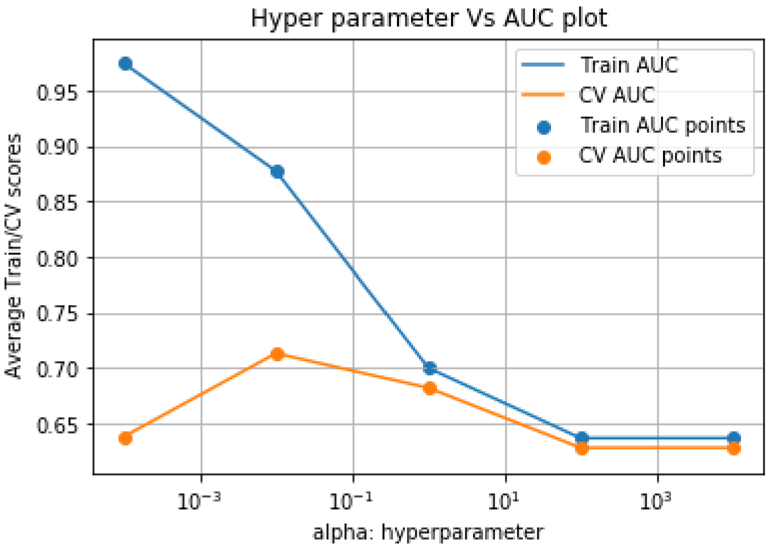
In [55]:

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

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs

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

    return y_data_pred
```

In [56]:

```
#code source: http://occam.olin.edu/sites/default/files/DataScienceMaterials/machine_learning_lecture_2/Mach


from sklearn.model_selection import train_test_split,GridSearchCV
#from sklearn.grid_search import GridSearchCV
from sklearn.datasets import *
from sklearn import linear_model
from sklearn.linear_model import SGDClassifier
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint


clf = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced')

tuned_parameters = {'alpha':[10**-4, 10**-2, 10**0, 10**2, 10**4]}
print(tuned_parameters)
model = GridSearchCV(clf, tuned_parameters, cv=5, return_train_score = True,scoring='roc_auc')
print(clf)
model.fit(X_train_set1, y_train)


results = pd.DataFrame.from_dict(model.cv_results_)
#results = results.sort_values(['param_n_neighbors'])

#print(results)

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
#K =  results['param_n_neighbors']

plt.plot(tuned_parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
#plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

plt.plot(tuned_parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
#plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

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

#https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.xscale.html:
plt.xscale('log')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel(" Average Train/CV scores")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```

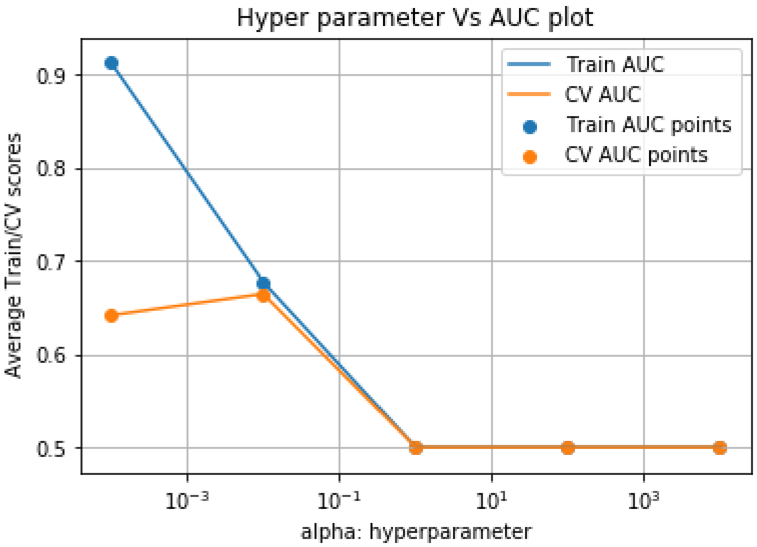
```
{'alpha': [0.0001, 0.01, 1, 100, 10000]}
SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```



In [60]:

```
#code source: http://occam.olin.edu/sites/default/files/DataScienceMaterials/machine_learning_lecture_2/Mach


from sklearn.model_selection import train_test_split,GridSearchCV
#from sklearn.grid_search import GridSearchCV
from sklearn.datasets import *
from sklearn import linear_model
from sklearn.linear_model import SGDClassifier
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint


clf = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced')

tuned_parameters = {'alpha':[10**-4, 10**-2, 10**0, 10**2, 10**4]}
print(tuned_parameters)
model = GridSearchCV(clf, tuned_parameters, cv=5, return_train_score = True,scoring='roc_auc')
print(clf)
model.fit(X_train_set1, y_train)


results = pd.DataFrame.from_dict(model.cv_results_)
#results = results.sort_values(['param_n_neighbors'])

#print(results)

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
#K =  results['param_n_neighbors']

plt.plot(tuned_parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
#plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

plt.plot(tuned_parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
#plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

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

#https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.xscale.html:
plt.xscale('log')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel(" Average Train/CV scores")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```

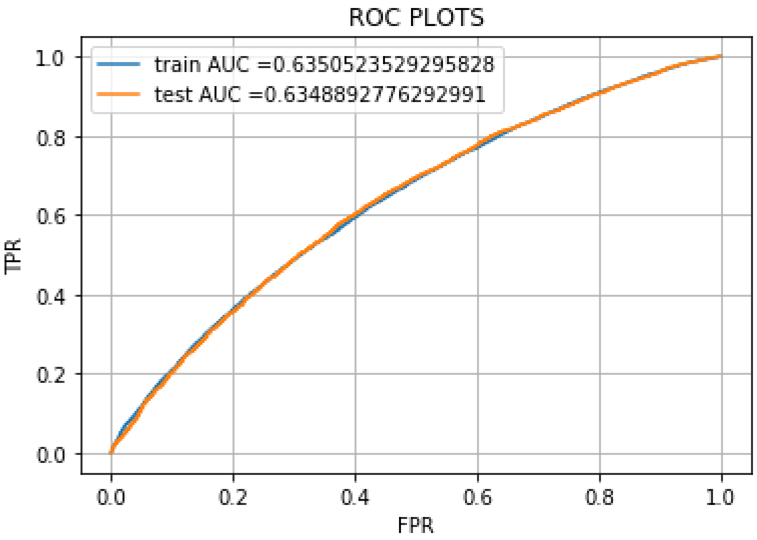
```
{'alpha': [0.0001, 0.01, 1, 100, 10000]}
SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l1',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```



Fitting Model to Hyper-Parameter Curve

In [57]:

```
from sklearn.metrics import roc_curve, auc


clf = SGDClassifier(loss = 'hinge', penalty = 'l2', alpha = 10,class_weight='balanced')
clf.fit(X_train_set1, 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 = clf.decision_function(X_train_set1)
y_test_pred = clf.decision_function(X_test_set1)

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("FPR")
plt.ylabel("TPR")
plt.title("ROC PLOTS")
plt.grid()
plt.show()
```



In [58]:

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

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

In [59]:

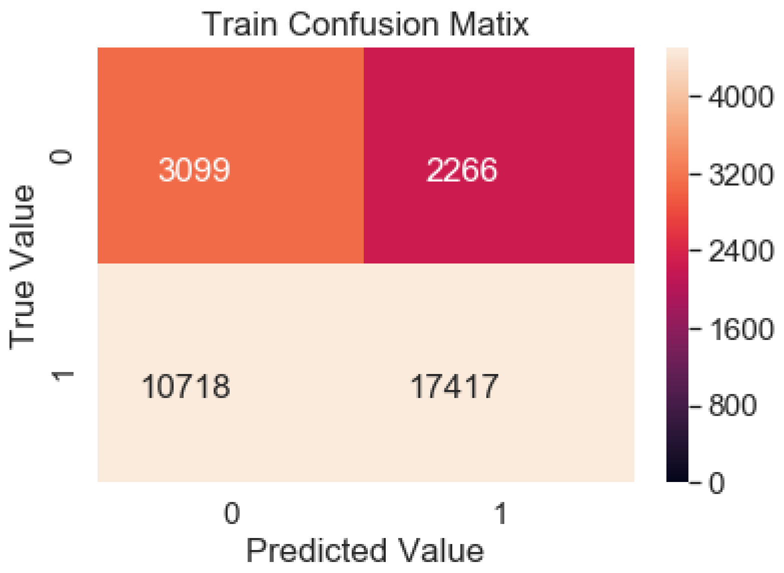
```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
#https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels/48018785
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.3575841680646715 for threshold 0.222
Train confusion matrix
[[ 3099  2266]
 [10718 17417]]
Test confusion matrix
[[1552 1090]
 [5361 8497]]
```

In [60]:

```
#https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels/48018785
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as sns
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_train,predict_with_best_t(y_train_pred,best_t)))
sns.set(font_scale=1.4)
fig = plt.figure()
ax = fig.add_subplot(111)
#ax.set_aspect(1)
#fig, ax = plt.subplots(figsize='3')
print(train_confusion_matrix)
akws = {"ha": 'right',"va": 'top'}

sns.heatmap(train_confusion_matrix, annot = True, annot_kws=akws, fmt = 'g',vmin=0.0 , vmax=4500)

plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Train Confusion Matix")
plt.show()
```

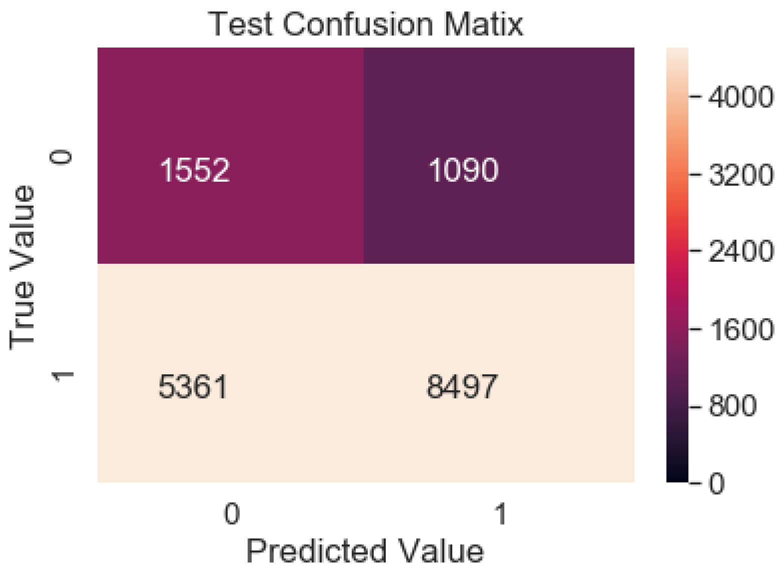
```
       0      1
0   3099   2266
1  10718  17417
```


Train Confusion Matix
True Value
0
1
3099
2266
10718
17417
0
1
Predicted Value
4000
3200
2400
1600
800
0

In [61]:

```
#https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels/48018785
#https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels/48018785
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as sns
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict_with_best_t(y_test_pred,best_t)))
sns.set(font_scale=1.4)
fig = plt.figure()
ax = fig.add_subplot(111)
#ax.set_aspect(1)
#fig, ax = plt.subplots(figsize='3')
print(train_confusion_matrix)
akws = {"ha": 'right',"va": 'top'}

sns.heatmap(train_confusion_matrix, annot = True, annot_kws=akws, fmt = 'g',vmin=0.0 , vmax=4500)

plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Test Confusion Matix")
plt.show()
```

```
      0     1
0  1552  1090
1  5361  8497
```

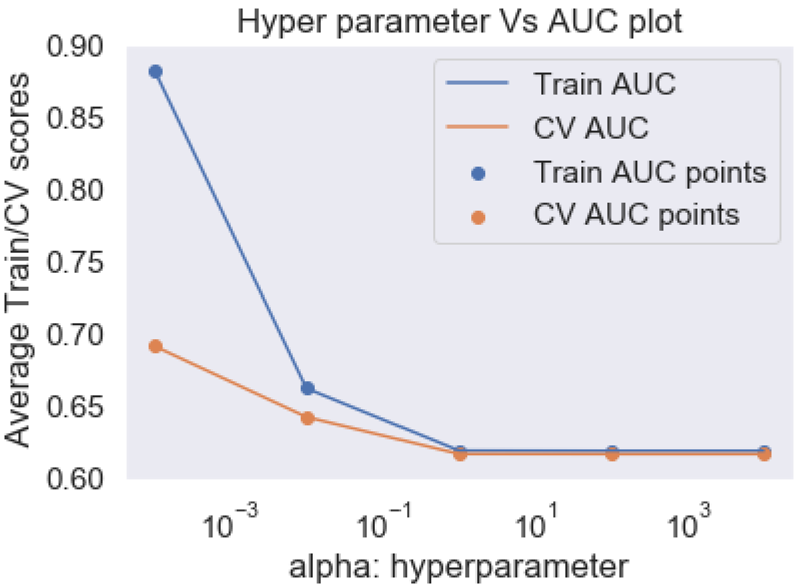


SVM on set2

In [62]:

```
#code source: http://occam.olin.edu/sites/default/files/DataScienceMaterials/machine_learning_lecture_2/Mach


from sklearn.model_selection import train_test_split,GridSearchCV
#from sklearn.grid_search import GridSearchCV
from sklearn.datasets import *
from sklearn import linear_model
from sklearn.linear_model import SGDClassifier
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint


clf = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced')

tuned_parameters = {'alpha':[10**-4, 10**-2, 10**0, 10**2, 10**4]}
print(tuned_parameters)
model = GridSearchCV(clf, tuned_parameters, cv=5, return_train_score = True,scoring='roc_auc')
print(clf)
model.fit(X_train_set2, y_train)


results = pd.DataFrame.from_dict(model.cv_results_)
#results = results.sort_values(['param_n_neighbors'])

#print(results)

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
#K =  results['param_n_neighbors']

plt.plot(tuned_parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
#plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

plt.plot(tuned_parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
#plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

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

#https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.xscale.html:
plt.xscale('log')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel(" Average Train/CV scores")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```

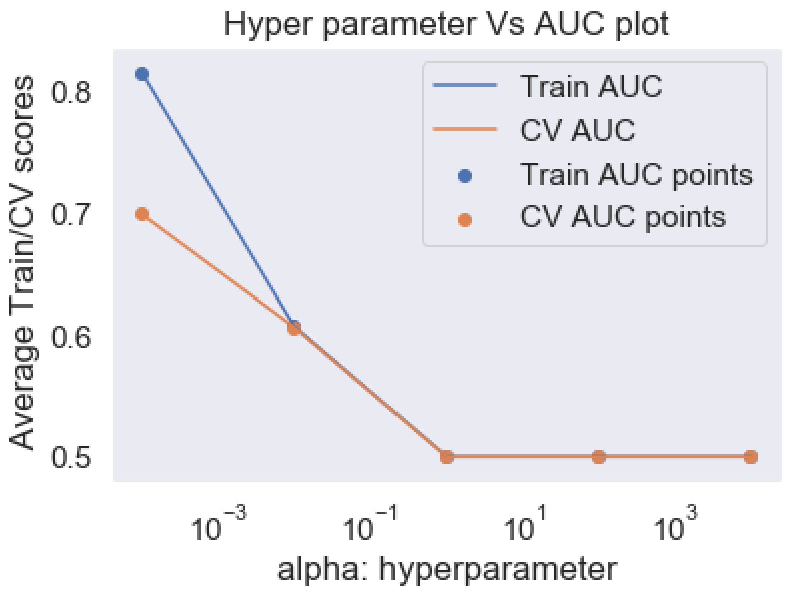
```
{'alpha': [0.0001, 0.01, 1, 100, 10000]}
SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```



In [63]:

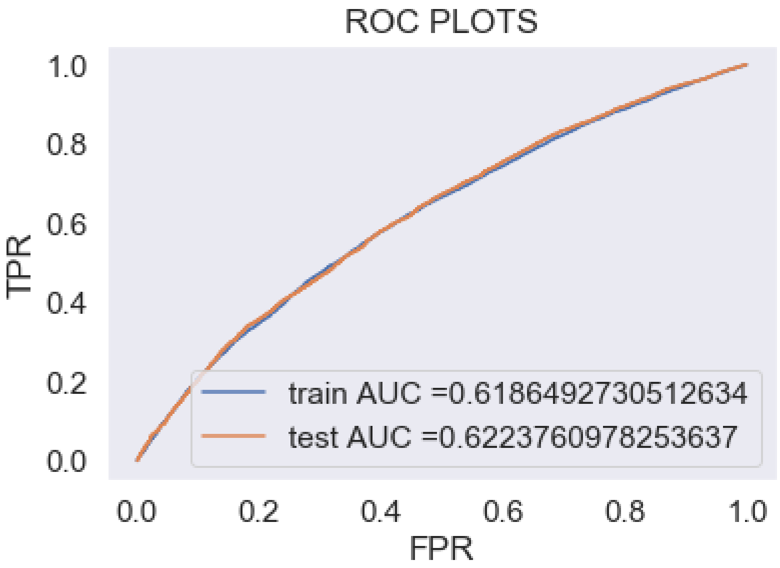
```
#code source: http://occam.olin.edu/sites/default/files/DataScienceMaterials/machine_learning_lecture_2/Mach


from sklearn.model_selection import train_test_split,GridSearchCV
#from sklearn.grid_search import GridSearchCV
from sklearn.datasets import *
from sklearn import linear_model
from sklearn.linear_model import SGDClassifier
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint


clf = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced')

tuned_parameters = {'alpha':[10**-4, 10**-2, 10**0, 10**2, 10**4]}
print(tuned_parameters)
model = GridSearchCV(clf, tuned_parameters, cv=5, return_train_score = True,scoring='roc_auc')
print(clf)
model.fit(X_train_set2, y_train)


results = pd.DataFrame.from_dict(model.cv_results_)
#results = results.sort_values(['param_n_neighbors'])

#print(results)

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
#K =  results['param_n_neighbors']

plt.plot(tuned_parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
#plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

plt.plot(tuned_parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
#plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

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

#https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.xscale.html:
plt.xscale('log')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel(" Average Train/CV scores")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```

```
{'alpha': [0.0001, 0.01, 1, 100, 10000]}
SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l1',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```



In [64]:

```
from sklearn.metrics import roc_curve, auc


clf = SGDClassifier(loss = 'hinge', penalty = 'l2',alpha=1,class_weight='balanced')
clf.fit(X_train_set2, 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 = clf.decision_function(X_train_set2)
y_test_pred = clf.decision_function(X_test_set2)

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("FPR")
plt.ylabel("TPR")
plt.title("ROC PLOTS")
plt.grid()
plt.show()
```



In [65]:

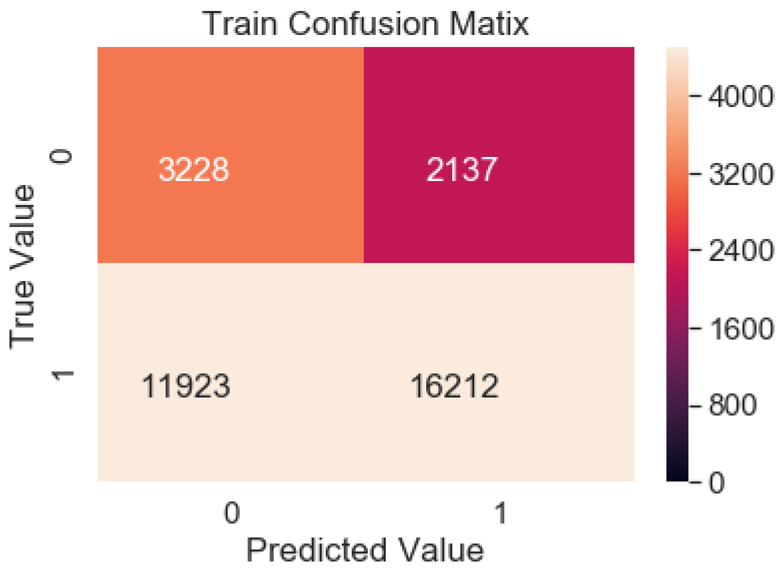
```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
#https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels/48018785
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.34669970755764 for threshold 0.029
Train confusion matrix
[[ 3228  2137]
 [11923 16212]]
Test confusion matrix
[[1602 1040]
 [5936 7922]]
```

In [66]:

```
#https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels/48018785
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as sns
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_train,predict_with_best_t(y_train_pred,best_t)))
sns.set(font_scale=1.4)
fig = plt.figure()
ax = fig.add_subplot(111)
#ax.set_aspect(1)
#fig, ax = plt.subplots(figsize='3')
print(train_confusion_matrix)
akws = {"ha": 'right',"va": 'top'}

sns.heatmap(train_confusion_matrix, annot = True, annot_kws=akws, fmt = 'g',vmin=0.0 , vmax=4500)

plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Train Confusion Matix")
plt.show()
```

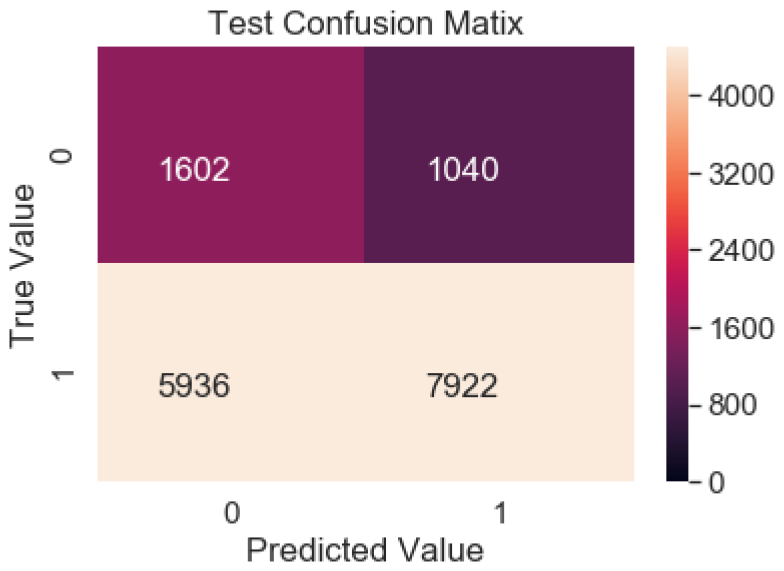
```
       0      1
0   3228   2137
1  11923  16212
```



In [67]:

```
#https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels/48018785
#https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels/48018785
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as sns
test_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict_with_best_t(y_test_pred,best_t)))
sns.set(font_scale=1.4)
fig = plt.figure()
ax = fig.add_subplot(111)
#ax.set_aspect(1)
#fig, ax = plt.subplots(figsize='3')
print(train_confusion_matrix)
akws = {"ha": 'right',"va": 'top'}

sns.heatmap(test_confusion_matrix, annot = True, annot_kws=akws, fmt = 'g',vmin=0.0 , vmax=4500)

plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Test Confusion Matix")
plt.show()
```

```
       0      1
0   3228   2137
1  11923  16212
```

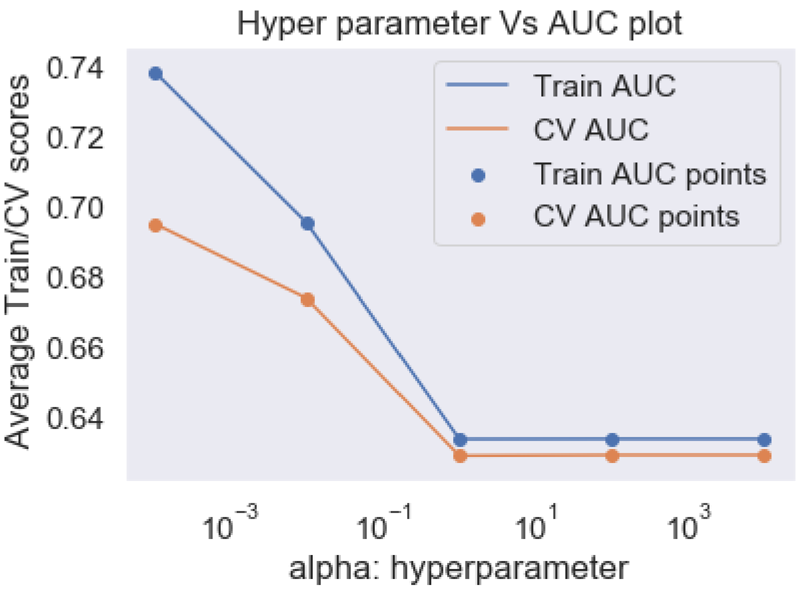


SVM on set3

In [68]:

```
#code source: http://occam.olin.edu/sites/default/files/DataScienceMaterials/machine_learning_lecture_2/Mach


from sklearn.model_selection import train_test_split,GridSearchCV
#from sklearn.grid_search import GridSearchCV
from sklearn.datasets import *
from sklearn import linear_model
from sklearn.linear_model import SGDClassifier
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint


clf = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced')

tuned_parameters = {'alpha':[10**-4, 10**-2, 10**0, 10**2, 10**4]}
print(tuned_parameters)
model = GridSearchCV(clf, tuned_parameters, cv=5, return_train_score = True,scoring='roc_auc')
print(clf)
model.fit(X_train_set3, y_train)


results = pd.DataFrame.from_dict(model.cv_results_)
#results = results.sort_values(['param_n_neighbors'])

#print(results)

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
#K =  results['param_n_neighbors']

plt.plot(tuned_parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
#plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

plt.plot(tuned_parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
#plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

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

#https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.xscale.html:
plt.xscale('log')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel(" Average Train/CV scores")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```

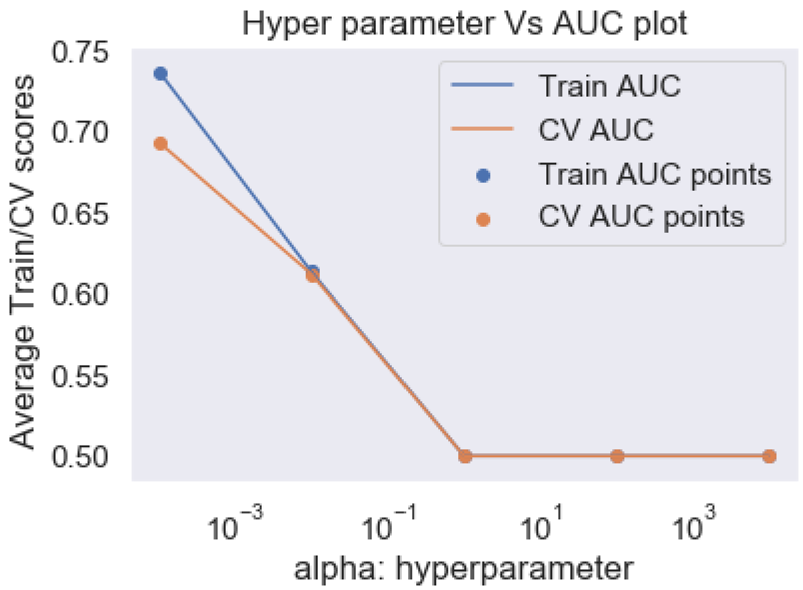
```
{'alpha': [0.0001, 0.01, 1, 100, 10000]}
SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```



In [69]:

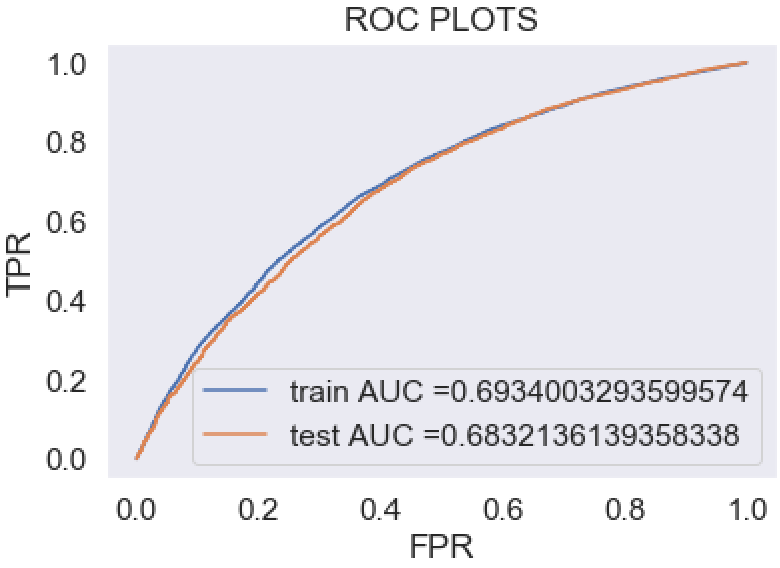
```
#code source: http://occam.olin.edu/sites/default/files/DataScienceMaterials/machine_learning_lecture_2/Mach


from sklearn.model_selection import train_test_split,GridSearchCV
#from sklearn.grid_search import GridSearchCV
from sklearn.datasets import *
from sklearn import linear_model
from sklearn.linear_model import SGDClassifier
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint


clf = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced')

tuned_parameters = {'alpha':[10**-4, 10**-2, 10**0, 10**2, 10**4]}
print(tuned_parameters)
model = GridSearchCV(clf, tuned_parameters, cv=5, return_train_score = True,scoring='roc_auc')
print(clf)
model.fit(X_train_set3, y_train)


results = pd.DataFrame.from_dict(model.cv_results_)
#results = results.sort_values(['param_n_neighbors'])

#print(results)

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
#K =  results['param_n_neighbors']

plt.plot(tuned_parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
#plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

plt.plot(tuned_parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
#plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

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

#https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.xscale.html:
plt.xscale('log')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel(" Average Train/CV scores")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```

```
{'alpha': [0.0001, 0.01, 1, 100, 10000]}
SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l1',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```



In [70]:

```
from sklearn.metrics import roc_curve, auc


clf = SGDClassifier(loss = 'hinge', penalty = 'l2',alpha=0.01,class_weight='balanced')
clf.fit(X_train_set3, 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 = clf.decision_function(X_train_set3)
y_test_pred = clf.decision_function(X_test_set3)

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("FPR")
plt.ylabel("TPR")
plt.title("ROC PLOTS")
plt.grid()
plt.show()
```



In [71]:

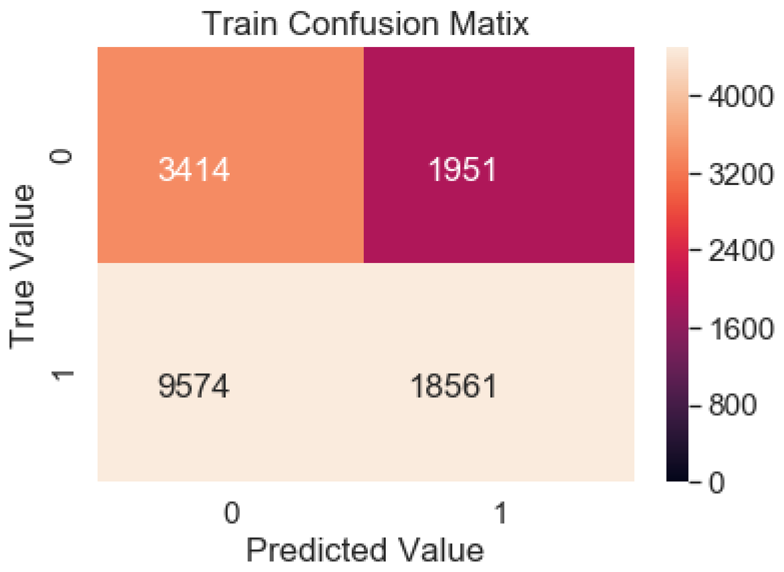
```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
#https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels/48018785
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.4198056136941928 for threshold -0.244
Train confusion matrix
[[ 3414  1951]
 [ 9574 18561]]
Test confusion matrix
[[1629 1013]
 [4664 9194]]
```

In [72]:

```python
#https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels/48018785
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as sns
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_train,predict_with_best_t(y_train_pred,best_t)))
sns.set(font_scale=1.4)
fig = plt.figure()
ax = fig.add_subplot(111)
#ax.set_aspect(1)
#fig, ax = plt.subplots(figsize='3')
print(train_confusion_matrix)
akws = {"ha": 'right',"va": 'top'}

sns.heatmap(train_confusion_matrix, annot = True, annot_kws=akws, fmt = 'g',vmin=0.0 , vmax=4500)

plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Train Confusion Matix")
plt.show()
```

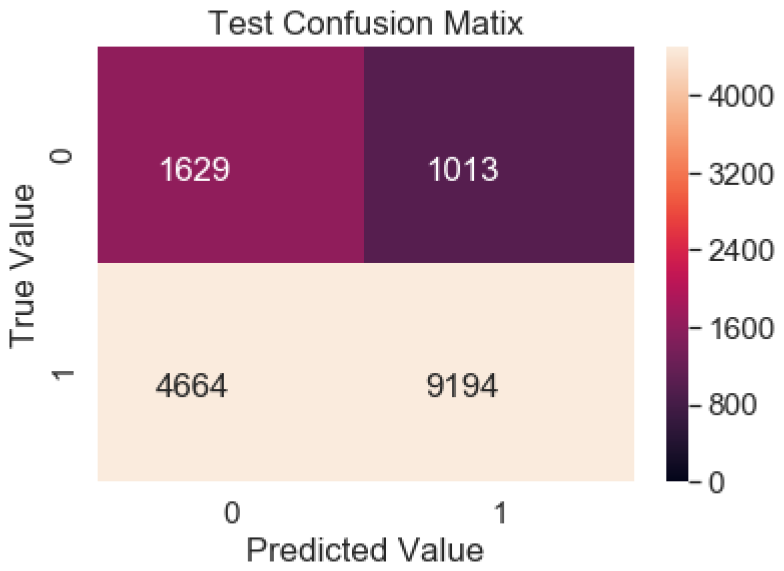
```
      0      1
0  3414   1951
1  9574  18561
```


Train Confusion Matix

In [73]:

```
#https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels/48018785
#https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels/48018785
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as sns
test_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict_with_best_t(y_test_pred,best_t)))
sns.set(font_scale=1.4)
fig = plt.figure()
ax = fig.add_subplot(111)
#ax.set_aspect(1)
#fig, ax = plt.subplots(figsize='3')
print(train_confusion_matrix)
akws = {"ha": 'right',"va": 'top'}

sns.heatmap(test_confusion_matrix, annot = True, annot_kws=akws, fmt = 'g',vmin=0.0 , vmax=4500)

plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Test Confusion Matix")
plt.show()
```

```
      0      1
0  3414   1951
1  9574  18561
```



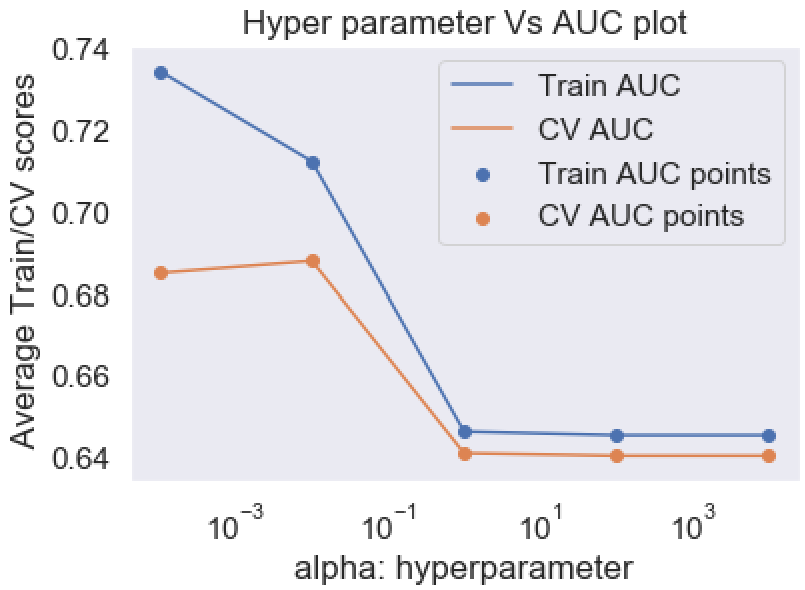
SVM on set4

In [74]:

```
#code source: http://occam.olin.edu/sites/default/files/DataScienceMaterials/machine_learning_lecture_2/Mach

from sklearn.model_selection import train_test_split,GridSearchCV
#from sklearn.grid_search import GridSearchCV
from sklearn.datasets import *
from sklearn import linear_model
from sklearn.linear_model import SGDClassifier
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint


clf = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced')

tuned_parameters = {'alpha':[10**-4, 10**-2, 10**0, 10**2, 10**4]}
print(tuned_parameters)
model = GridSearchCV(clf, tuned_parameters, cv=5, return_train_score = True,scoring='roc_auc')
print(clf)
model.fit(X_train_set4, y_train)


results = pd.DataFrame.from_dict(model.cv_results_)
#results = results.sort_values(['param_n_neighbors'])

#print(results)

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
#K =  results['param_n_neighbors']

plt.plot(tuned_parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
#plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

plt.plot(tuned_parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
#plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

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

```
#https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.xscale.html:
plt.xscale('log')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel(" Average Train/CV scores")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```

```
{'alpha': [0.0001, 0.01, 1, 100, 10000]}
SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

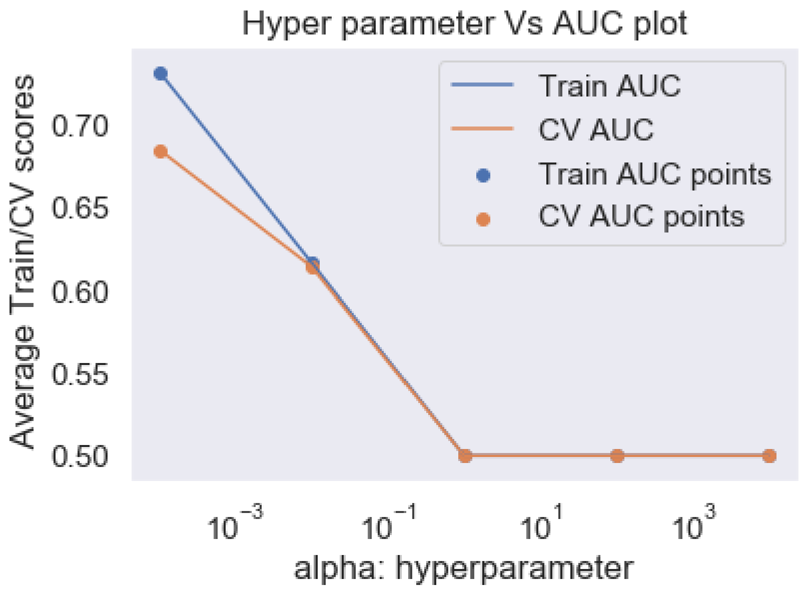


In [75]:

```
#code source: http://occam.olin.edu/sites/default/files/DataScienceMaterials/machine_learning_lecture_2/Mach

from sklearn.model_selection import train_test_split,GridSearchCV
#from sklearn.grid_search import GridSearchCV
from sklearn.datasets import *
from sklearn import linear_model
from sklearn.linear_model import SGDClassifier
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint


clf = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced')

tuned_parameters = {'alpha':[10**-4, 10**-2, 10**0, 10**2, 10**4]}
print(tuned_parameters)
model = GridSearchCV(clf, tuned_parameters, cv=5, return_train_score = True,scoring='roc_auc')
print(clf)
model.fit(X_train_set4, y_train)


results = pd.DataFrame.from_dict(model.cv_results_)
#results = results.sort_values(['param_n_neighbors'])

#print(results)

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
#K =  results['param_n_neighbors']

plt.plot(tuned_parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
#plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

plt.plot(tuned_parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
#plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

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

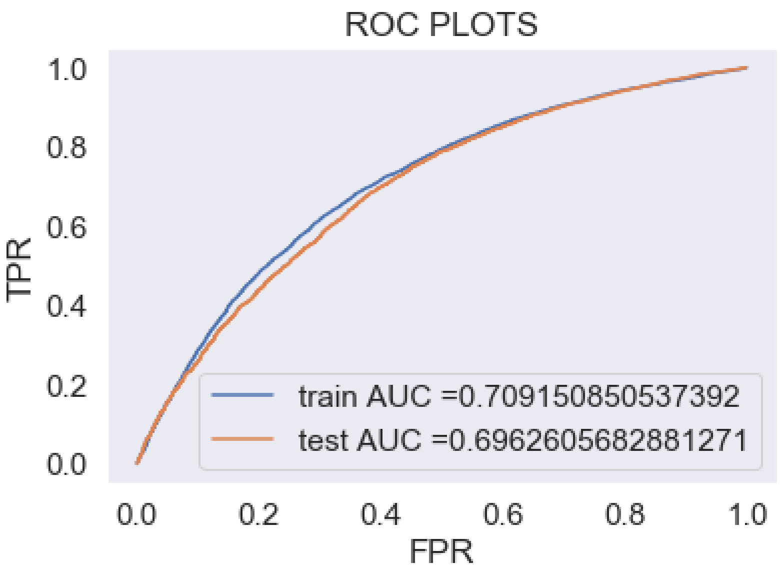
```
#https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.xscale.html:
plt.xscale('log')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel(" Average Train/CV scores")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```

```
{'alpha': [0.0001, 0.01, 1, 100, 10000]}
SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l1',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```



In [76]:

```
from sklearn.metrics import roc_curve, auc


clf = SGDClassifier(loss = 'hinge', penalty = 'l2',alpha=.01, class_weight = 'balanced')
clf.fit(X_train_set4, 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 = clf.decision_function(X_train_set4)
y_test_pred = clf.decision_function(X_test_set4)

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("FPR")
plt.ylabel("TPR")
plt.title("ROC PLOTS")
plt.grid()
plt.show()
```



In [77]:

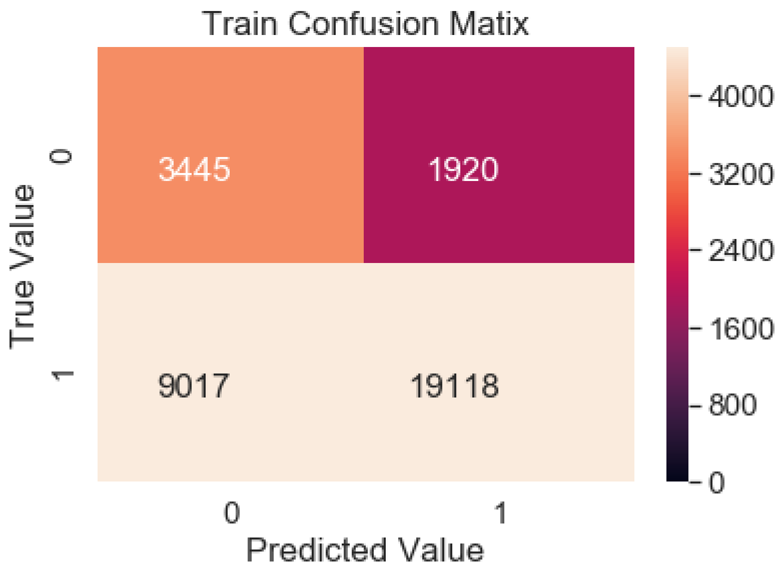
```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
#https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels/48018785
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.43632996349149383 for threshold -0.034
Train confusion matrix
[[ 3445  1920]
 [ 9017 19118]]
Test confusion matrix
[[1634 1008]
 [4439 9419]]
```

In [78]:

```
#https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels/48018785
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as sns
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_train,predict_with_best_t(y_train_pred,best_t)))
sns.set(font_scale=1.4)
fig = plt.figure()
ax = fig.add_subplot(111)
#ax.set_aspect(1)
#fig, ax = plt.subplots(figsize='3')
print(train_confusion_matrix)
akws = {"ha": 'right',"va": 'top'}

sns.heatmap(train_confusion_matrix, annot = True, annot_kws=akws, fmt = 'g',vmin=0.0 , vmax=4500)

plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Train Confusion Matix")
plt.show()
```

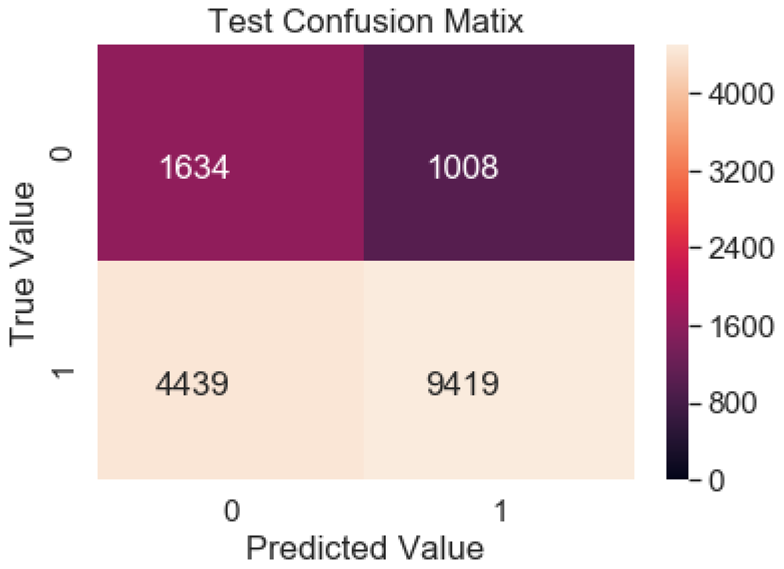
```
      0      1
0  3445   1920
1  9017  19118
```



In [80]:

```
#https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels/48018785
#https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels/48018785
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as sns
test_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict_with_best_t(y_test_pred,best_t)))
sns.set(font_scale=1.4)
fig = plt.figure()
ax = fig.add_subplot(111)
#ax.set_aspect(1)
#fig, ax = plt.subplots(figsize='3')
print(train_confusion_matrix)
akws = {"ha": 'right',"va": 'top'}

sns.heatmap(test_confusion_matrix, annot = True, annot_kws=akws, fmt = 'g',vmin=0.0 , vmax=4500)

plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Test Confusion Matix")
plt.show()
```

```
      0      1
0  3445   1920
1  9017  19118
```



Sentiment Score of essay

There is no need of standardization or normalization of the sentiment score. We are converting score to positive or negative. We will not be using sentiment score in the final model.

we need to consider neg, neu, pos, compound as 4 different features by adding these as 4 columns in project_data

In [81]:

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

# import nltk
nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()
sentiment_essay_train=[]

for essay in tqdm(X_train['clean_essays'].values):
    sentiment_essay_train = sid.polarity_scores(essay)
print(sentiment_essay_train)

X_train['neg_sent']=sentiment_essay_train['neg']
X_train['neu_sent']=sentiment_essay_train['neu']
X_train['pos_sent']=sentiment_essay_train['pos']
X_train['comp_sent']=sentiment_essay_train['compound']
# the above print statement gives us features of sentiment_scores for train data.


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

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\Admin\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
100%|██████████████████████████████████████████████████████████████████████████| 33500/33500 [01:25<00:00,
389.70it/s]

{'neg': 0.0, 'neu': 0.835, 'pos': 0.165, 'compound': 0.946}
```

In [82]:

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

# import nltk
nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()
sentiment_essay_test=[]

for essay in tqdm(X_test['clean_essays'].values):
    sentiment_essay_test = sid.polarity_scores(essay)

X_test['neg_sent']=sentiment_essay_test['neg']
X_test['neu_sent']=sentiment_essay_test['neu']
X_test['pos_sent']=sentiment_essay_test['pos']
X_test['comp_sent']=sentiment_essay_test['compound']
print(sentiment_essay_test)
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\Admin\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
100%|██████████████████████████████████████████████████████████████████████████| 16500/16500 [00:46<00:00,
357.45it/s]

{'neg': 0.052, 'neu': 0.672, 'pos': 0.277, 'compound': 0.9948}
```

TruncatedSVD on TfidfVectorizer of essay text,choose the number of components (n_components)using elbow method :numerical data

The approach here should be to: (i) find your mapping matrix from SVD using the training data; (ii) save this matrix; and (iii) directly apply this matrix to the training data during your evaluation.

Fit the SVD model on train dataset and using 'transform', transform the cv and test dataset.

Don't take higher dimesnional data. Restrict the number of dimensions to 3000 using max_features in TFIDF. Now set n_componenets= 2999 and then take the cumilative sum of explained variance ratio.

In [83]:

```
X_train_essay_tfidf.shape
```

Out[83]:

```
(33500, 5000)
```

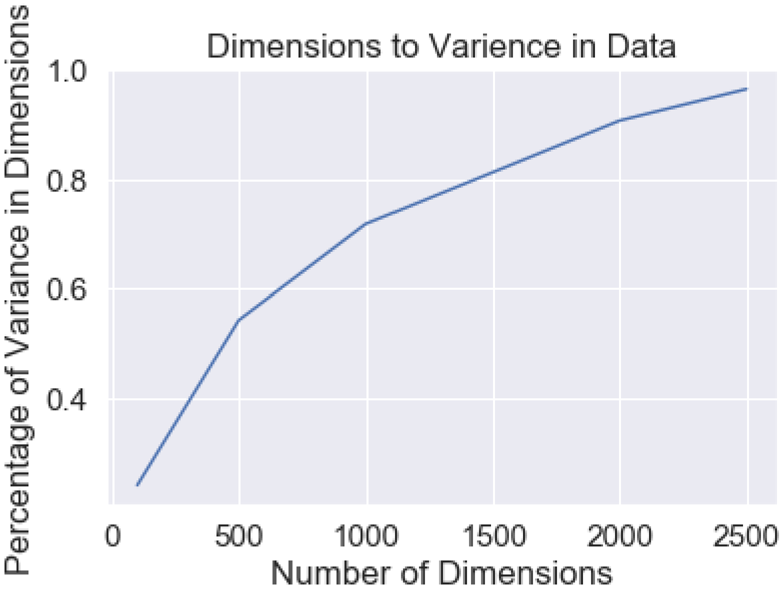
In [84]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html
#print(X_train_essay_tfidf.shape)
from sklearn.decomposition import TruncatedSVD
from sklearn.random_projection import sparse_random_matrix
X_train_essay_tfidf = X_train_essay_tfidf[:,0:3000]
X_test_essay_tfidf = X_test_essay_tfidf[:,0:3000]

index=[100,500,1000,2000,2500]

Varience_ratio= []
for i in index:
    svd = TruncatedSVD(n_components=i, random_state=42)
    svd.fit(X_train_essay_tfidf)
    Varience_ratio.append(svd.explained_variance_ratio_.sum())
print(Varience_ratio)
```

```
[0.24049730036784753, 0.542728900727482, 0.718632286813604, 0.9080752279966608, 0.9658889743817953]
```

In [85]:

```
plt.xlabel("Number of Dimensions")
plt.ylabel("Percentage of Variance in Dimensions")
plt.title("Dimensions to Varience in Data")
plt.plot(index,Varience_ratio)
plt.show()
```



#best n_dimension = 2000

#Hence we will tranform the old features to new one by fitting svd using n_components=2000

In [86]:

```
svd=TruncatedSVD(n_components = 2000)
svd.fit(X_train_essay_tfidf)
X_train_essay_tfidf= svd.transform(X_train_essay_tfidf )
X_test_essay_tfidf= svd.transform(X_test_essay_tfidf )
```

Number of words in the title : numerical data

In [87]:

```
#https://stackoverflow.com/questions/49984905/count-number-of-words-per-row
title_length_train=[]
for i in X_train['clean_titles']:
    #title_length_train.append(X_train.project_title.apply(lambda x:len(x.split())))
    title_length_train.append(len(i.split()))
#print(title_length_train)

#title_length_cv=[]
#for i in X_cv['clean_titles']:
    #title_length_train.append(X_train.project_title.apply(lambda x:len(x.split())))
 #   title_length_cv.append(len(i.split()))
#print(i)


title_length_test=[]
for i in X_test['clean_titles']:
    #title_length_train.append(X_train.project_title.apply(lambda x:len(x.split())))
    title_length_test.append(len(i.split()))
#print(title_length_test)
```

In [88]:

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

Out[88]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_title
0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades_PreK-2	Engineering STEAM into the Primary Classroom
1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades_3-5	Sensory Tools for Focus

In [89]:

```
title_length_train=np.array(title_length_train).reshape(-1,1)
#title_length_cv=np.array(title_length_cv).reshape(-1,1)
title_length_test=np.array(title_length_test).reshape(-1,1)

print(title_length_train)
```

```
[[6]
 [9]
 [6]
 ...
 [8]
 [7]
 [4]]
```

Number of words in the combine essays

In [90]:

```
essay_length_train=[]
for i in X_train['clean_essays']:
    essay_length_train.append(len(i.split()))
#print(essay_length_train)

essay_length_cv=[]
#for i in X_cv['clean_essays']:
 #   essay_length_cv.append(len(i.split()))
#print(essay_length_cv)


essay_length_test=[]
for i in X_test['clean_essays']:
    essay_length_test.append(len(i.split()))
#print(essay_length_test)
```

In [91]:

```
essay_length_train=np.array(essay_length_train).reshape(-1,1)
#essay_length_cv=np.array(essay_length_cv).reshape(-1,1)
essay_length_test=np.array(essay_length_test).reshape(-1,1)
print(essay_length_test)
```

```
[[200]
 [137]
 [107]
 ...
 [110]
 [ 89]
 [199]]
```

Merging set5 features:

Apply Logistic Regression on the below feature set Set 5 by finding the best hyper parameter as suggested in step 2 and step 3.

Consider these set of features Set 5 : school_state : categorical data clean_categories : categorical data clean_subcategories : categorical data project_grade_category :categorical data teacher_prefix : categorical data quantity : numerical data teacher_number_of_previously_posted_projects : numerical data price : numerical data sentiment score's of each of the essay : numerical data number of words in the title : numerical data number of words in the combine essays : numerical data

In [92]:

```
X_train.head(2)
```

Out[92]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	pro
45092	147838	p244095	8b9e123a40e6ba989bb61b3450f2c8e0	Mrs.	ND	2016-09-06 23:19:41	Grades_6-8	Lite
36322	136556	p207935	e1d4badcb3431c167e65fe948c084613	Mr.	CA	2016-08-25 20:08:22	Grades_3-5	Sup Sc B

2 rows × 21 columns

In [93]:

```
neg_train=X_train['neg_sent'].values.reshape(-1,1)
pos_train=X_train['pos_sent'].values.reshape(-1,1)
neu_train=X_train['neu_sent'].values.reshape(-1,1)
comp_train=X_train['comp_sent'].values.reshape(-1,1)

#neg_cv=X_cv['neg_sent'].values.reshape(-1,1)
#pos_cv=X_cv['pos_sent'].values.reshape(-1,1)
#neu_cv=X_cv['neu_sent'].values.reshape(-1,1)
#comp_cv=X_cv['comp_sent'].values.reshape(-1,1)

neg_test=X_test['neg_sent'].values.reshape(-1,1)
pos_test=X_test['pos_sent'].values.reshape(-1,1)
neu_test=X_test['neu_sent'].values.reshape(-1,1)
comp_test=X_test['comp_sent'].values.reshape(-1,1)
```

In [94]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train_set5 = hstack((X_train_cat_hot,X_train_subcat_hot,X_train_prefix_hot,
X_train_sch_state_hot,X_train_grade_hot,X_train_price_std,X_train_prev_projects_std,X_train_qty_std,title_le
                      pos_train,neu_train,comp_train,X_train_essay_tfidf)).tocsr()

#X_cval_set5 =  hstack((X_cv_cat_hot,,,X_cv_prefix_hot,X_cv_sch_state_hot,
#X_cv_grade_hot,X_cv_price_std,X_cv_prev_projects_std,X_cv_qty_std,
                       #title_length_cv,essay_length_cv,neg_cv,pos_cv,neu_cv,comp_cv)).tocsr()


X_test_set5 = hstack((X_test_cat_hot,X_test_subcat_hot,X_test_prefix_hot,
X_test_sch_state_hot,X_test_grade_hot,X_test_price_std,X_test_prev_projects_std,X_test_qty_std,title_length_
                      ,essay_length_test,neg_test, pos_test,neu_test,comp_test,X_test_essay_tfidf  )).tocsr(

print(X_train_set5.shape, y_train.shape)
#print(X_cval_set5.shape, y_cv.shape)
print(X_test_set5.shape, y_test.shape)
print("="*100)
```

```
(33500, 2108) (33500,)
(16500, 2108) (16500,)
====================================================================================================
```

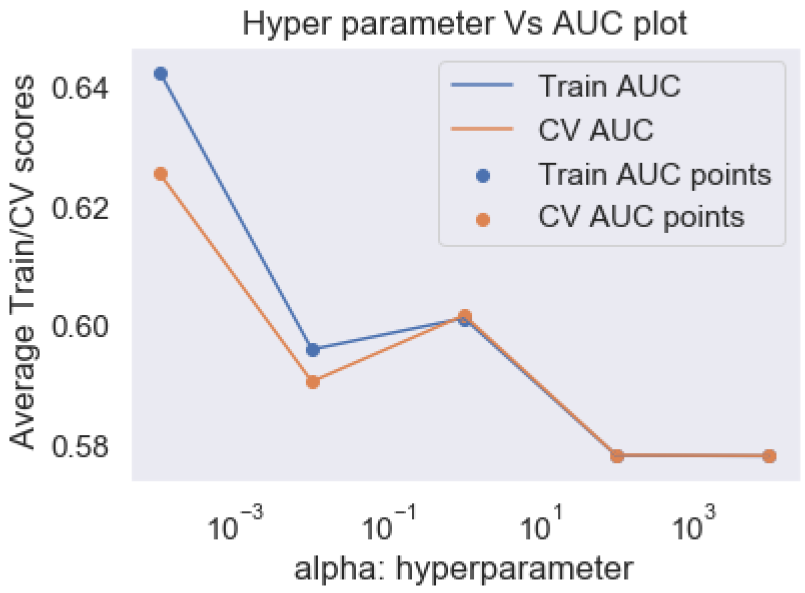
Apply SVM on set5

In [95]:

```
#code source: http://occam.olin.edu/sites/default/files/DataScienceMaterials/machine_learning_lecture_2/Mach

from sklearn.model_selection import train_test_split,GridSearchCV
#from sklearn.grid_search import GridSearchCV
from sklearn.datasets import *
from sklearn import linear_model
from sklearn.linear_model import SGDClassifier
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint


clf = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced')

tuned_parameters = {'alpha':[10**-4, 10**-2, 10**0, 10**2, 10**4]}
print(tuned_parameters)
model = GridSearchCV(clf, tuned_parameters, cv=5, return_train_score = True,scoring='roc_auc')
print(clf)
model.fit(X_train_set5, y_train)


results = pd.DataFrame.from_dict(model.cv_results_)
#results = results.sort_values(['param_n_neighbors'])

#print(results)

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
#K =  results['param_n_neighbors']

plt.plot(tuned_parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
#plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

plt.plot(tuned_parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
#plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

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

```
#https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.xscale.html:
plt.xscale('log')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel(" Average Train/CV scores")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```

```
{'alpha': [0.0001, 0.01, 1, 100, 10000]}
SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

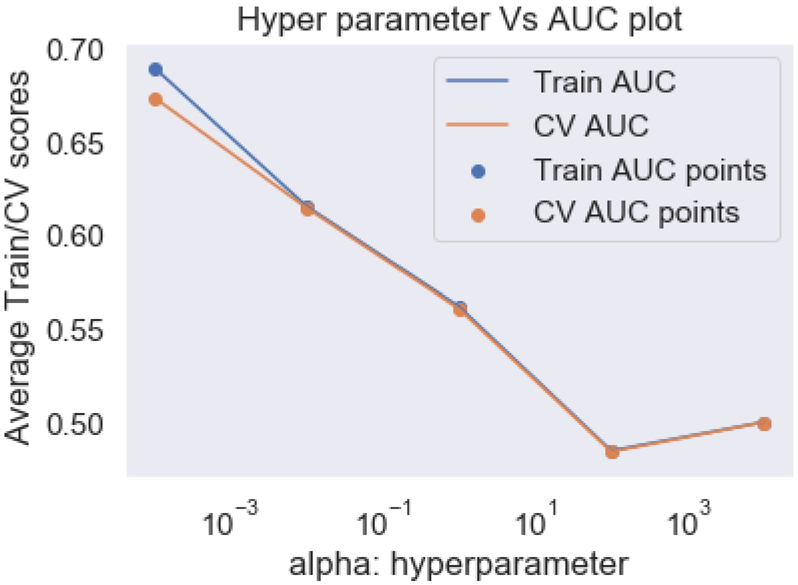


In [97]:

```
#code source: http://occam.olin.edu/sites/default/files/DataScienceMaterials/machine_learning_lecture_2/Mach

from sklearn.model_selection import train_test_split,GridSearchCV
#from sklearn.grid_search import GridSearchCV
from sklearn.datasets import *
from sklearn import linear_model
from sklearn.linear_model import SGDClassifier
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint


clf = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced')

tuned_parameters = {'alpha':[10**-4, 10**-2, 10**0, 10**2, 10**4]}
print(tuned_parameters)
model = GridSearchCV(clf, tuned_parameters, cv=5, return_train_score = True,scoring='roc_auc')
print(clf)
model.fit(X_train_set5, y_train)


results = pd.DataFrame.from_dict(model.cv_results_)
#results = results.sort_values(['param_n_neighbors'])

#print(results)

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
#K =  results['param_n_neighbors']

plt.plot(tuned_parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
#plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

plt.plot(tuned_parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
#plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

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

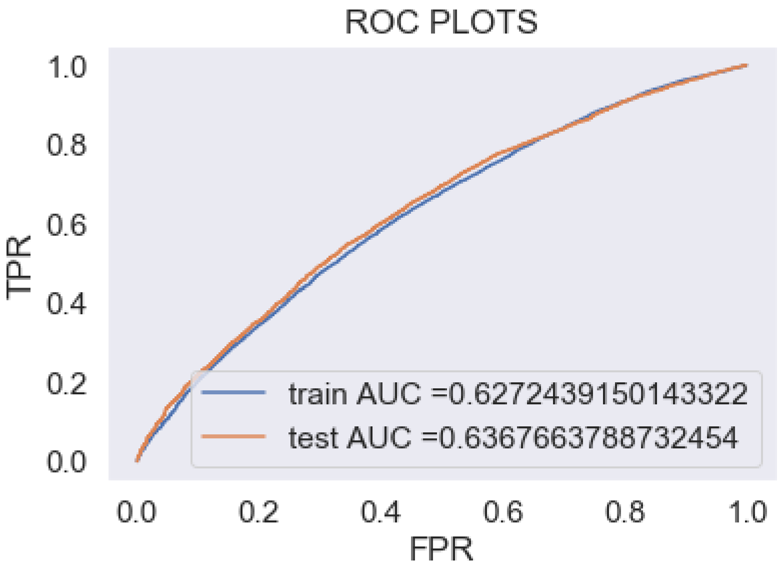
```
#https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.xscale.html:
plt.xscale('log')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel(" Average Train/CV scores")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```

```
{'alpha': [0.0001, 0.01, 1, 100, 10000]}
SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l1',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```



In [99]:

```
from sklearn.metrics import roc_curve, auc


clf = SGDClassifier(loss = 'hinge', penalty = 'l2',alpha=0.1,class_weight='balanced')
clf.fit(X_train_set5, 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 = clf.decision_function(X_train_set5)
y_test_pred = clf.decision_function(X_test_set5)

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("FPR")
plt.ylabel("TPR")
plt.title("ROC PLOTS")
plt.grid()
plt.show()
```



In [100]:

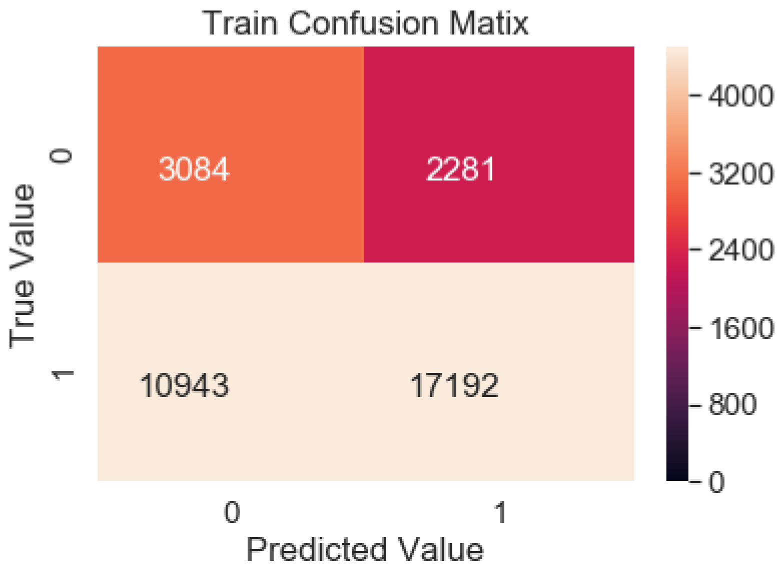
```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
#https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels/48018785
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.35125630302971084 for threshold 0.702
Train confusion matrix
[[ 3084  2281]
 [10943 17192]]
Test confusion matrix
[[1531 1111]
 [5273 8585]]
```

In [101]:

```
#https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels/48018785
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as sns
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_train,predict_with_best_t(y_train_pred,best_t)))
sns.set(font_scale=1.4)
fig = plt.figure()
ax = fig.add_subplot(111)
#ax.set_aspect(1)
#fig, ax = plt.subplots(figsize='3')
print(train_confusion_matrix)
akws = {"ha": 'right',"va": 'top'}

sns.heatmap(train_confusion_matrix, annot = True, annot_kws=akws, fmt = 'g',vmin=0.0 , vmax=4500)

plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Train Confusion Matix")
plt.show()
```

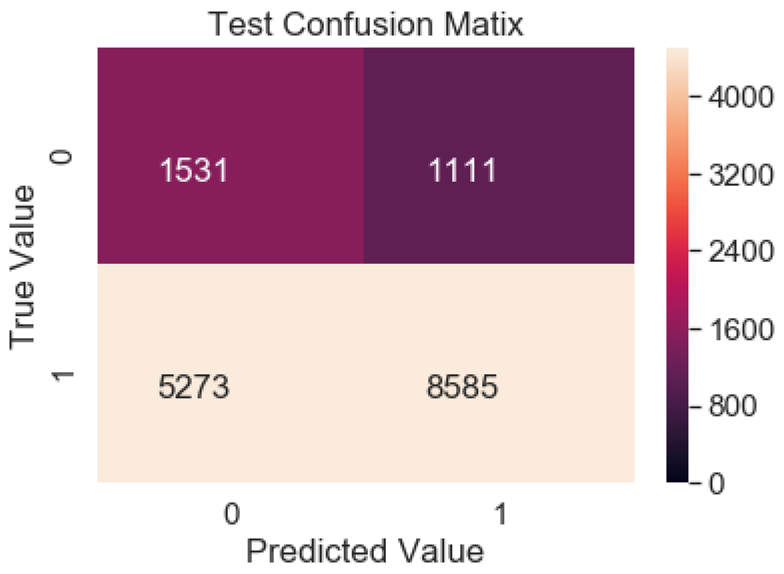
```
       0      1
0   3084   2281
1  10943  17192
```



In [102]:

```
#https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels/48018785
#https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels/48018785
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as sns
train_confusion_matrix = pd.DataFrame(confusion_matrix(y_test,predict_with_best_t(y_test_pred,best_t)))
sns.set(font_scale=1.4)
fig = plt.figure()
ax = fig.add_subplot(111)
#ax.set_aspect(1)
#fig, ax = plt.subplots(figsize='3')
print(train_confusion_matrix)
akws = {"ha": 'right',"va": 'top'}

sns.heatmap(train_confusion_matrix, annot = True, annot_kws=akws, fmt = 'g',vmin=0.0 , vmax=4500)

plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Test Confusion Matix")
plt.show()
```

```
      0     1
0  1531  1111
1  5273  8585
```



3. Conclusion

In [103]:

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

#http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()

x = PrettyTable()
x.field_names = ["Vectorizer_method", "Hyperparameter", "AUC"]
x.add_row(["Set-1",10 ,0.5])
x.add_row(["Set-2", 1 , 0.5])
x.add_row(["Set-3", 10**-1 , 0.5])
x.add_row(["Set-4", 10**-1 , 0.6])
x.add_row(["Set-5", 10**-1 , 0.5])

print(x)
```

```
+-------------------+----------------+-----+
| Vectorizer_method | Hyperparameter | AUC |
+-------------------+----------------+-----+
|       Set-1       |       10       | 0.5 |
|       Set-2       |       1        | 0.5 |
|       Set-3       |      0.1       | 0.5 |
|       Set-4       |      0.1       | 0.6 |
|       Set-5       |      0.1       | 0.5 |
+-------------------+----------------+-----+
```

In [ ]: